



تکنولوژی کرای انجین

رؤیای یک برنامه نویس

مؤلف: احمد کرمی

سرشناسه	:	کرمی، احمد، ۱۳۶۴-
عنوان و نام پدیدآور	:	رویای یک برنامه‌نویس، تکنولوژی کرای انجین / مؤلف احمد کرمی.
مشخصات نشر	:	بوکان : زانکو، ۱۳۹۷.
مشخصات ظاهری	:	۶۳۴ص.
شابک	:	۹۷۸-۶۰۰-۹۹۳-۰۷۲-۲
وضعیت فهرست نویسی	:	فپا
موضوع	:	بازی‌های ویدئویی
موضوع	:	Video games
موضوع	:	سی ++ (زبان برنامه‌نویسی کامپیوتر)
موضوع	:	C++ (Computer program language)
رده بندی کنگره	:	GV۱۴۶۹/۳۷/۵۴٫۹ ۱۳۹۷
رده بندی دیویی	:	۷۹۴/۸
شماره کتابشناسی ملی	:	۵۳۰۶۲۶۷



آدرس ناشر: بوکان، خیابان مولوی شرقی - داخل بازار تاریک - پاساژ عابدی

صندوق پستی: ۱۳۱-۵۹۵۱۵

مرکز پخش: ۰۹۱۴۳۸۲۲۱۳۹ (حامد قدرتی)

نام کتاب: تکنولوژی کرای انجین

مؤلف: احمد کرمی

ناشر: بوکان - انتشارات زانکو

نوبت چاپ: اول - تابستان ۱۳۹۷

تیراژ: ۱۰۰۰ نسخه

چاپ: بوکان، رامان

نرخ: تومان

کلیه حقوق چاپ و نشر برای مؤلف محفوظ است.



کسی نیست جز خودم که خودم نیستم جز، چیست؟

آنکه آمده من بودم و آنکه رفته من بودم خدا چیست؟

جز آن است که جز نیستی و هست رازی از خداست

خدا آن که ست که ز ازل و ابد را بی منتهای انتهاست

(احمد کرمی بوکانی)



تقدیم به پدر فداکار و صبورم که رفت

تقدیم به نفس های مادر دلسوزم که جاریست

تقدیم به برادر عزیزم و خواهران نازنینم

فهرست مطالب

صفحه	عنوان
۱	تاریخچه کرایتک و کرای انجین:
۴	مقدمه
۵	فصل اول: واژه‌شناسی و مفاهیم بازی‌سازی
۱۰	مقدمه
۳۶	فصل دوم: نصب CryEngine و پروژه GameSDK
۵۶	فصل سوم: محیط نرم افزاری CryEngine و ابزارهای مختلف آن
۸۵	فصل چهارم: نحوه نصب Visual Studio ۲۰۱۵ Enterprise برای CryEngine C++
۹۹	فصل پنجم: نحوه نصب CMake و ایجاد Solution ها برای Template های مختلف
۱۱۵	فصل ششم: کدهای راکتوری C++ و API های CryEngine
۱۶۴	فصل هفتم: ساخت Entity های جدید و Component های جدید
۳۲۳	فصل هشتم: ساخت یک پلیمر جدید برای بازی
۴۵۲	فصل نهم: هوش مصنوعی
۵۲۲	فصل دهم: کاربردهای Flowgraph در جلوه های تصویری دوربین و ساخت منوهای بازی
۵۷۲	فصل یازدهم: زبان جدید Schematyc و محیط ادیتوری آن
۶۱۸	فصل دوازدهم: خروجی گرفتن از بازی و



تاریخچه کرایتک و کرای انجین:

کرای انجین در سال ۱۹۹۸ توسط برادران ترک زبان اهل ترکیه بنیان گذاری شد و این سه برادر شرکتی را با نام کرایتک در یک استارت آپ فقط با داشتن چند صد دلار تاسیس کردند و در تمام این سال ها خدمات بزرگی را به صنعت بازی سازی جهان عرضه کردند، این شرکت با انتشار کرای انجین در کشور آلمان عملاً رقیبی جدی برای موتورهای بازی سازی ۲ امریکایی و اروپایی شده است، در چند دهه گذشته انقلاب های مختلفی در صنعت بازی سازی جهان رخ داده است و شرکت کرایتک از سال ۱۹۹۸ همیشه مردم و دیگر شرکت ها را غافلگیر کرده است، اولین بازی کرای انجین با نام Farcry منتشر شد و با عرضه این بازی موفق در نسخه ۱، شرکت Ubisoft یک کپی از کرای انجین خریداری کرد و موتور بازی سازی Dunia Engine را ساخت و از آن به بعد بازی های فارکرای در نسخه های مختلف با Dunia Engine طراحی شدند، در واقع دنیا انجین همان کرای انجین تغییر یافته یوبی سافت است، بازی های مختلفی با کرای انجین منتشر شد و مهمترین آنها بازی های سری کرایسیس بود و اوج شکوفایی و عظمت در کرایسیس ۳ با کرای انجین ۳ به بازی-دوستان جهان عرضه شد، این بازی بیشترین گرافیک و قدرت رندرینگ را برای دستگاه های پلی استیشن و کامپیوتر می خواست و تنظیمات آن در حالت ultra باعث میشد هر کامپیوتری نتواند این بازی را اجرا کند و جمله معروف Can it run Crysis را پدید آورد، بازی های دیگری نیز توسط دیگر شرکت ها با کرای انجین مانند بازی Star Citizen توسط شرکت Cloud Premium ساخته شد، با انتشار بازی Ryse و عدم

استقبال مردم از آن بازی، کرایتک بیشتر منابع مالی خود را به خاطر این بازی و دستمزد برنامه نویسان از دست داد و تعدادی از استودیوهای این شرکت در سراسر جهان تعطیل شدند، اگرچه مدیران کرایتک هرگز به وضعیت وخیم این شرکت اعتراف نکردند اما فیدبک های کاربران نشان میداد که کرایتک در حال ورشکسته شدن است، آیا واقعا کرایتک و کرای انجین تمام شد؟ در سال ۲۰۱۵ در اخبار غیر رسمی اعلان شد که شرکت آمازون به مبلغ ۷۰ میلیون دلار به شرکت کرایتک برای خرید سورس کد کرای انجین ۳,۸,۶ پرداخت کرده است، کرایتک قبلا نیز لایسنس ها و سورس کدهای کرای انجین را به شرکت های معماری آلمانی و فرانسوی فروخته بود، اما این رقم از طرف آمازون رویایی بود، این موضوع باعث شد که موتور بازی سازی آمازون در سال ۲۰۱۶ با هسته کرای انجین و رابط کرای انجین ۳,۸,۶ و علاوه بر آن اضافه شدن منوهای جدید با نام لامبریارد نسخه ۱ منتشر شود، اگرچه هنوز هم بسیاری از مردم معتقدند که کرای انجین کرایتک بهتر از کرای انجین آمازون است و اخبار دیگری نیز منتشر شده بود که دولت ترکیه مبلغ صدها هزار دلار در شرکت کرایتک سرمایه گذاری کرده است و تصاویری از بنیانگذاران شرکت کرایتک و رئیس جمهور ترکیه در فضای مجازی منتشر شده است، با انتشار کرای انجین ۵ مدیران روابط عمومی کرایتک در اخبار غیر رسمی و در فروم ها اعلام کردند که ما به هیچ وجه مشکلات مالی نداریم و همه این ها شایعه است و ما با قدرت کرای انجین را توسعه خواهیم داد، کرای انجین ۵ منتشر شد و با آن دو بازی با سبک جدید ساخته شده و انقلاب

دیگری را رقم زد، بازی های Robinson the Journey و Climb

در حال حاضر کرای انجین ۵,۴ منتشر شده است و برنامه نویسان کرایتک بیشترین بهینه سازی در سطح کدها در کرای انجین لحاظ کرده اند، جالب است بدانید که شروع پروژه زبان سیماتیک توسط پل سلینگر اهل امریکا

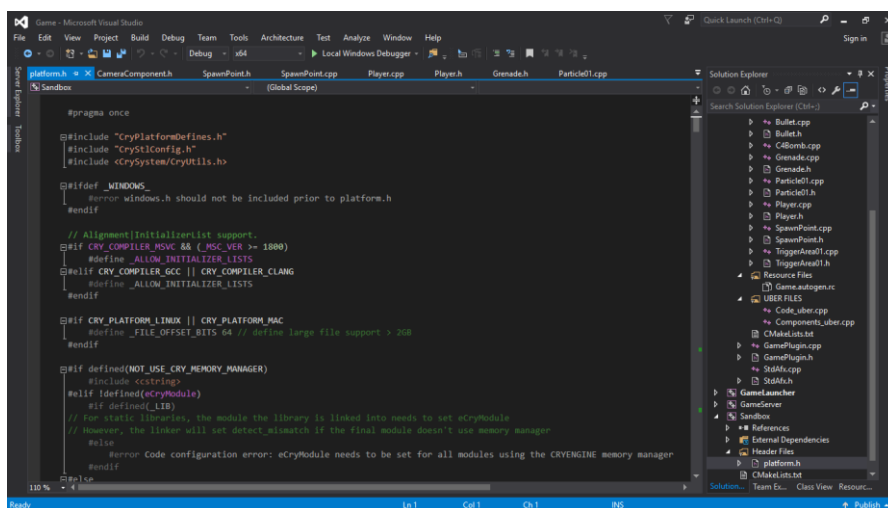
انجام شد و این ابزار نودبیزی در بازی Hunt به کار گرفته شد و هم اکنون زبان سیماتیک در کرای انجین قابل استفاده است، کرایتک اعلام کرد که یک سیستم جدید مالی به صورت مجازی ایجاد کرده است که شما می توانید با آن اسلحه، مهمات، سلامتی پلیمر و غیره را خریداری کنید، این سیستم پولی جدید با نام کرای-کش (CryCash) شناخته شده است و قرار است تا اواخر سال ۲۰۱۹ در کلیه بازی ها مورد استفاده شود و سیستمی جدید برای پرداخت درون برنامه ای برای بازی های ساخته شده با کرای انجین باشد، این سیستم هم اکنون در حال تست شدن است.

دولت ترکیه با تاسیس مدرسه ای و همکاری های آموزشی در کاربرد دروس تحصیلی معماری- کامپیوتر- هنر- گرافیک- ریاضی و فیزیک و غیره با شرکت کرایتک قراردادهایی را منعقد کرده است، این تحول بزرگ برای ورود کرای انجین به دیگر کشورهای خاورمیانه مانند ایران در کاربرد دروس تحصیلی است، کرایتک قصد دارد از پروژه های تیم های مستقل بازی سازی تا تک نفری پشتیبانی آموزشی را به عمل آورد و همانطور که مدیران روابط عمومی این شرکت اعلام کرده اند، اخبار جدی از آموزش های کرای انجین در راه است، بهترین پشتیبانی از آموزش ها در این موتور را می توان به زبان ++C اشاره کرد، این نقطه عطفی برای کشف اسرارهای زیادی در رابطه با این موتور است، به دنیای کرای انجین خوش آمدید.

مقدمه

۱- موتور بازی سازی یا **گیم انجین ها** برای ساخت انواع بازی های کامپیوتری و موبایل طراحی شده است و کرای انجین بهترین موتور بازی سازی جهان است.

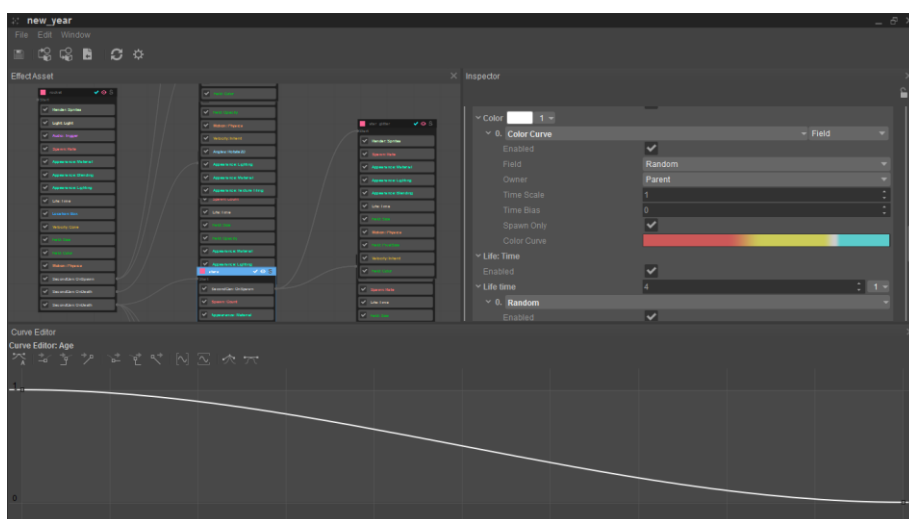
۲- زبانی برنامه نویسی همه منظوره که از طراحی سیستم عامل ماهواره گرفته تا بازی های کامپیوتری با آن ساخته می شود، با یادگیری این زبان شما در واقع کلاس های خلاقیت را می گذرانید.



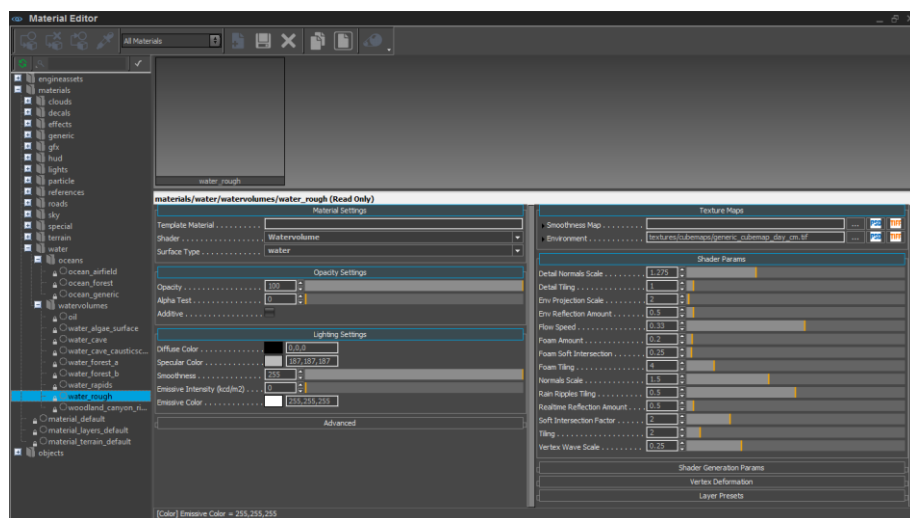
فصل اول

واژه‌شناسی و مفاهیم بازی‌سازی

- ۱- پروژه معادل جریانی از انجام یک کار گروهی با تفکری پایه و اساسی در تجارت الکترونیک مورد بحث اقتصاد جهانی با محوریت قانون کپی رایت است
- ۲- پلاگین در واقع امکانات افزودنی است تا از قابلیت موتور بازی سازی بیشتر استفاده شود و عملاً قدرت یک موتور بازی سازی نیز با انتشار پلاگین‌های مختلف برای کاربران مورد سنجش قرار می‌گیرد تا بازی سازان بیشتری به طرف موتور بازی سازی مهاجرت کنند.
- ۳- سیستم ذرات در محیط پارتيكل اديتور در منوی Tools قابل دسترس است و شما می‌توانید با نودها (گراف‌ها) جلوه‌های ویژه مختلفی را ایجاد کنید، مباحث سیستم ذرات خارج از بحث این کتاب است و کتاب مفصل دیگری را می‌طلبید، کرای انجین الگوهای (Templates) پیش فرضی را در این اديتور به همراه دارد که طراحی سیستم‌های ذرات دیگر را سهل می‌نماید.



۴- متریال‌ها با استفاده از محیط متریال ادیتور ساخته می‌شوند، متریال ادیتور با اوج خلاقیت ساخته شده است و کاملاً پارامتریکال است و در آینده متریال ادیتور نودبیزی (گراف‌ها) منتشر خواهد شد، مباحث متریال ادیتور کتاب مفصل و جداگانه‌ای را می‌طلبد.



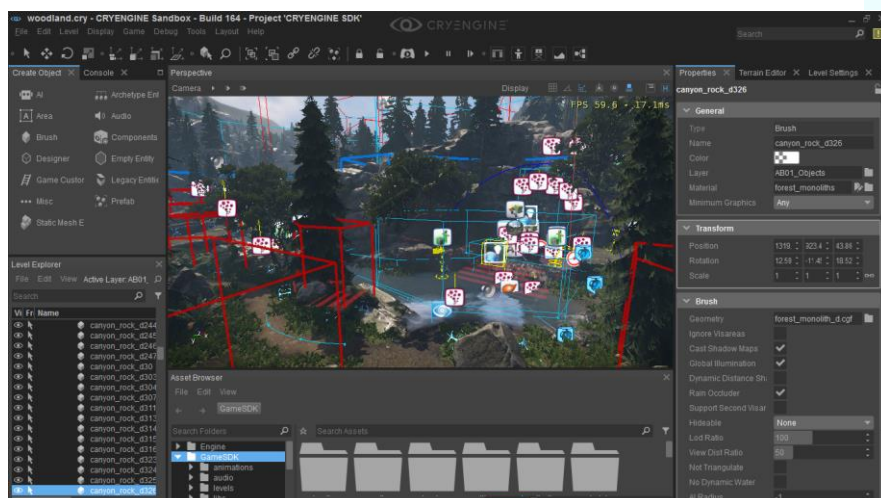
۵- شیدرها مباحث گسترده‌ای از کدنویسی GPU را در بر می‌گیرد، شرکت‌های NVIDIA – AMD – INTEL تحقیقات گسترده‌ای را برای بهینه‌سازی کدهای شیدری انجام داده‌اند اما بیشترین و موثرترین تحقیقات مربوط به کیهاس‌گروپ است که باعث بنیانگذاری مجموعه‌ای از کدهای جدید رندرینگ با قابلیت مشترک و اجرا بر روی دستگاه‌های موبایل با سیستم عامل مختلف و مجموعه سیستم عامل‌های مختلف کامپیوتر را در بر می‌گیرد، این کدها که با نام Vulkan API شناخته شده است ارتباطات تنگاتنگی با کدهای شیدری دارد، API‌های DirectX تنها برای سیستم عامل‌های ویندوز، API‌های OpenGL برای سیستم عامل لینوکس و ویندوز و API

های Metal برای سیستم عامل مکینتاش است و با ارائه سیستم جدید رندرینگ Vulkan API افق بسیار روشنی برای ساخت بازی های مختلف بر روی دستگاه های مختلف سخت افزاری پدیدار شده است، کرای انجین از جدیدترین نسخه این API ها پشتیبانی می کند، مباحث کدهای شیدری کتاب های مختلفی را می طلبد و در این کتاب مباحث آن نمی گنجد.

۶- زبان گرافیکی که با نام های زبان های نودبیزی (Node Base) (گره ها را به هم متصل کنید) یا زبان های ویژوال اسکریپتینگ (Visual Scripting) (به صورت بصری کدنویسی کنید) نیز شناخته می شود، به زبانی گفته می شود که بدون کدنویسی می توانید بازی هایتان را بسازید اما اگر بازی تان بسیار بزرگ شود مانند حجم مرحله در سطح یک شهر، زبان برنامه نویسی ++C پیشنهاد می شود، زیرا کدنویسی در صدها هزار خط کد آسان تر از اتصال چند هزار نود یا گره به هم دیگر است.

۷- Stostrup : باعث اضافه کردن مفاهیم کلاس، ارث بری، وراثت، چندریختی و تمپلت ها و... در زبان C شد، در واقع پیشرفت های دنیای سرگرمی بازی و فیلم، صنایع مختلف فضایی و دفاعی، سیستم های توزیع شده و بلادرنگ و غیره مدیون پروفیسور استواستراپ طراح زبان ++C است، او زبان C را توسعه داد و ++C را بنیان گذاری کرد

۸- به محیط نرم افزاری تکنولوژی کرای انجین سندباکس (Sandbox) گفته می شود، آنجایی که شرکت کرایتک، توسعه دهنده موتور بازی سازی کرای انجین به انتشار کامل سورس کد کامل کرای انجین در آدرس اینترنتی زیر پرداخته است، شما می تونید نیز محیط ادیتور سندباکس را توسعه دهید



۹- به مجموعه امکانات اضافه شده در هر نسخه یا آپدیت جدید که قرار است منتشر شود در یک بازه زمانی مشخص را نقشه راه (Roadmap) گفته می‌شود، همه ی شرکت ها باید یک نقشه راه داشته باشند و هر چند ماه یکبار آن را به روز رسانی کنند و نقشه راه کرایتک برای کرای انجین در آدرس اینترنتی زیر است :

۱۰- زمان طراحی مرحله به زمانی که شما در حال ساخت مرحله و اضافه و کم کردن، تغییر اندازه، تغییر موقعیت، تغییر چرخش اشیاء یا کشیدن اشیاء هستید و باعث ساخت موانع طبیعی یا مصنوعی مانند یک شهر جنگ زده یا جنگل یا کوهستان هستید، گفته می‌شود.



مقدمه

دنیای امروز دنیای علم - مهارت و کسب درآمد هزاران میلیارد دلار است و بیشتر کشورهایی با ذخایر نفتی و گازی با اسم کشورهای جهان سوم شناخته می‌شوند، در اروپا و آمریکا ذخایر نفتی و گازی جزء استراتژی‌های نوین برای آینده به حساب می‌آیند و اکنون این کشورها تکیه بر اقتصاد غیر نفتی و غیر گازی زده و به صورت نمایی علم شان و مهارتشان در حال رشد است مانند کشور ژاپن که هیچ منبع نفتی و گازی ندارد.

صنعت بازی‌سازی یکی از این قطب‌های اقتصادی بر پایه علم و مهارت است و این کشورها به خوبی آینده را ترسیم کرده‌اند، صرف نظر از توضیحات و علم درهم تنیده و پیچیده مورد اشاره این مهم در خطوط بالا، دانش کامپیوتر و خصوصا صنعت بازی‌سازی به زبان برنامه‌نویسی محبوب و بسیار قدرتمندی گره خورده است و آن زبان ++C است، زبانی که از سیستم عامل مایکروسافت گرفته تا زیردریایی‌های اتمی و کاوشگرهای فضایی ناسا با این زبان طراحی شده‌اند و می‌توان به جرات گفت که اگر برنامه‌نویسانی که در صنعت بازی‌سازی به زبان برنامه‌نویسی ++C مسلط نباشند، مدرک تحصیلی و دانشگاهی آنها زیر سوال است.

زبان برنامه‌نویسی ++C عمیقا در موتور بازی^۱ سازی کرای انجین ۵ مورد استفاده قرار می‌گیرد و این موتور با این زبان طراحی شده است و برای طراحی بازی‌ها با این موتور از همین زبان استفاده می‌شود، کرای انجین ۵ قدرتمندترین موتور بازی‌سازی دنیا است و بعضی از موتورهای دیگر نیز مانند دنیا انجین از شرکت یوبی سافت، لامبریاردا از شرکت آمازون از موتور ساخت

بازی کرای انجین توسعه یافته اند و بازی های قابل قبولی با موتور بازی سازی کرای انجین ساخته شده اند : کرایسیس ۳، بازگشت به جزیره دایناسورها، سفر رابینسون و...

آنچه که به رشته تحریر درآمده، با سال ها تجربه در تحقیق و پژوهش بدست آورده ام و گویای آن است که برای کسانی که در صنعت بازی سازی فعالیت می کنند، پیامی است که باید زبان ++C^۲ را بدانند، زبان های دیگری نیز وجود دارند که با میزان رشد ++C نمی توانند رقابتی داشته باشند، این کتاب برای کسانی است که در زبان برنامه نویسی ++C حرفه ای هستند اگرچه تمام تلاشم را انجام داده ام تا این کتاب نیز برای مبتدیان کاملاً مفید باشد، به دنیای ++C کرای انجین خوش آمدید، این دنیا به شما می آموزد کلاس های خلاقیت، پرورش ذهن و مغز را پاس کنید، در این راه قدم بردارید و از آنچه می آموزید لذت ببرید، امروز عصر مهارت و عمل گرایی است، به دیگران این کتاب را پیشنهاد دهید، این کتاب برای معلمان، اساتید دانشگاه ها، دانشجویان، دانش آموزان و کارآموزان نوشته شده است تا بهتر با کاربرد درس تحصیلی در رشته های مختلف، ریاضی فیزیک، معماری، کامپیوتر، گرافیک، موسیقی، انیمیشن آشنا شوند و در واقع مجموعه ی این رشته ها است که ارتباط صنعت با دانشگاه و مدارس را معنا دار می کند، این کتاب گام نخست برای تبیین و مبین شدن صنعت بازی سازی برای جامعه است و پتانسیلی برای رشد همه دوستاران صنعت بازی سازی کشورمان است، در نگاه دیگر مطمئناً منابع کلان درآمدزایی ارزی و ریالی به تجارت الکترونیک و قانون کپی رایت با ارتباط تنگاتنگی واقع است و این منابع به قدری کلان و غیر قابل چشم پوشی است که شرکت کرایتک سیستم جدید مالی با نام CryCash را بنیان گذاری کرده است و در اواخر سال ۲۰۱۹ از آن رو نمایی خواهد کرد و فیدبک های این

سیستم در بازی وارفیس زبان ترکی آغاز شده است، کلام من خطاب به دلسوزان صنعت بازی سازی کشورمان است که جایگاه سیستم جدید مالی در بازی های ایرانی با کسب درآمدهای نجومی کجاست؟ اسپانسرهای بزرگ تبلیغاتی کشور برای خدمت به صنعت بازی سازی کجا هستند؟ آیا بنیاد ملی بازی های رایانه ای به وظیفه خود درست عمل کرده است؟ آیا فیدبک های بازی های ایرانی در سیستم عامل ها یا پلتفرم های ویندوز و پلی استیشن و ایکس باکس مثبت بوده است؟ و سوالات دیگری که در این مقدمه نمی گنجد.

آنچه که مشخص است، این کتاب نقطه عطف دیگری را در زندگی من رقم زد تا بیشترین تلاش را برای خدمت به صنعت بازی سازی کشورم انجام دهم و خداوند را شاکرم که همیشه به من یاری رسانده است و همچنین در کنار آن، بیشترین کمک خانواده ام باعث شده که در عرصه علمی و آموزشی، انسانی خدمتگزار و موفق باشم.

این کتاب در درجه اول برای کلیه علاقه مندانی است که دوست دارند، بازی های خود را به زبان برنامه نویسی ++C در موتور بازی سازی کرای انجین ۵,۴ آغاز کنند و در درجه دوم با امکانات و ابزارهای مختلف در کرای انجین ۵,۴ آشنا شوند، از آنجایی که این کتاب مهارت ساخت بازی ها را با دانش نامه ای بلند از توابع ++C را در خود دارد، خوانندگان و مرجوعین به این کتاب باید قبلا دانش کافی و در حد متوسط از زبان برنامه نویسی ++C را بدانند و این کتاب برای علاقه مندانی که برنامه نویسی ++C را نمی دانند و حتی مبتدی هستند، نوشته نشده است، اگرچه با توضیحات کافی و مبین سعی شده است که مبتدیان نیز از این کتاب بیشترین بهره را ببرند، لازم به ذکر است که در این کتاب در بخش برنامه نویسی بصری سیماتیک کرای انجین ۵,۳ و ۵,۴

را مد نظر گرفته ام و بقیه فصل‌ها مرتبط با کرای انجین ۵,۴ است، امیدوارم با توجه به ۱۰ سال برنامه نویسی این حقیر، کلیه خوانندگان ارجمند هرگونه انتقاد، پیشنهاد، سوالات تان را از راه‌های زیر در جریان بگذارند تا آثار دیگری را پر بارتر منتشر نمایم.

احمد کرمی بوکانی ۱۳۹۷/۰۲/۰۹

پروژه^۱: هر بازی معادل یک پروژه است و اسم پروژه در واقع اسم بازی است مانند بازی کرایسیس ۳ با نام پروژه کرایسیس ۳ نام برده میشود.

مرحله یا نقشه: هر بازی می تواند شامل چندین مرحله یا چندین نقشه (Map یا Level) باشد که قهرمان داستان باید آن را به پایان رساند

پلاگین^۲: برای افزایش قابلیت در موتورهای بازی سازی، برنامه نویس یا برنامه نویسانی که حرفه ای هستند، امکاناتی را طراحی می کنند تا طراحان بازی با کد کمتری بتواند بازی را طراحی کنند، مانند پلاگین پخش ویدئو در کرای انجین.

Marketplace: فروشگاه مجازی برای عرضه انواع منابع آماده مانند مدل های سه بعدی، کاراکترها یا شخصیت ها، صداها، درختان و گیاهان و غیره تا در بازی استفاده شود، فروشگاه کرای انجین به آدرس زیر است:

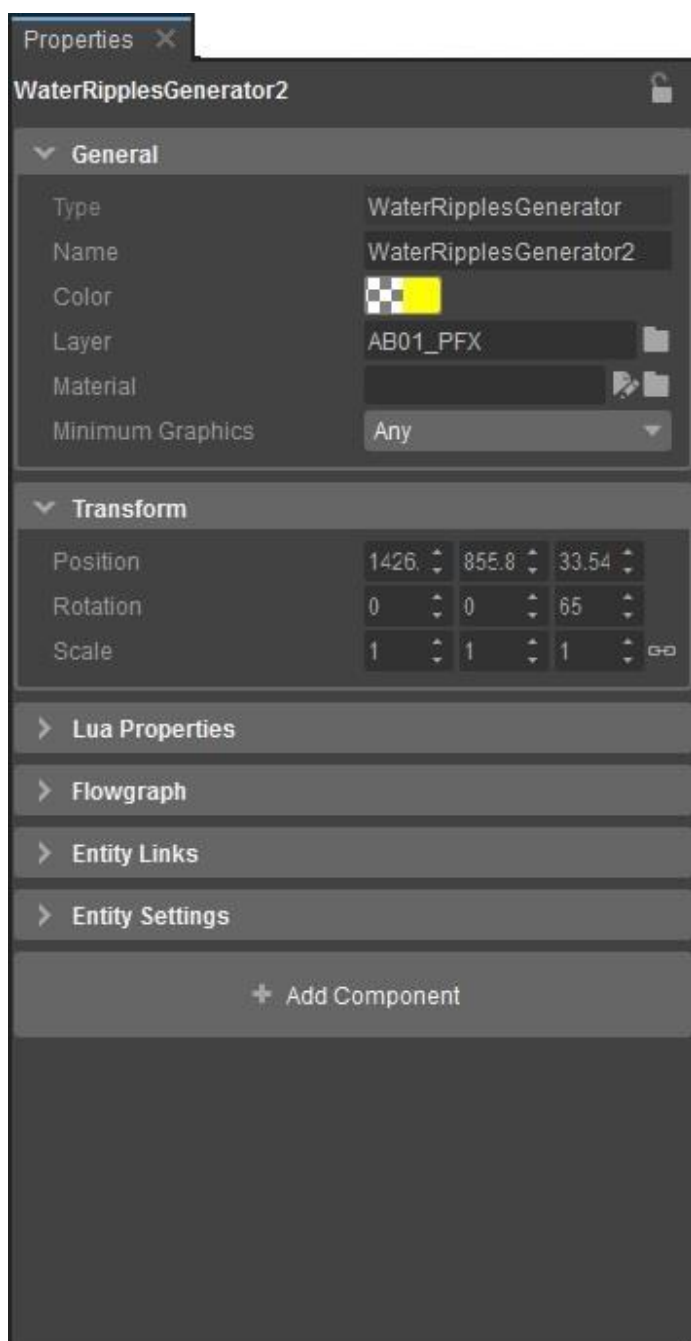
<https://www.cryengine.com/marketplace>

این منابع می تواند به صورت رایگان یا به صورت پولی ارائه شود تا کاربران آن را خریداری کنند و در بازی هایشان استفاده کنند

موجودیت یا شی (Entity): یک بازی شامل هزاران شی است و هر شی در بازی از نوع های مختلفی تشکیل شده است مانند درخت، خانه، دشمن، آتش، جاده، رودخانه و غیره که به هر شی از بازی اینتیتی (Entity) گفته می شود، مثلاً یک شی با نام نور حداقل از دو جزء یا کامپونت Transform و Light تشکیل شده است، در واقع به مجموعه چند کامپونت (Component) اینتیتی (Entity) هم گفته می شود، هر اینتیتی حداقل یک کامپونت با نام Transform با سه خصوصیت Position، Rotation و Scale است، به هر Entity شی یا آبجکت (Object) نیز گفته می شود



شی اشاره شده با پیکان زردرنگ در عکس یکی از هزاران شی در مرحله بازی است که وظیفه آن تولید امواج بر روی آب در کنار آبشار است، توجه داشته باشید که بعضی از اشیاء در مراحل بازی قابل رویت نیستند و این به معنی عمل نکردن شی در جهان بازی نیست، این اشیاء می‌توانند حامل قوانین فیزیک و ریاضی و هر چیز دیگری باشند، نمونه آن شی بالا در عکس مربوطه است، برخی از اشیاء نیز قابل رویت بوده که می‌توانند مثلاً مدل درختان یا بوته‌ها را در خود نمایش دهند، اشیاء دیگری نیز هستند که غیرفعال هستند که بر اساس تصمیم‌گیری طراح یا طراحان مرحله می‌توانند به صورت دائمی یا موقت غیرفعال شوند، مثلاً دشمنی که کشته شود، باید در حالت غیرفعال قرار بگیرد یا حتی می‌تواند از جهان بازی حذف شود.

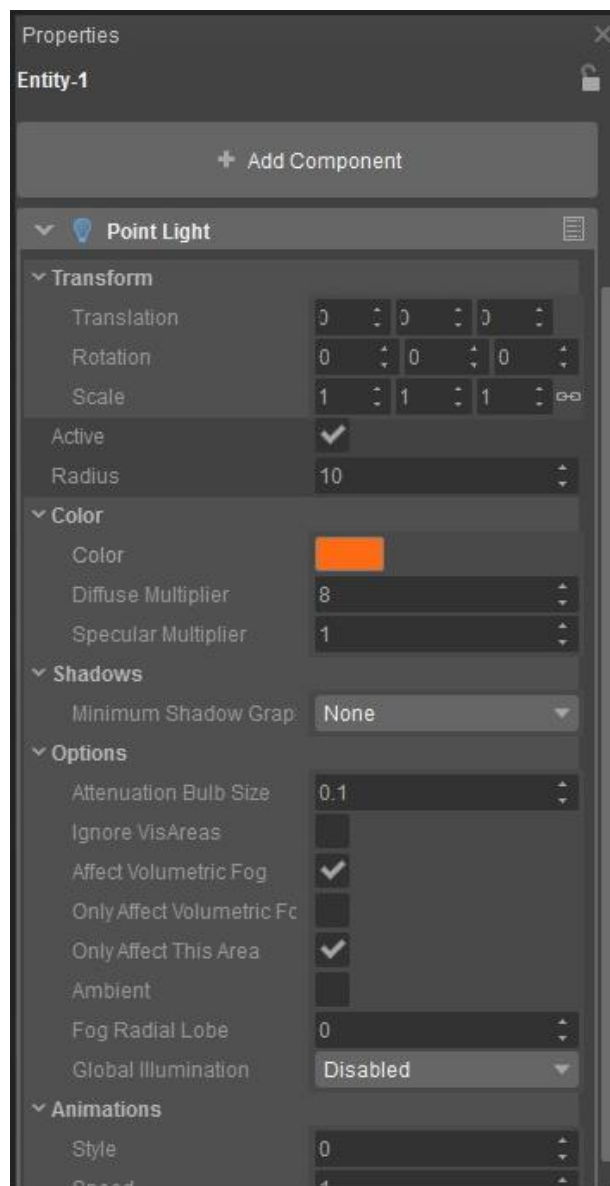


شی مربوطه در عکس با نام `WaterRipplesGenerator۲` دارای ۶ کامبونت است که `Transform` و `General` دو کامبونت از ۶ کامبونت را در این شی تشکیل می‌دهند.

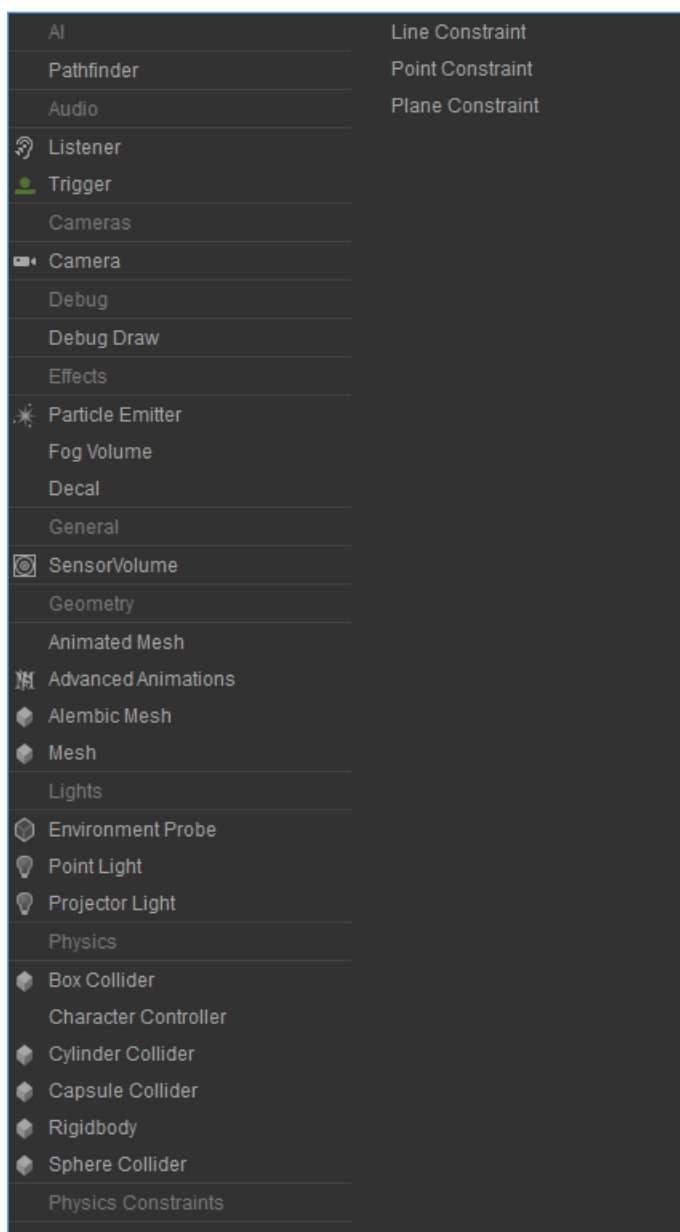
کامبونت `(Component)`: به هر جزء از یک اینتیتی `(Entity)` گفته می‌شود، مثلاً اینتیتی یا شی آتش از سه بخش یا سه جزء `Transform` (شرایط انتقال) `Particle – (ذرات)` و `Light` (نور) تشکیل شده است.



در میان درختان و صخره‌ها یک شی نور اضافه شده است که وظیفه آن روشن کردن محیط اطراف بر اساس کامبونت `Light` است



کامپونت Light در تصویر را می‌بینید که از پارامترهای مختلف تشکیل شده است و دکمه‌ای نیز با نام "Add Component" در بالای کامپونت Light وجود دارد که می‌توانید کامپونت‌های دیگری را به اینتیتی نور اضافه کنید.

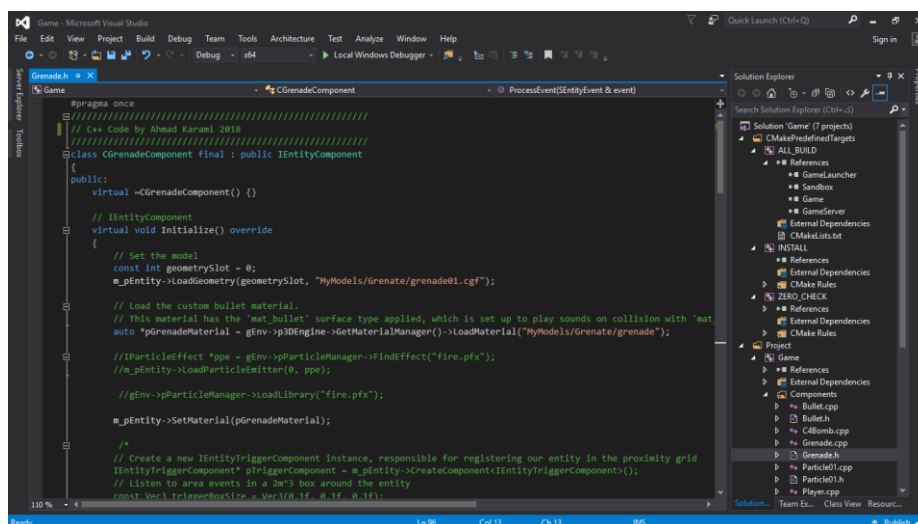


با توجه به عکس بالا، تعداد کامبونت‌های موجود با کدهای C++ می‌تواند توسعه پیدا کنند و شما می‌توانید کامبونت‌های بیشتری را به این

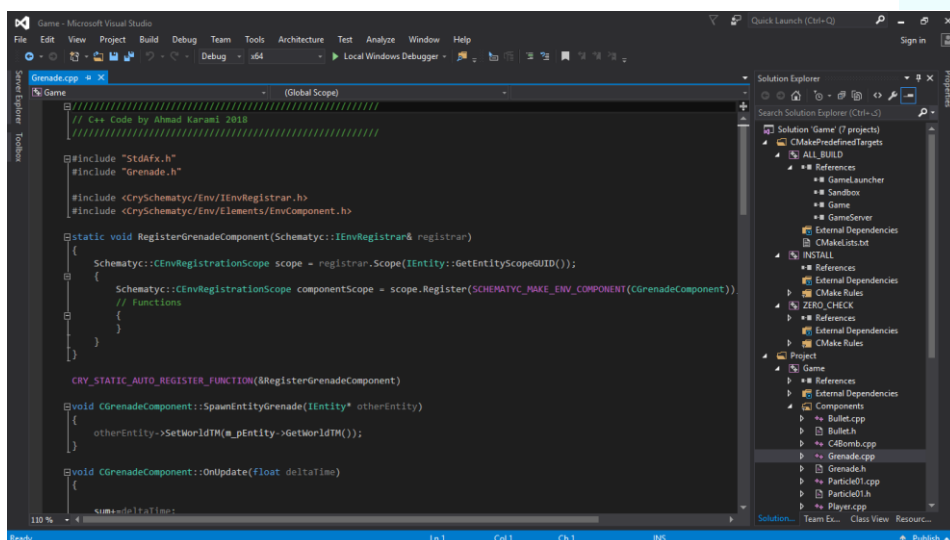
لیست با استفاده از زبان قدرتمند C++ اضافه کنید، در فصل‌های دیگر به این مهم خواهیم پرداخت.

در کرای انجین ۵،۵ تعداد کامبونت‌ها بسیار بیشتر از چیزی است که در عکس بالا می‌بینید مانند کامبونت باران، کامبونت‌های VR – کامبونت‌های هوش مصنوعی (AI)

هدر فایل (*.h) و سی‌پی‌پی فایل (*.cpp) : برای ساخت هر اینتیتی یا کامبونت‌ها باید از دو فایل هدر و سی‌پی‌پی که با کدهای برنامه‌نویسی C++ استفاده شود، بستگی به نوع رجیستر کردن این نوع کامبونت‌ها و استفاده آن در مرحله بازی تفاوت‌هایی در کدنویسی وجود دارد.



در تصویر بالا یک هدر فایل که وظیفه آن پیاده‌سازی رخدادهای شی را بر عهده دارد را می‌بینید، هدر فایل‌ها دارای پسوند h. هستند



در تصویر بالا یک فایل سی پی پی که وظیفه آن پیاده سازی توابع شی را بر عهده دارد را می بینید، فایل های سی پی پی دارای پسوند .cpp. هستند، لازم به ذکر است که پیاده سازی توابع به رخدادهای هدر فایل ها وابسته هستند و گام نخست باید رخدادهای مورد نظران را پیاده کنید و سپس به پیاده سازی توابع بپردازید، در این کتاب به این مهم خواهیم پرداخت.

پارتیکل (سیستم ذرات) ^۳: دنیای بازی مانند دنیای واقعی ما نیز شامل سیستم ذرات است و از مجموعه ای از نقاط برداری قابل اندازه گیری با خواص پارامتریکال گرافیکی قابل سنجش تشکیل شده است مانند: آتش، ریزگرد، دود، صاعقه، گرما، گردباد و غیره



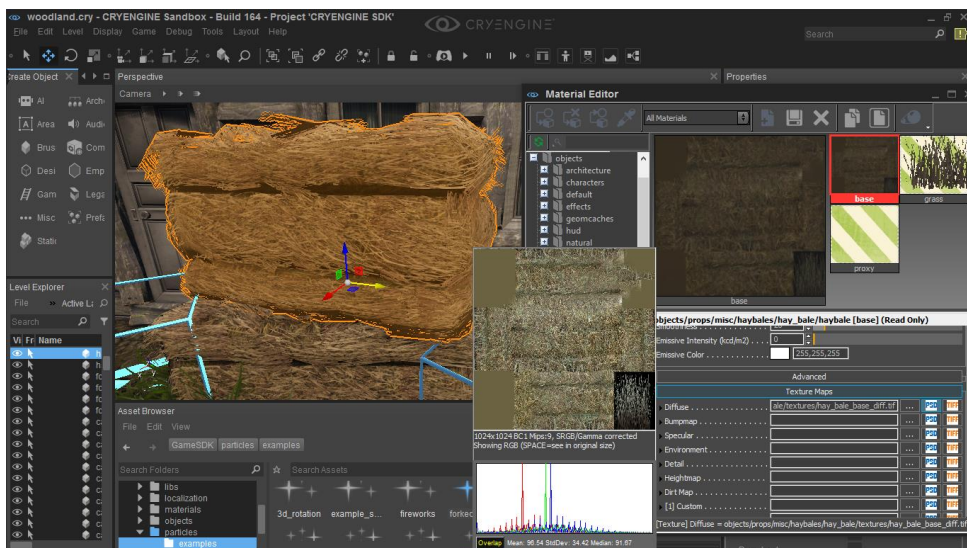
تصویر یک صاعقه با توجه به گراف یا نودها بر اساس پارامترهای مختلف ایجاد می‌شود، منشاء آن معادلات بصری ریاضی و فیزیک است و متریال‌هایی که ایجاد شده‌اند.

متریال (سیستم مواد) ^۴: در واقع مجموعه قوانین شیمی به نوع دیگری در این سیستم پیاده‌سازی می‌شود و هر متریال به تکسچرها و شیدرها وابستگی دارد، این مواد می‌تواند هر نوعی باشد مانند آب، روغن، آهن، گچ، گل و لای و غیره.



تصویر متريبال گل و لای ترکیب شده با سنگ که قوانین شیمی را به گونه ای دیگر نشان می دهد، نگران نباشید! فرمول های شیمی به صورت کدهای شیدری در GPU پیاده سازی می شوند و فرمول های شیمی را در اینجا نخواهید دید.

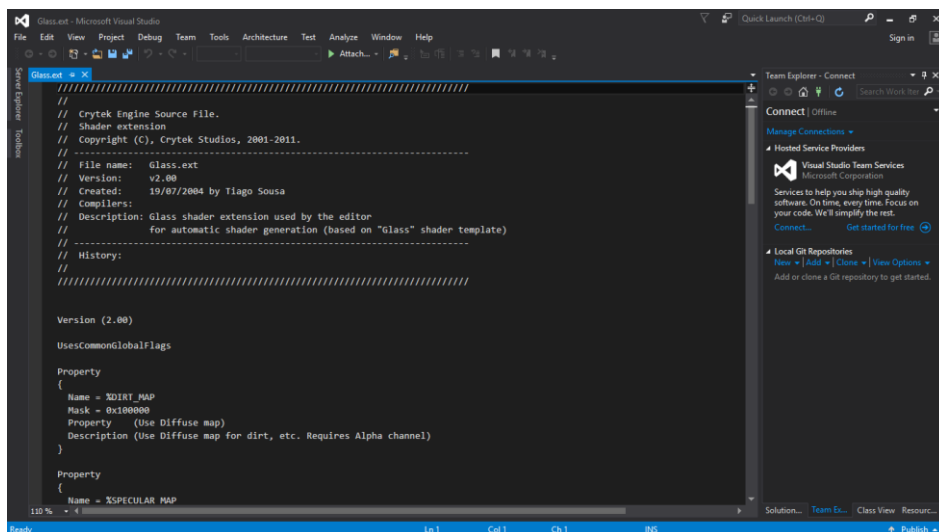
تکسچر : به بافت های تصویری نیز گفته می شود که بر روی مدل های سه بعدی نگاشت (کشیده و نقاشی) می شوند، یک جعبه را در نظر بگیرید، اگر تصویری از مهمات بر روی جعبه نقش بسته شود، جعبه مهمات خواهد بود و اگر تصویری از بستنی بر روی جعبه کشیده شود، جعبه بستنی خواهد شد.



در تصویر بالا می بینید که این جعبه با تکسچر کاه و گیاه نمایش داده شده است

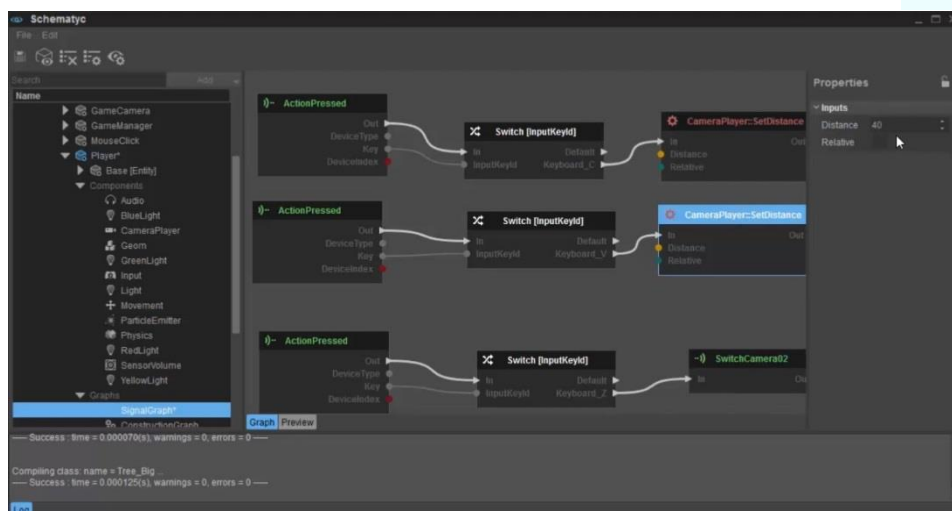
شیدرها^۵ : به مجموعه ای از کدهای آماده ای که قبلاً نوشته شده است و در سطح GPU کارت گرافیکی اجرا می شوند (Shader) و در حقیقت نقش سایه زنی و ترکیب زنی برای شبیه سازی مواد را آنجا می دهد، در واقع

مجموعه‌ای از کدهای شیدری و تکسچرها یک متریال یا چند متریال (Material) ساخته می‌شوند.



در تصویر بالا می‌بینید که با استفاده از کدهای شیدری، ماده یا متریال شیشه پیاده‌سازی شده است!

سیماتیک (Schematyc): جدیدترین زبان بهینه‌سازی شده گرافیکی^۶ در کرای انجین ۵,۳ و به بعد است که با این زبان بدون تایپ خط کد می‌توانید بازی‌هایتان را طراحی کنید، در واقع شما تنها با گراف‌ها در ارتباط هستید و هر آنچه که می‌سازید مانند یک اینتیتی در سطح بازی دیده می‌شود و در سطح پروژه مانند پریفب است.

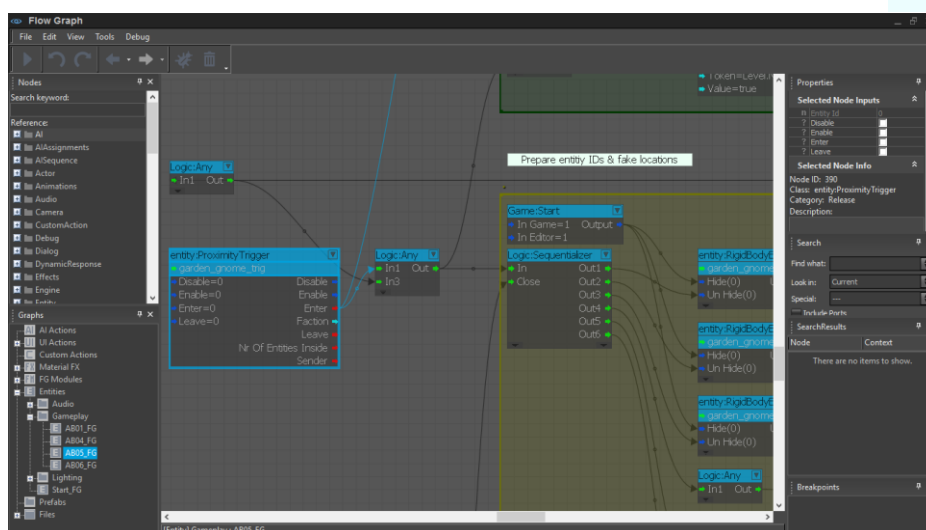


در تصویر بالا نمونه ای از گراف طراحی بازی ها بدون کدنویسی در کرای انجین ۵,۳ را می بینید،بهترین انتخاب برای ساخت بازی ها با گراف ها (بدون کدنویسی) انتخاب موتور بازی سازی کرای انجین ۵,۵ است، زیرا که سیماتیک در کرای انجین ۵,۴ بازسازی مجدد شد و به شما پیشنهاد میکنم که اگر قصد دارید بازی هایتان را با سیماتیک بسازید از سیماتیک کرای انجین ۵,۵ شروع کنید،اما برای شروع بهتر است حتما این کتاب را تمام کنید،تا درک بسیار بهتری از سیماتیک داشته باشید،اگرچه بازهم شرکت کرایتک به همه بازی سازان جهان توصیه کرده است،از زبان ++C در کرای انجین برای ساخت بازی ها استفاده کنید،از این جا دوباره معلوم می شود که زبان ++C یک اولویت انکار ناپذیر برای همه طراحان ساخت بازی است.

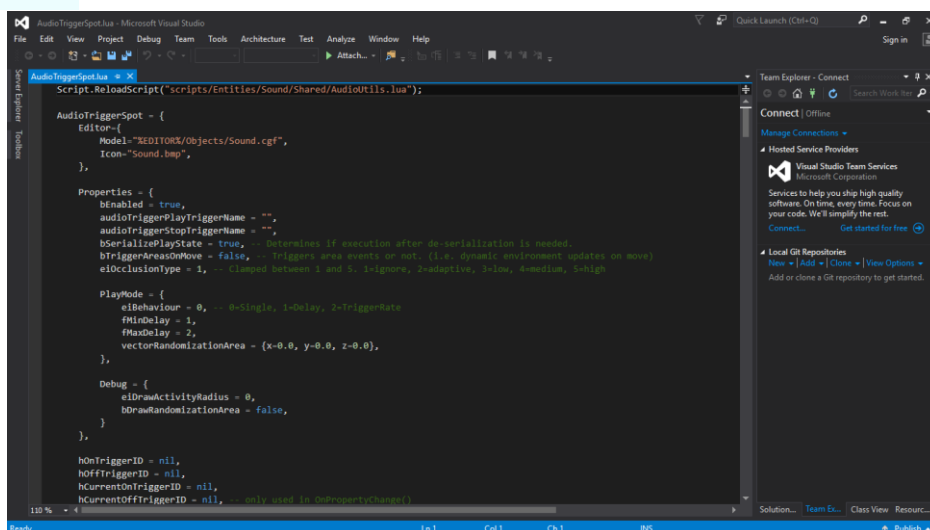
پریفب یا سیماتیک پریفب : قبل از ظهور زبان گرافیکی سیماتیک هر شی مرجعی که ساخته می شود با نام پریفب شناخته می شد یعنی شما فقط یکبار شی را می ساختید مانند دشمن و دشمنی را که طراحی می کردید،می توانستید هزاران بار استفاده کنید و البته همین دشمن را به تعداد هزاران نسخه در سطح مرحله یا مراحل استفاده کنید،در حقیقت پریفب ها از هر

نوعی که باشند جهت جلوگیری از کارهای تکراری است و شما فقط یکبار پریفب می‌سازید و سپس از آن بارها و بارها استفاده می‌کنید، پریفب‌ها بیشترین وابستگی به کدهای C++ را دارند و با ظهور سیماتیک شما پریفب‌ها را بدون کدنویسی می‌ساختید، اما مشکل اصلی پریفب‌های سیماتیک یا حتی زبان سیماتیک برای ساخت بازی‌های بزرگ تا این تاریخ از انتشار کتاب جوابگو نیست و اینجا است که C++ راه حل‌های زیادی را ارائه می‌دهد (استواستراپ^۷ فرد خطرناکی است)

اسکرپت فلوگرافی (Flowgraph) تا اسکرپت لوا (Lua) : همانند سیماتیک، زبان دیگری مانند فلوگراف (بدون کدنویسی) و لوا (کدنویسی) وجود دارد که در سطح مرحله و شی اتفاقات مختلفی را مانند انفجار می‌توان در بازی رقم زد، در واقع زبان‌هایی که اسکرپتی هستند، باعث آسانی کار طراح بازی یا طراح مرحله خواهند شد اما این انتقاد وجود دارد که زبان‌های اسکرپتی فلوگراف و لوا مستقل از زبان C++ عمل نمی‌کنند و وابستگی شدیدی به زبان C++ دارند و این مسئله باعث شد زبان اسکرپتی در سطح مرحله و شی بوجود آید و اسم آن زبان سیماتیک است، این زبان اگرچه به زبان C++ وابسته است اما با توسعه کامپونت‌های مختلف در سیماتیک در آینده عملاً استقلال خود را از C++ برای ساخت بازی‌های بزرگ بدست خواهد آورد، این زبان از سیستم جدید مدیریت Component Entity System استفاده می‌کند، جالب است بدانید که اگرچه می‌توانید با زبان C# و C++ اینتیتی‌های جدیدی را با ترکیب سیماتیک خلق کنید اما مسئله اینجاست که کدهای کرای انجین زبان سی شارپ در سطح API به فایل‌های DLL که با C++ نوشته شده وابسته است و دوباره اینجاست که قدرت C++ نمایان می‌شود.



یک عکس از گراف‌ها در سیستم برنامه‌نویسی فلوگراف را مشاهده می‌کنید، با توجه به فیدبک‌های کاربران از سراسر جهان این زبان احتمالاً در آینده از کرای انجین حذف خواهد شد، کرایتک تاریخ دقیقی برای حذف فلوگراف بیان نکرده است اما مدیر روابط عمومی این شرکت بیان کرده است، ما زمانی را برای حذف آن در نظر نگرفته‌ایم، اگرچه در سال گذشته میلادی (۲۰۱۷) در فروم‌های مختلف کاربران جهان زبان گرافی سیماتیک را جایگزین خوبی برای فلوگراف دانسته‌اند.



یک عکس از کدهای برنامه نویسی در زبان لوا را مشاهده می کنید، با توجه به فیدبک های کاربران از سراسر جهان در آینده این زبان برنامه نویسی در کرای انجین حذف خواهد شد، اگرچه کرایتک در این رابطه سکوت کرده است و کرایتک عملاً بررسی مجدد زبان لوا در کرای انجین را در نقشه راه ^۸ منتشر کرده است.

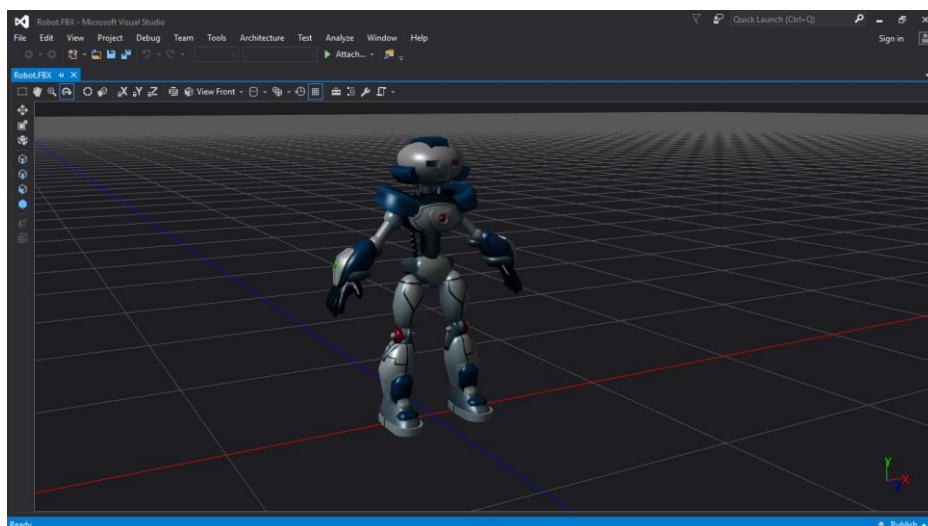
کدهای C++ : اگر قبلاً با زبان C++ کار کرده باشید می دانید که این زبان کاملاً میانی بوده و سرعت نسبتاً خوبی در اجرای کدها را دارد و خواندن کدها توسط برنامه نویس امکان پذیر است، کرای انجین بالای ۸۵ درصد با زبان C++ نوشته شده است و باقی مانده کرای انجین با زبان های پایتون، کیوتی، اسمبلی و ... است، کدهای کرای انجین به صورت داخلی با لیست پیوندی C++ عجین شده است و ساختارهای جدیدی با توسعه زبان کیوتی (QT) با زبان C++ باعث می شود شما به ساخت پلاگین های مختلف در سطح سندباکس ^۹ بپردازید.

فیزیک : مجموعه ای غنی و قدرتمند از مباحث فیزیک کلاسیک را شامل میشود، مانند شبیه سازی انفجار، اضافه کردن نیرو، گرانش، جاذبه و حجم عظیمی از مباحث را می توان به صورت عملی و بصری را امتحان کرد و از آن لذت ببرید، مجموعه ابزارهای فیزیک در کرای انجین در حال توسعه است و نقشه راه برای آن طرح ریزی شده است، این ابزارها تنها مربوط به فیزیک نیست و طیف وسیعی از ابزارها و ویژگی ها را دربر می گیرد.

ریاضیات : مجموعه ای فوق العاده و جذاب از مباحث ریاضیات را می توان دید مانند بردارها، آرایه ها، ماتریکس ها، معادلات خطی و جبری و نسبت های مثلثاتی و حجم عظیمی از مباحث را می توان به صورت عملی و بصری را امتحان کرد و از آن استفاده کنید، به جرات می توان گفت که استقلال مباحث ریاضی و فیزیک در کرای انجین منحصر به فرد است.

مدل های سه بعدی : کرای انجین در فضایی ۶ بعدی است و بازی هایی که طراحی می کنید می توانید در ابعاد ۳ بعدی و ۲،۵ بعدی اتفاق بیفتند، مجموعه ای از ابعاد طول، عرض، ارتفاع، زمان، طول رابط کاربری ، عرض رابط کاربری ۶ بعد کرای انجین را تشکیل می دهند، در این میان آنچه که در یک بازی یا در زمان طراحی مراحل ^{۱۰} می بینید مانند صندلی، میز، دشمن، در، درخت و غیره از مجموعه ابعاد طول، عرض و ارتفاع تشکیل شده است و متریکال یا متریکال ها بر اساس شیدر و تکسچر روی آن نمونه یا مدل سه بعدی نگاشت می شود، مدل های سه بعدی از مجموعه ای از مثلث ها و چندضلعی ها تشکیل شده است و بر اساس توان و مهارت مدل ساز سه بعدی که مدل را پدید می آورد، یک مدل مانند یک درخت یا یک انسان از چند صد مثلث تا چند ده هزار مثلث می تواند تشکیل شده باشد، حتی یک کره سه بعدی نیز از چند ده مثلث و یا چند ضلعی تشکیل

شده است و واضح است که هر چه تعداد مثلث‌ها یا چند ضلعی‌ها کمتر باشد، کیفیت مدل پایین آمده و همزمان پردازش مدل سه بعدی توسط CPU سبک‌تر و راحت‌تر است و برای افزایش کیفیت مدل‌ها با تکنیک‌های ایجاد تکسچرهای با کیفیت بالا و شیدرهای پیشرفته کیفیت مدل‌های سه بعدی را افزایش می‌دهند اما در کنار آن میزان استفاده از RAM و Hard-Disk افزایش می‌یابد.



یک مدل سه بعدی از یک ربات با تسکچر و متریال مربوطه که در Visual Studio ۲۰۱۵ لود شده است

اسپرایت (Sprite): به یک تصویر یا مجموعه‌ای از تصاویر که باعث پدیدار شدن یک موضوع یا یک انیمیشن گفته می‌شود، اسپرایت‌ها اگرچه مشخصاً در بازی‌های دو بعدی استفاده می‌شوند اما در بازی‌های سه بعدی نیز کاربرد دارد، مثلاً در کوهستان‌های پوشیده از جنگل، درختان دور دست با لود شدن اسپرایت نه مدل‌های سه بعدی باعث افزایش سرعت بازی می‌شود و به جای پردازش درختان دور دست که شامل چند هزار مثلث می‌شود، تصاویر

درختان به صورت اسپرایت لود می‌شود و بقیه مصرف پردازش CPU صرف مدل‌های سه بعدی مهم‌تر خواهد شد، مثلاً نزدیک‌ترین مدل‌هایی که به دوربین نزدیک هستند

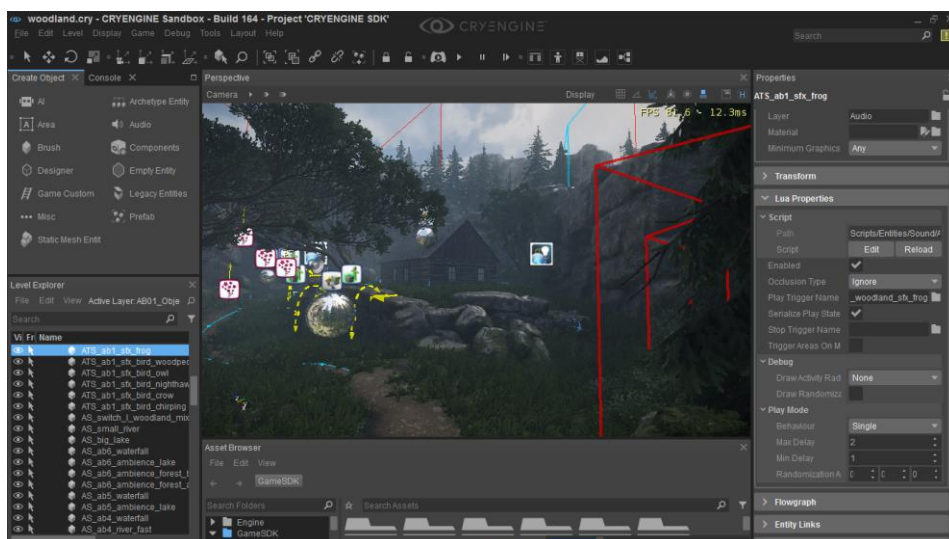
نورپردازی (Lightmapping) : بسیار جالب است بدانید که تنها موتور بازی‌سازی که نورپردازی آن ریل‌تایم و در زمان واقعی انجام می‌شود کرای انجین است، در موتورهای بازی‌سازی دیگر اگرچه این سیستم وجود دارد اما ناقص است و اثرات آن در یک فضای باز بزرگ مانند یک شهر بزرگ نتایج معکوس و حتی باعث کرش بازی یا موتور می‌شود اما کرای انجین با ارائه سیستم جدید نورپردازی علاوه بر ریل‌تایم بودن، نورحجمی را به صورت زیبا و با کیفیت عالی شبیه‌سازی می‌نماید.

دیکال (Decal): این یک اینتیتی بر اساس متریالی است که می‌تواند اثر گلوله بر روی دیوار، روغن ریخته شده بر روی آسفالت خیابان، خون‌های ریخته شده بر روی زمین، اثر ترمز اتومبیل بر روی جاده و غیره را شبیه‌سازی نماید.

سیستم آب و هوا (Weather System): تنها موتور بازی‌سازی که این سیستم را به صورت پیش‌فرض و با بالاترین دقت همراه دارد کرای انجین است، این سیستم شامل ابرهای سه بعدی بر اساس ارتفاع از زمین، بادهای مختلف، باران، برف، سیستم اثرپذیری بر روی آب با نام ripple-generator است، علاوه بر آن وجود آب سه بعدی و تاثیر از نورهای مختلف مثل نورخورشید، وجود سیستم‌های دریاچه، رودخانه و اقیانوس باعث جذابیت بیشتر این موتور شده است و در انجین‌های دیگر این امکانات وجود ندارد، نکته جالب اینجاست که سیستم طراحی جاده در کنار سیستم رودخانه

نیز وجود دارد، این سیستم‌ها برای ساخت بازی‌های ۲،۵ بعدی و صد البته سه بعدی بسیار ضروری هستند.

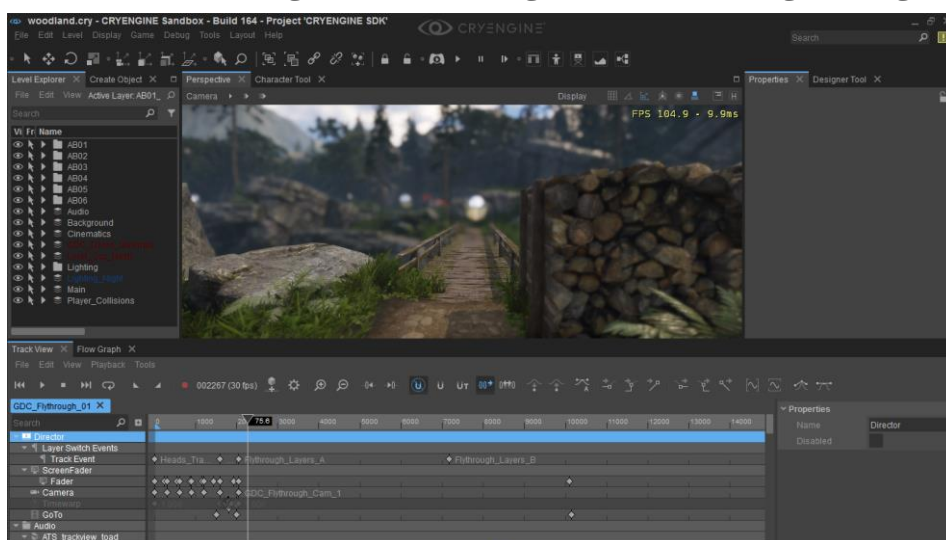
سیستم شب و روز (Time of Day): این سیستم شبیه‌سازی کاملاً واقعی از چرخش شب و روز بر اساس طولانی کردن شب یا طولانی کردن روز بر اساس میزان پارامترها را شامل می‌شود، این سیستم به نحوه نمایش یا عدم نمایش مه در آسمان و زمین و کنترل مه در هر ساعت از شبانه روز کنترل دقیقی دارد و بررسی امکانات این سیستم در بحث این کتاب نمی‌گنجد.



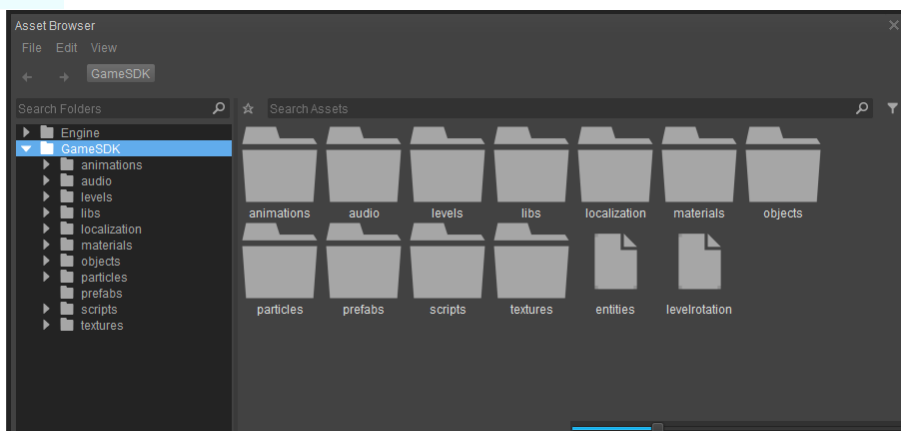
انیمیشن: به مجموعه‌ای از قاب‌ها یا تصاویر پیوسته بر اساس تغییر حالت رفتار یا تغییر حالت شی نمایش داده می‌شود مانند باز شدن و بسته شدن در اتاق، دویدن قهرمان داستان، پرتاب و حرکت موشک به طرف دشمن برای پیاده‌سازی انیمیشن‌ها بر اساس موضوع یا شی هدف ۳ راه حل مختلف وجود دارد:

۱- استفاده از نرم افزارهای مدل‌سازی سه بعدی مانند مکس، مایا، بلندر، سینمادی برای قهرمان داستان

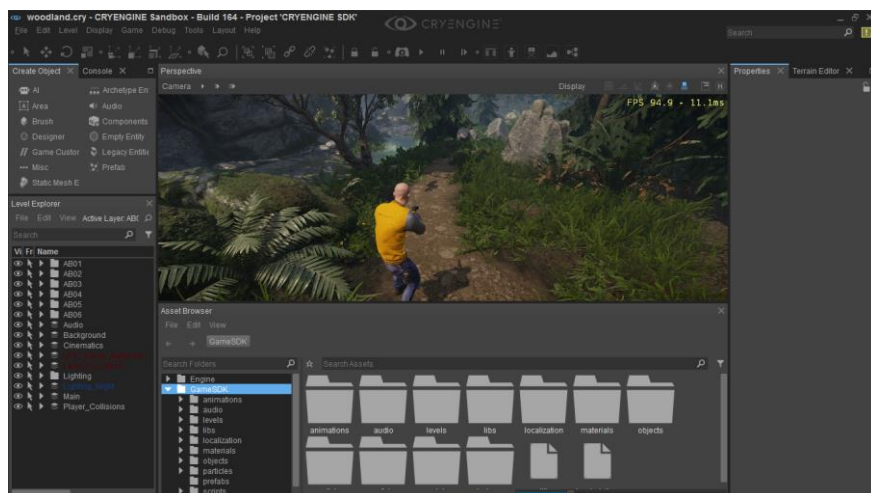
- ۲- استفاده از ابزار سیماتیک کرای انجین برای باز و بسته شدن در اتاق
- ۳- استفاده از معادلات فیزیک و ریاضی در کدهای C++ برای پرتاب و حرکت موشک به طرف دشمن
- کات سین (Cut-scene): در بازی‌ها مشاهده کرده‌اید که انیمیشنی در حال اجراست که پلیس یا قهرمان داستان کنترلی بر روی آن ندارد و این انیمیشن در حال نمایش یک موضوع و یا برای راهنمایی کردن قهرمان داستان در داخل مرحله است، مثلاً نمایش آغاز بازی که دزدان دریایی به کشتی حمله می‌کنند و کاپیتان کشتی درخواست کمک می‌کند



Asset (منابع): به مجموعه‌ای از فایل‌های تصویری مانند تکسچرها و اسپرایت‌ها، انیمیشن‌ها، موسیقی‌ها، اصوات و مکالمه‌ها، مدل‌های سه بعدی، متریال‌ها و شیدرها و غیره گفته می‌شود. Asset‌ها در سطح پروژه شامل فایل‌های فیزیکی با پسوندهای مختلف هستند و اینتیتی‌ها و پریفب‌ها در زمان طراحی مرحله منطق بازی را پیاده می‌کنند و منشاء آنها از Asset‌ها است.



پلیر (Player): به قهرمان داستان گفته می‌شود که نقش اصلی را در بازی ایفا می‌کند، این قهرمان باید مراحل بازی با چالش‌های جدید را با موفقیت بگذراند.



کرای-کش (CryCash): سیستمی جدید برای پرداخت درون برنامه‌ای بازی‌ها در کرای انجین است که شما می‌توانید از این سیستم استفاده کنید، مثلاً ارتقاء اسلحه، خرید مهمات، دریافت سلامتی پلیر

برای درک مطالب این فصل می‌توانید به آموزش‌های ویدئویی بیشتر در اکانت یوتیوب من یا آپارات من و در صورت اینکه به اینترنت دسترسی ندارید به حداقل آموزش‌های من بروی DVD در ضمیمه این کتاب مراجعه بفرمایید

فصل دوم

نصب CryEngine و پروژه GameSDK

۱- به فروشگاه اینترنتی شرکت کرایتک گفته می شود که شامل منبع عظیمی از پروژه های مختلف است که می توانید بخشی از این پروژه ها را به صورت رایگان در اختیار کاربران و یا بخش دیگری از محصولات تان را با پرداخت هزینه توسط کاربران از طریق ویزاکارت-مسترکارت-پیپال و غیره به فروش رسانید، شما نیز می توانید منابع ارزشمندتان را به صورت رایگان یا پولی در این فروشگاه منتشر کنید، طبق قوانین کپی رایت خرید و فروش محصولات انجام می شود، این فروشگاه در آدرس زیر وجود دارد :

قبل از اینکه شروع به نصب لانچر نمایید لازم است بدانید که مشخصات سیستم شما چیست و آیا سیستم شما قدرت آن را دارد که کرای انجین را اجرا کند یا نه؟ حداقل سیستم شما باید دارای RAM با حافظه ۴ گیگابایت و CPU دو هسته ای با سرعت ۲ گیگاهرتز باشد و سیستم عامل ۶۴ بیتی (ویندوز ۷ - ویندوز ۸ یا ویندوز ۱۰) و کارت گرافیک نیز حداقل باید ۱ گیگابایت حافظه داشته باشد، اگر سیستم شما قدرتمندتر است، شما می توانید بهتر و سریع تر بازی تان را بسازید، سیستم من دارای حافظه ۱۶ گیگابایت با CPU هفت هسته ای ۴ گیگاهرتز و حافظه هارد دیسک ۱ ترابایت و کارت گرافیکی ۲ گیگابایت حافظه است

مشخصات کامپیوتری که برای نصب و راه اندازی کرای انجین (حداقل منابع سخت افزاری)

OS: Windows Vista SP۱, Windows ۷, ۸,۱, ۱۰ (۶۴-bit only)

Processor: Intel Dual-Core ۲GHz or AMD Dual-Core ۲GHz

Memory: ۴ GB RAM

Graphics: NVIDIA GeForce ۴۰۰ series or AMD Radeon HD ۶۰۰۰ series

DirectX: Version ۱۱

Hard Drive: ۸ GB available space

Sound Card: DirectX Compatible Sound Card with latest drivers

مشخصات کامپیوتری که برای نصب و راه اندازی کرای انجین (توصیه شده برای داشتن منابع سخت افزاری)

OS: Windows ۷, ۸,۱, ۱۰ (۶۴-bit)

Processor: Intel Quad-Core (i۵ ۲۳۰۰) or AMD Octo-Core (FX ۸۱۵۰)

Memory: ۸ GB RAM

Graphics: NVIDIA GeForce ۶۶۰Ti or higher, AMD Radeon HD ۷۹۵۰ or higher

DirectX: Version ۱۱

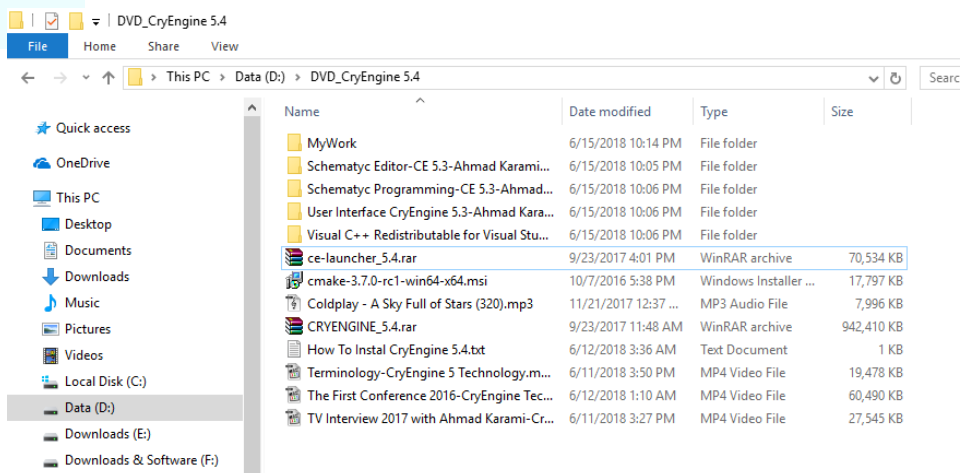
Hard Drive: ۸ GB available space

Sound Card: DirectX Compatible Sound Card with latest drivers

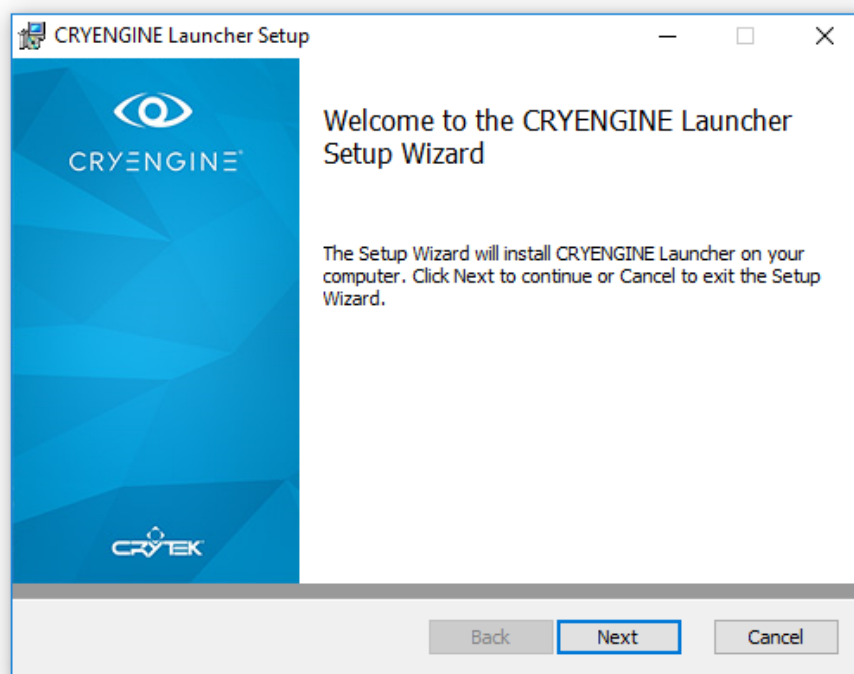
هرگز فراموش نکنید که برنامه نویسی در کرای انجین با زبان C++ در محیط ویندوز ۶۴ بیتی و سخت افزار ۶۴ بیتی امکان پذیر است و برای محیط ویندوز ۳۲ بیتی و سخت افزار ۳۲ بیتی امکان پذیر نیست اما اجرای بازی ها در محیط ویندوز ۳۲ بیتی و سخت افزار ۳۲ بیتی کاملاً امکان پذیر است، پس اگر ویندوز شما ۶۴ بیتی نیست و سخت افزار شما ۶۴ بیتی نیست، نمی توانید از کرای انجین استفاده کنید

بیايد وارد مرحله نخست از نصب کرای انجین شويم :
ابتدا لانچر کرای انجین ۵,۴ را نصب می کنیم که مراحل آن به صورت زیر است :

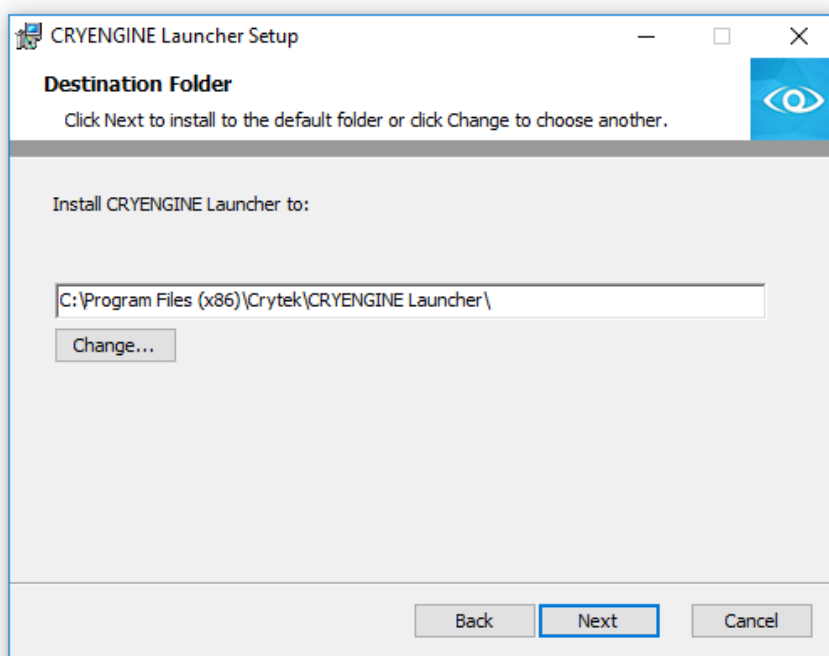
برای دسترسی به کرای انجین و پروژه های مختلف در Marketplace^۱ نیاز است که حتماً لانچر (Launcher) را نصب کنید، در این فصل نحوه نصب لانچر را به صورت گام به گام و البته تصویری توضیح داده ام و در آخر به شرح و راه اندازی پروژه نمونه و عالی GameSDK می پردازم.
برای دسترسی به لانچر بر روی DVD در ضمیمه این کتاب مراجعه کنید و فایل لانچر را از حالت فشرده خارج کنید (رمز اکستراکت کردن فایل فشرده مشخص است)، من برای اکستراکت کردن لانچر و کرای انجین از نرم افزار WinRAR استفاده می کنم و شما نیز از یکی از نرم افزارها برای اکستراکت کردن لانچر و کرای انجین استفاده کنید و نهایتاً تا به فایل اجرایی لانچر دسترسی داشته باشید و سپس بر روی فایل لانچر دو بار کلیک کنید



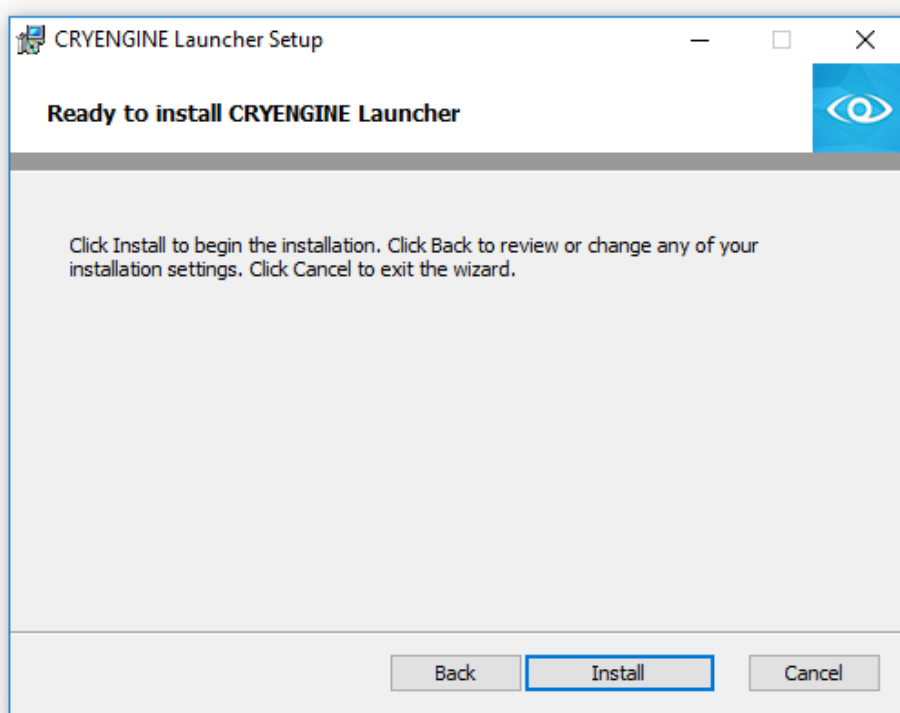
حالا پنجره ای به شکل زیر ظاهر می شود و از شما درخواست می کند که برای ادامه نصب روی دکمه Next کلیک کنید یا برروی دکمه Cancel کلیک کنید و انصراف از نصب را به ویندوز اعلان کنید.



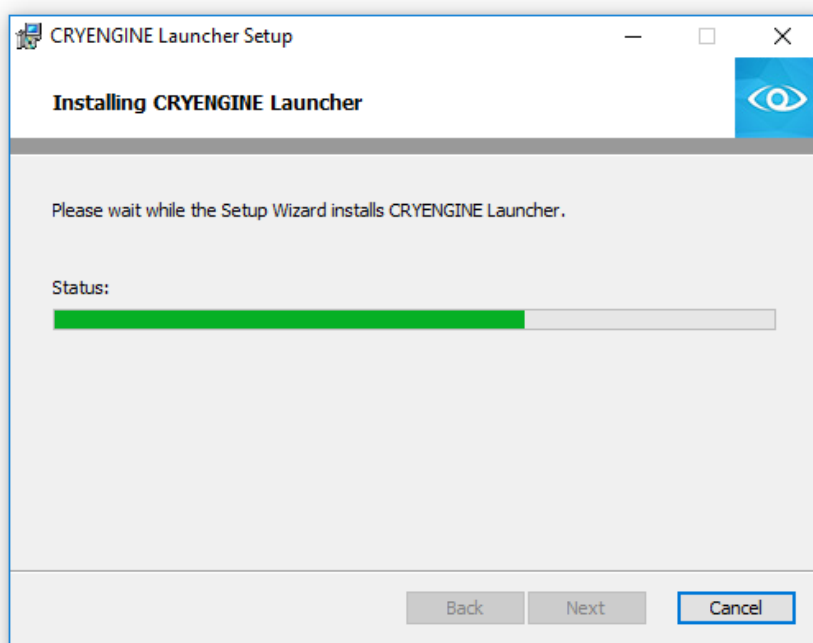
با کلیک برروی دکمه **Next**، مرحله بعدی از نصب لانچر کلید می خورد و برای شما مسیر پیش فرضی را پیشنهاد می کند، همانطور که از روی شکل مشخص است مسیر برروی درایو C است و لانچر با نوع ساختار اجرایی x87 یعنی همان معادل ۳۲ بیتی (سابق) نصب می شود.



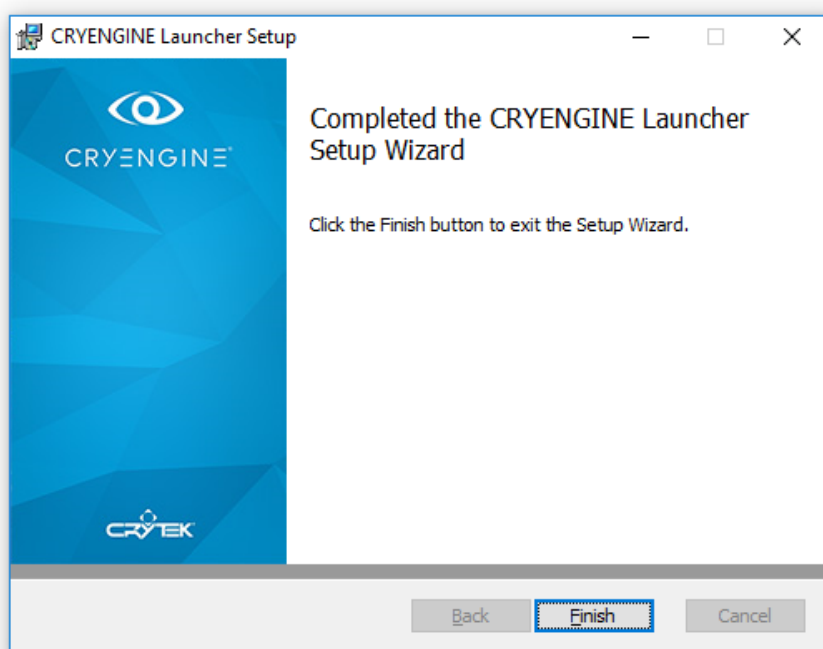
حالا برروی دکمه **Next** کلیک کنید تا وارد مرحله دیگری از نصب شویم یا با کلیک برروی دکمه **Back** به یک مرحله عقب برگردید یا با کلیک برروی دکمه **Cancel** انصراف از عمل نصب لانچر را اعلان کنید، بسیار خب، پنجره بعدی منتظر فرمان شماست و در این پنجره شما می توانید برروی دکمه **Install** کلیک کنید



با کلیک بر روی دکمه Install، عملیات اکستراکت و کپی برداری از فایل های لانچر از روی DVD بر روی مسیر مشخص شده بر روی Hard-disk آغاز می شود.



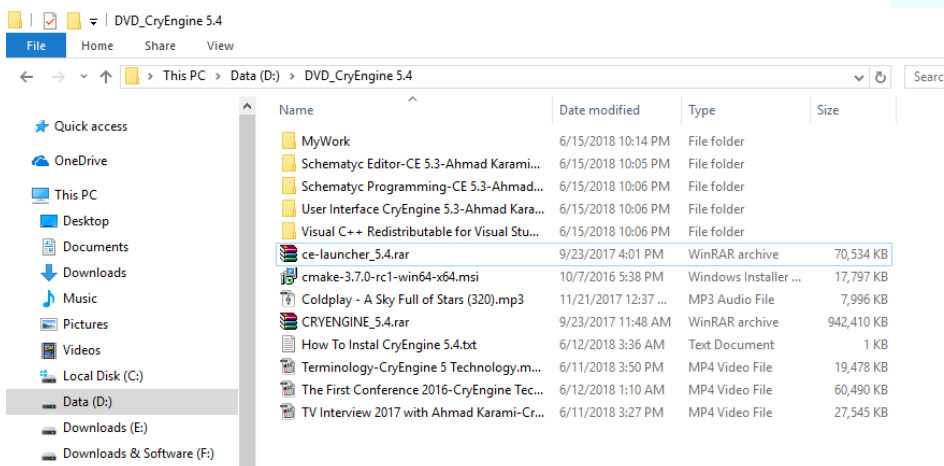
در این میان و فاصله از نصب شما می‌تونید برروی دکمه Cancel کلیک کنید و با این کار عمل نصب لانچر متوقف می‌شود، اگر منتظر بمانید، پنجره بعدی ظاهر می‌شود



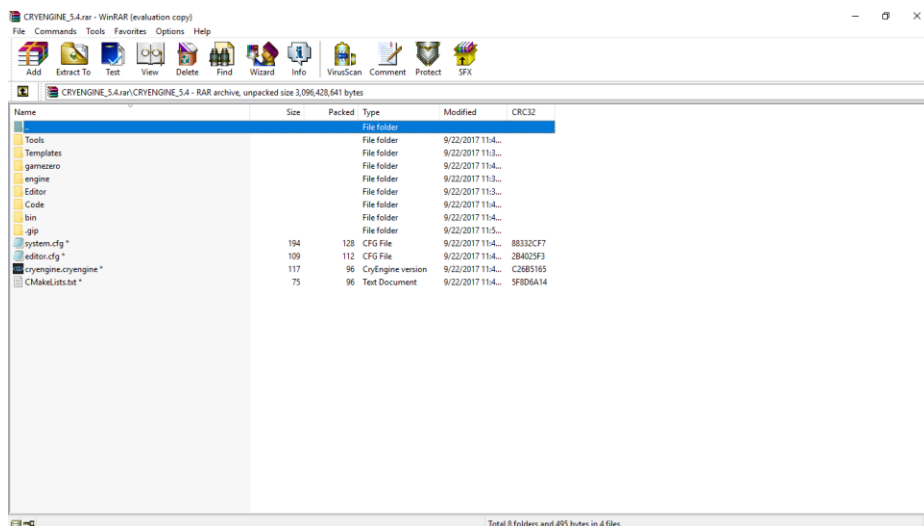
حالا بر روی دکمه Finish کلیک کنید.

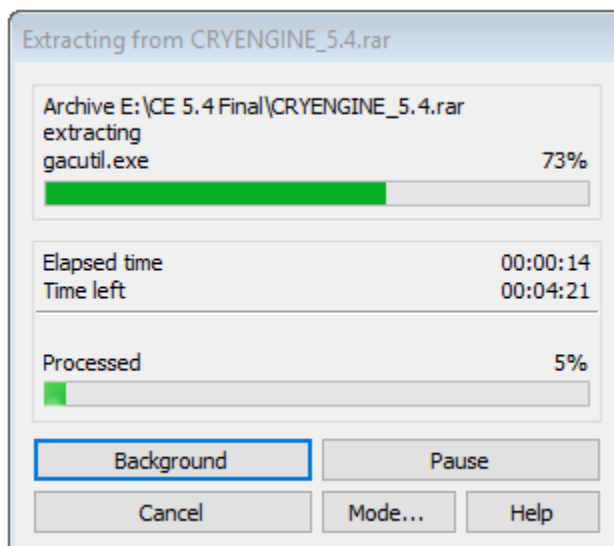
به شما تبریک میگم، شما گام نخست برای دسترسی به کرای انجین را با موفقیت پشت سر گذاشتید.

در این مرحله نوبت آن است که کرای انجین ۵,۴ را نیز نصب کنید، در داخل DVD در ضمیمه این کتاب فایل فشرده کرای انجین وجود دارد و آن را از حالت اکستراکت خارج کنید

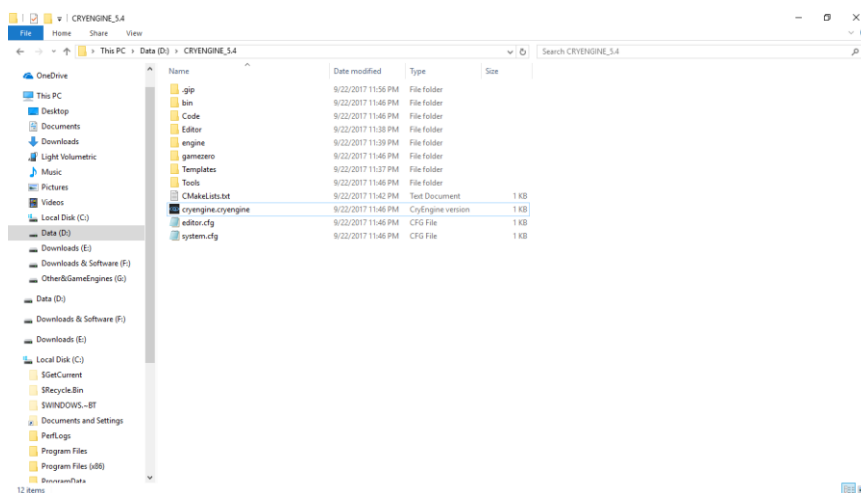


این فایل در مسیر دلخواه می تواند از حالت فشرده خارج شود (رمز اکستراکت کردن فایل فشرده مشخص است) اما برای راه اندازی کرای انجین لازم است که فایل کرای انجین ۵,۴ که از اکستراکت یا فشرده شده خارج شده است را در مسیر مشخص به شرح زیر Copy/Paste کنید.

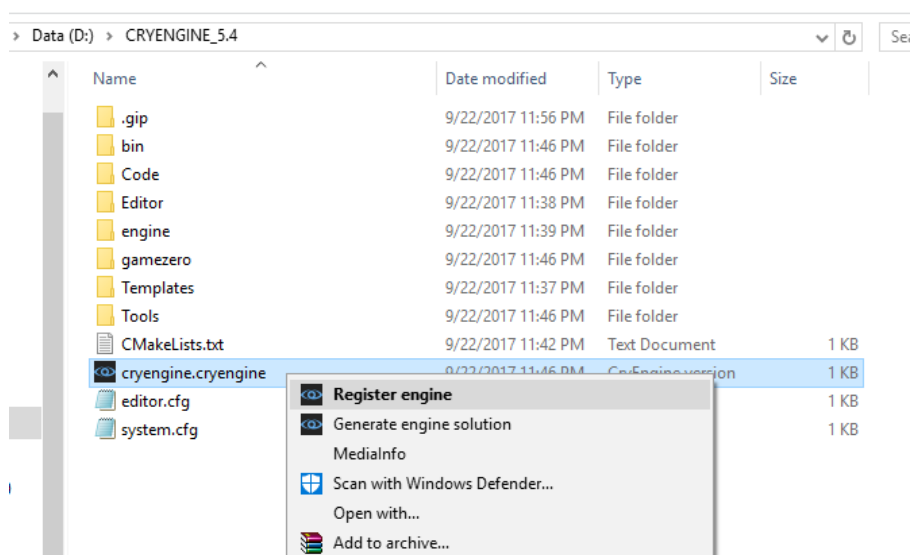




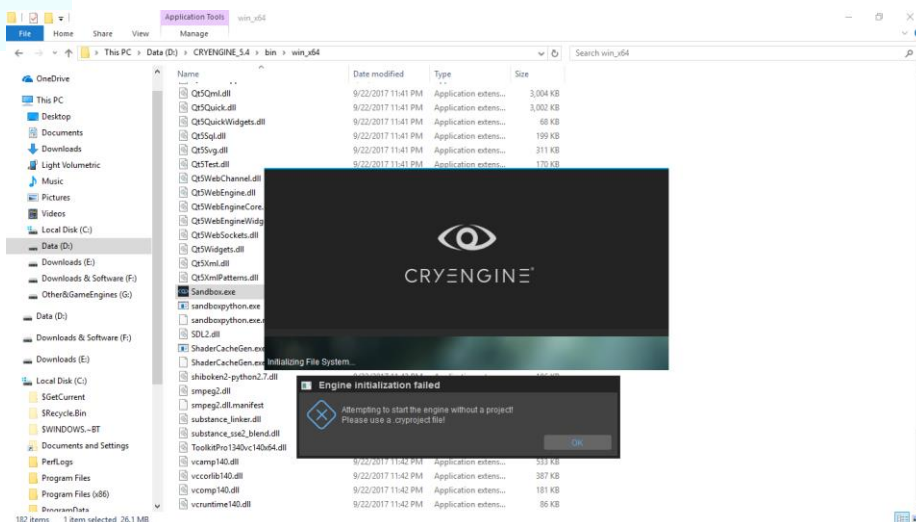
عملیات اکسترکت یا از فشرده خارج کردن فایل کرای انجین می تواند کمی زمانبر باشد پس شکبیا باشید و منتظر بمانید، پس از اتمام عملیات، تصویر آنچه که در زیر می بیند را برروی کامپیوترتان خواهید یافت، این پوشه با کلیه فایل ها و زیر پوشه ها در مسیری دلخواه می تواند قرار گیرد



فراموش نکنید که حتما نیاز است که عملیات رجیستر کردن و ثبت کرای انجین در ویندوز لازم است و مثل آنچه که در تصویر زیر می بینید عمل نمایید.

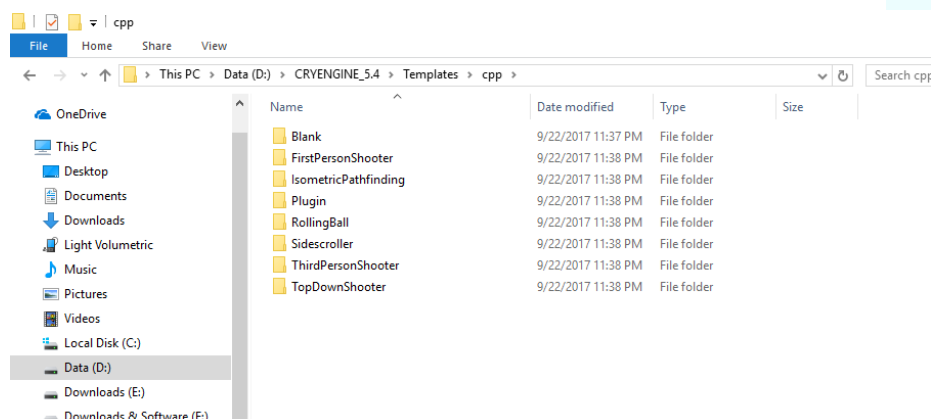


برروی فایل cryengine.cryengine راست کلیک کنید و سپس گزینه Register engine را انتخاب کنید، با انتخاب گزینه Register engine صفحه سیاهی ظاهر می شود و سپس با اتمام عملیات رجیستر شدن موتور کرای انجین، صفحه سیاه رنگ محو می شود، فراموش نکنید برای دسترسی به سندباکس کرای انجین با نام Sandbox.exe با پسوند اجرایی باید به مسیری شبیه مسیر زیر بروید

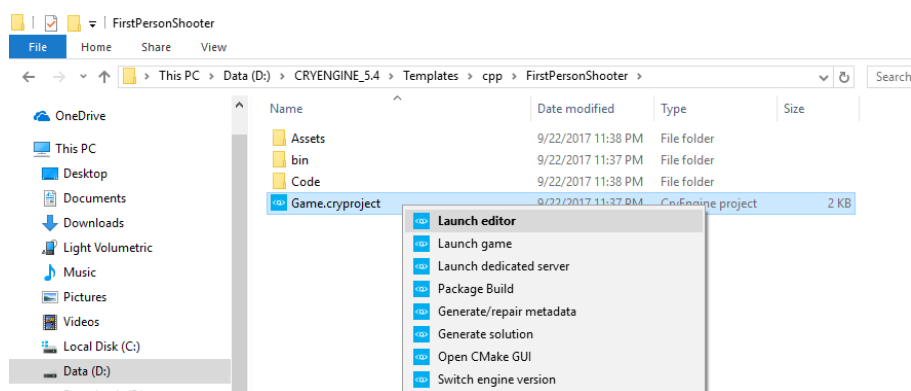


وقتی بر روی فایل اجرایی **Sandbox.exe** دوبار کلیک کنید، پیغام هشدار صادر می شود که شما برای راه اندازی انجین حتما باید پروژه ای داشته باشید تا انجین با ادیتور سندباکس برای شما نمایش داده شود، اگر مستقیما فایل اجرایی سندباکس را اجرا کنید، این پیغام هشدار مثل آنچه که در تصویر بالا می بینید بر روی کامپیوترتان را خواهید دید.

آنچه که در شکل زیر می بیند مجموعه ای از پروژه های آماده در کرای انجین است که با زبان **C++** تهیه شده است و به شما کمک می کند تا به صورت سریع پروژه های تان را با کرای انجین شروع و بازی های شگفت انگیزی را با این **Template** ها نمایش دهید، اگرچه **Template** هایی نیز به زبان سیماتیک و سی شارپ وجود دارد اما تمرکز اصلی من بر روی **C++** خواهد بود و زبان سیماتیک را به همراه بخشی از فلوگراف پوشش خواهم داد.



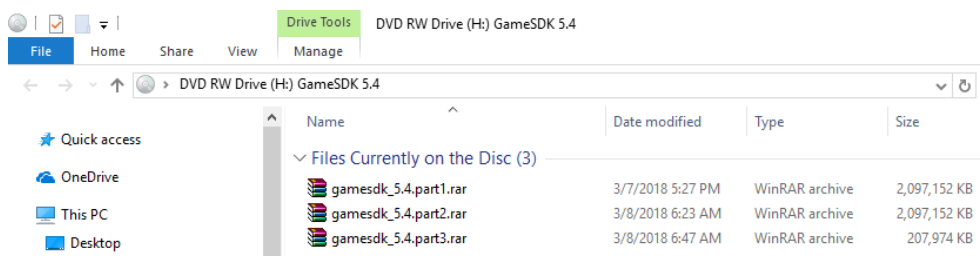
در ادامه این کتاب در فصل جداگانه به بررسی یکی از Template ها می پردازم و راه را برای ادامه کار با Template های C++ را هموار خواهم کرد.

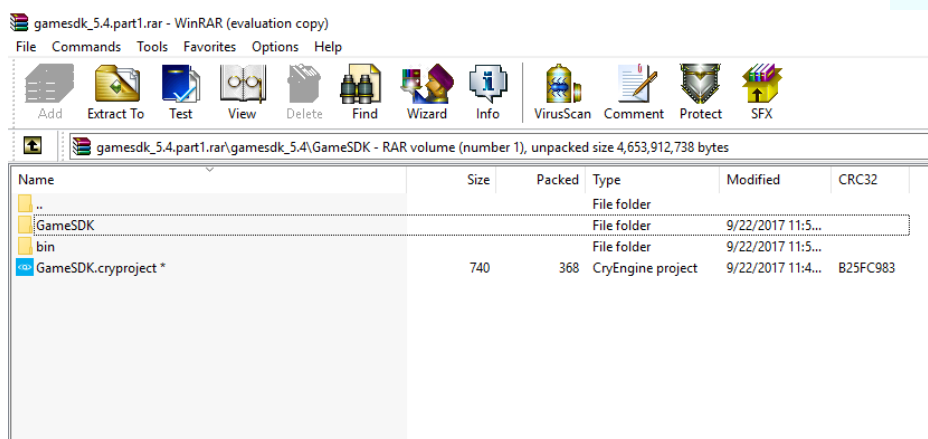


بهتر است که یک کپی پشتیبان از تمپلت ها ایجاد کنید و سپس بر روی تمپلت ها کار کنید، در غیر اینصورت باید دوباره فایل کرای انجین را از فشردگی خارج کنید تا به Template های بدون دستکاری شده دسترسی داشته باشید، فراموش نکنید که همیشه به عنوان یک برنامه نویس آخرین تغییرات اعمال شده بر روی پروژه تان را با یک یا چند کپی پشتیبان بر روی حافظه یا حافظه های DVD، Flash خارج از

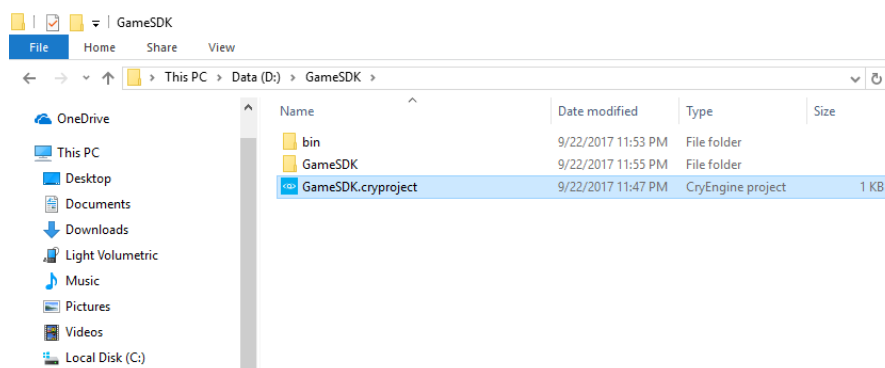
کامپیوتر داشته باشید تا در صورت مشکل و تغییرات ناخواسته و تست کدهای جدید در پروژه تان، با داشتن کپی پشتیبان از پروژه تان، زحمات تان از بین نرود، این مهم ترین اصلی است که یک برنامه نویس باید آن را رعایت کند، تجربه نشان داده است که اتفاقات ناگوار در هر پروژه می تواند رخ دهد.

جدی ترین و مهمترین پروژه از آغاز کرای انجین ۵ برای کسانی که در حال یادگیری کرای انجین هستند، همین پروژه است، پروژه GameSDK که می توانید با مراجعه به DVD در ضمیمه این کتاب، از منابع غنی و بسیار خوب این پروژه بهره ببرید، این پروژه بخش بسیار کوچکی از بازی کرایسیس ۳ در کرای انجین است، این پروژه به صورت سه part بسیار فشرده جدا است و کافیست بر روی part ۱ راست کلیک کنید و عمل اکسترکت یا خارج کردن از فشرده را در مسیر دلخواه انجام دهید، بقیه part ها نیز به صورت اتوماتیک از فشرده گی خارج می شوند، رمز نیز داخل part ۱ است و با باز کردن part ۱ کاملاً مشخص است، من از نرم افزار WinRAR برای اکسترکت کردن استفاده می کنم



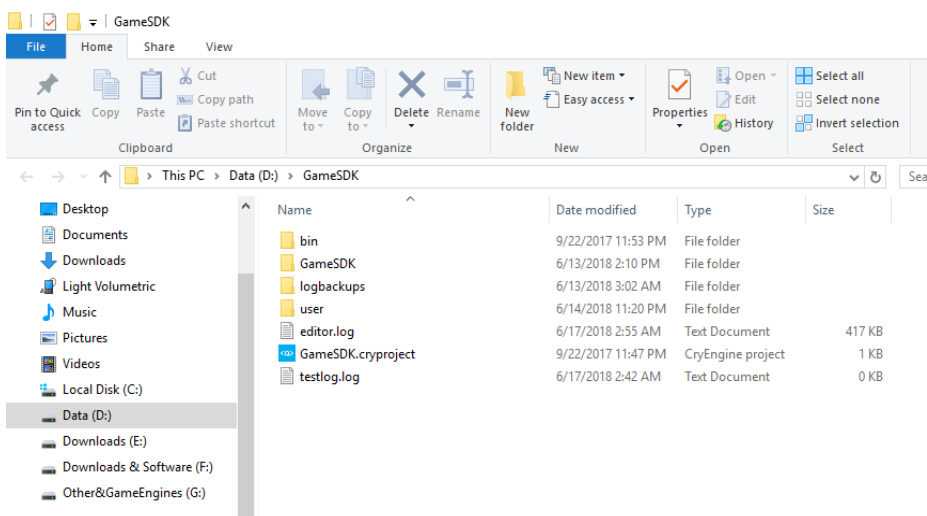


از آنجایی که حجم این پروژه سنگین است دقایقی را باید منتظر بمانید تا از فشردگی خارج شود، اگر Hard-disk شما از نوع SSD است، این زمان می تواند بسیار کمتر شود و البته سرعت اجرای بازی و طراحی بازی بسیار سریع تر انجام می شود و همانطور که از روی تصویر مشخص است، فایل پروژه با یک آیکن آبی رنگ با نام GameSDK.cryproject است

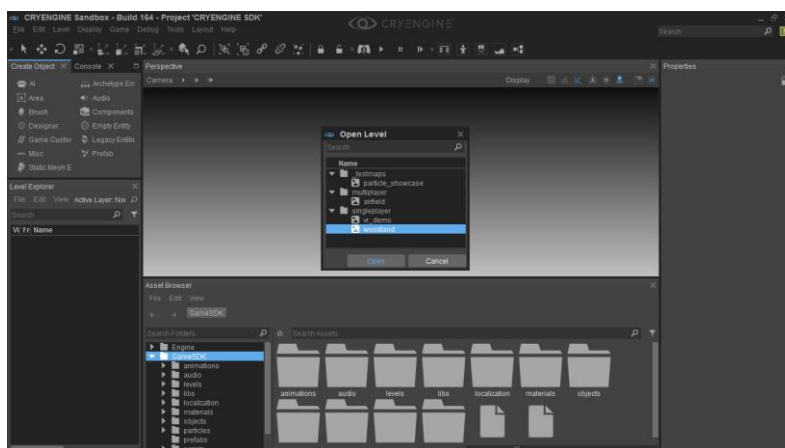


با دوبار کلیک کردن بر روی فایل پروژه ، سندباکس کرای انجین باز می شود و یا با راست کلیک کردن بر روی فایل پروژه و انتخاب گزینه " Launch editor" سندباکس کرای انجین باز می شود و سپس با مراجعه به منوی

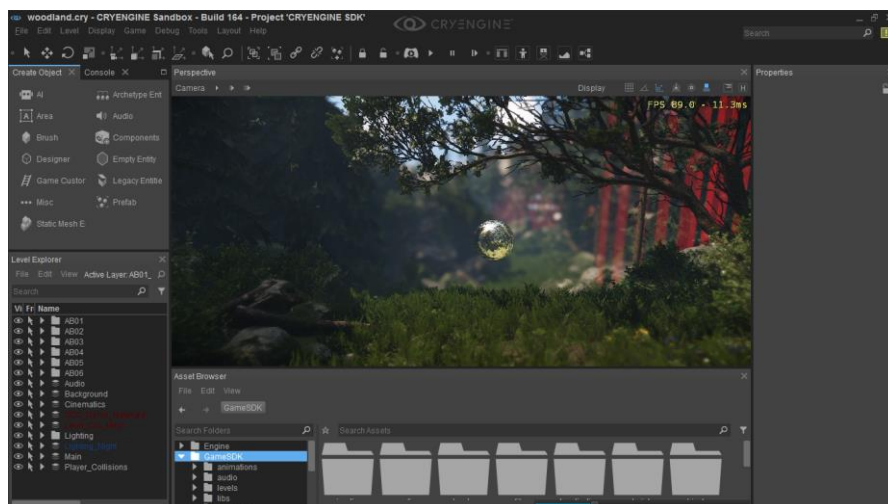
File و انتخاب گزینه Open، لیستی از مراحل بازی در این پروژه را خواهید دید.



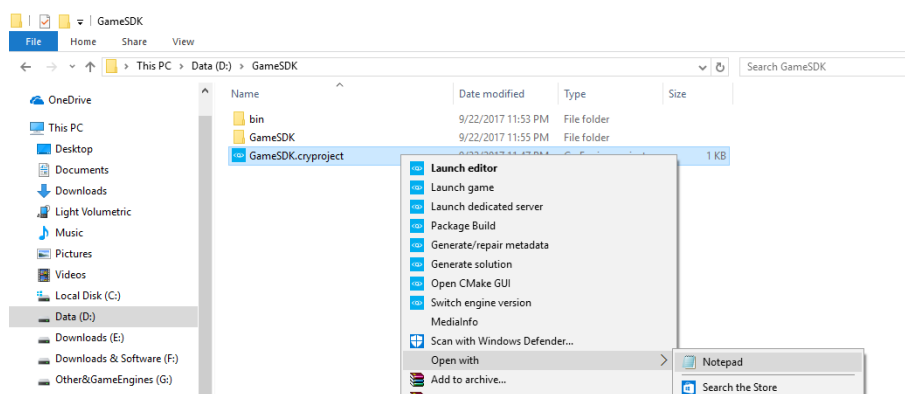
لازم به ذکر است مانند آنچه که در تصویر بالا می بیند، با باز شدن پروژه پوشه های user و logbackups و فایل های editor.log و testlog.log ایجاد می شوند که برای ثبت اتفاقات - ذخیره عملیات کاربر روی طراحی بازی کاربرد دارد.



در میان مراحل بازی گزینه woodland را انتخاب کنید و دکمه open را کلیک کنید تا مرحله woodland باز شود، این مرحله شامل جنگلی با بیشترین ابزارهای به کار رفته در کرای انجین است، این پروژه درک خوبی را از دنیای کرای انجین ۵،۴ به شما می دهد

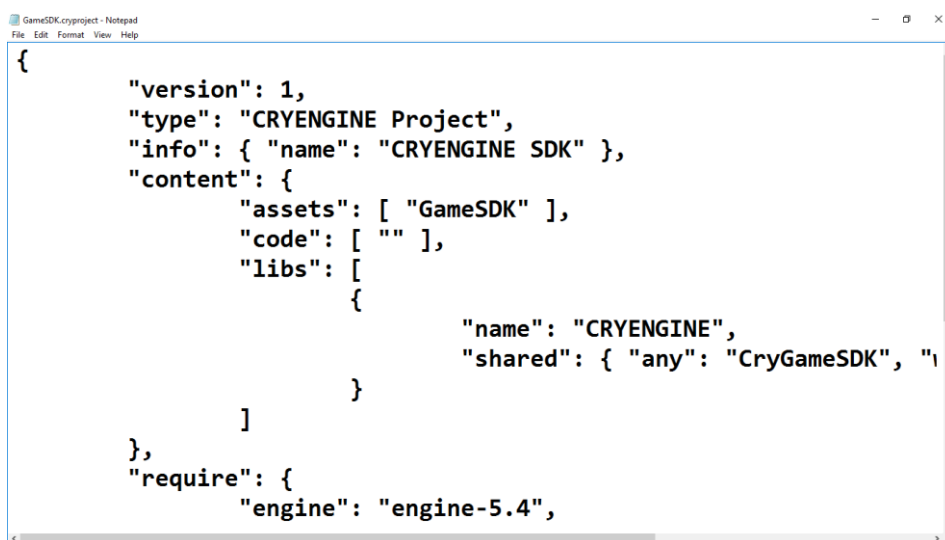


از Sandbox کرای انجین ۵،۴ خارج شوید وادیتور را ببندید و به فایل پروژه مطابق شکل زیر برگردید



به فایل پروژه برگردید و برای دسترسی به محتوای متنی فایل پروژه می توانید از نرم افزارهای ویرایشگر متنی مانند notepad استفاده کنید، بر روی

فایل پروژه راست کلیک کنید و گزینه notepad را انتخاب کنید تا شکل زیر ظاهر شود.

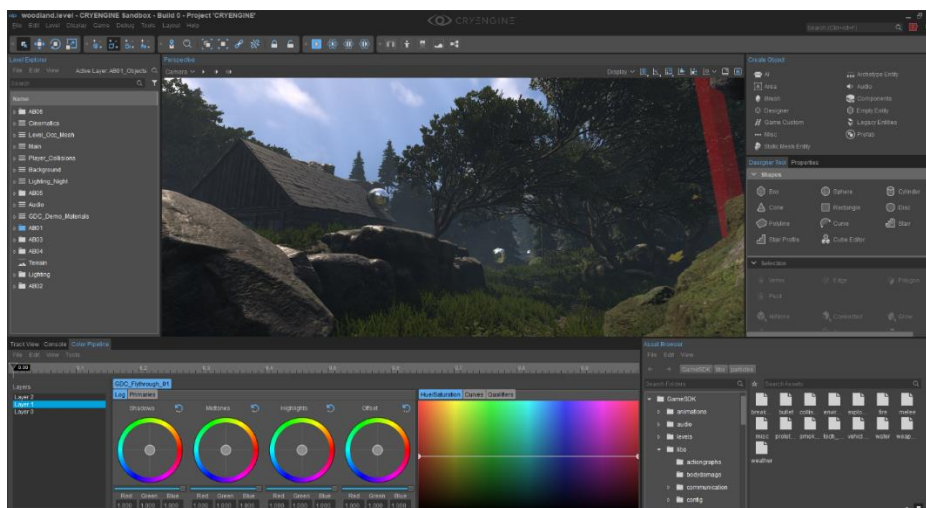


```
{
  "version": 1,
  "type": "CRYENGINE Project",
  "info": { "name": "CRYENGINE SDK" },
  "content": {
    "assets": [ "GameSDK" ],
    "code": [ "" ],
    "libs": [
      {
        "name": "CRYENGINE",
        "shared": { "any": "CryGameSDK", "1"
      }
    ]
  },
  "require": {
    "engine": "engine-5.4",
```

فراموش نکنید که همه Template ها یا پروژه ها در کرای انجین با پسوند فایل `*.cryproject` از نوع فایل متنی است و داخل آن را می توان ویرایش کرد یا با انتخاب اولین گزینه ادیتور "Launch editor" پروژه را باز کرد، فایل پروژه با توجه به تنظیمات مختلف از طریق لانچر به کرای انجین می گوید که این پروژه چگونه باز شود و کرای انجین نیز با توجه به محتوای متن فایل پروژه، پروژه را برای شما باز می کند، درواقع لانچر به معنی پرتاب کننده است و برای شما فایل پروژه و مجموعه مراحل و Asset ها را برای کرای انجین پرتاب می کند، در ادامه این کتاب با گزینه های لانچر بیشتر آشنا خواهید شد.

در اینجا تصویری از رابط کاربری کرای انجین ۵،۵ بتا با پروژه GameSDK را مشاهده می کنید، قرار است این تکنولوژی در تابستان ۲۰۱۸ منتشر شود و

هم اکنون که این کتاب در حال اتمام است، هفته های بعدی این تکنولوژی در نسخه نهایی ۵,۵ منتشر خواهد شد



برای درک مطالب این فصل می توانید به آموزش های ویدئویی بیشتر در اکانت یوتیوب من یا آپارات من مراجعه نمایید و در صورت اینکه به اینترنت دسترسی ندارید به حداقل آموزش های من به همراه دسترسی به پروژه GameSDK که شامل منابع عظیمی از Asset ها است می توانید به DVD در ضمیمه این کتاب مراجعه بفرمایید

فصل سوم

محیط نرم افزاری CryEngine و

ابزارهای مختلف آن

۱- به مجموعه کدهایی که برای نمایش دادن جلوه های ویژه در دوربین گفته می شود، مثلاً ریزش و جاری شدن قطرات آب، سیاه و سفید شدن رنگ ها، نمایش دادن محو یا تار شدن تصاویر و غیره

۲- به داستان بازی نیز گفته می شود که قهرمان داستان باید چالش ها را پست سر بگذارد، هر چه داستان بازی قوی تر باشد، میزان محبوبیت بازی بالاتر خواهد رفت، این مسئله مستقل از کدنویسی محکم با کمترین باگ ها، گرافیک خیره کننده نیز در ارتباط است، سناریو یا داستان خوب و عالی ریشه بازی است، به کسانی که سناریو بازی را می نویسند سناریونویس می گویند.

۳- Lowpoly در واقع از چندضلعی ها یا مثلث هایی تعریف شده در مدل های سه بعدی اشاره دارد، همه مدل های سه بعدی از چندصد یا چندین هزار مثلث یا چند ضلعی تشکیل شده است، به این معنا که برای ساخت هر مدل سه بعدی بر اساس مهارت تان، شما در نرم افزارهای سه بعدی مدلینگ مانند مکس، مایا، سینما ۴ دی، بلندر، هودینی انجین از مجموعه مثلث ها و چندضلعی ها استفاده می کنید، کلمه Lowpoly یعنی شما در بازی تان تا حد امکان از مدل های سه بعدی استفاده می کنید که کمترین مثلث را داشته باشند، نرم افزارهایی هستند که تعداد مثلث ها را براساس پارامترهایی کاهش می دهند و مدل های حجیم سه بعدی یا مدل های سه بعدی سنگین را با کاهش مثلث ها سبک و به Lowpoly تبدیل می کنند.

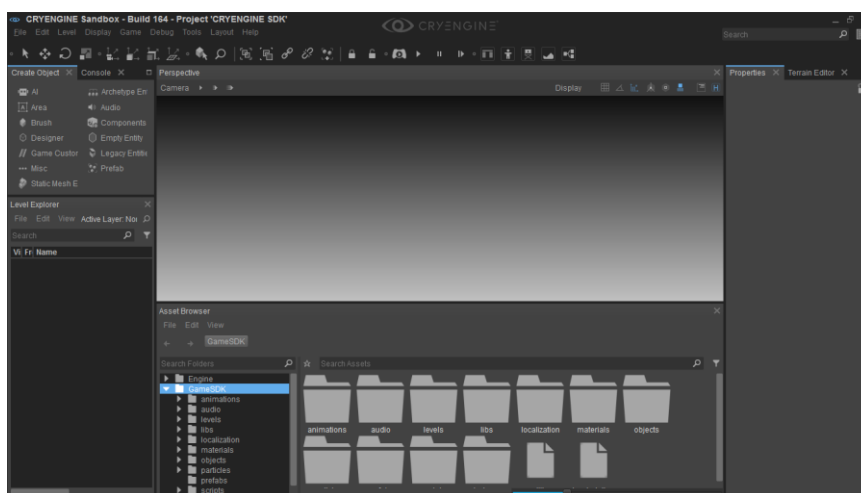
۴- واحد زمان اجرای بازی فریم ریت (Frame-rate) است، هر چه عدد فریم ریت بیشتر باشد، بازی روان تر و قابل قبول تر اجرا می شود، نرخ استاندارد واحد زمان اجرای بازی برای کامپیوتر ۶۰ فریم و برای بازی موبایل ۳۰ فریم است، به این معنا که در این محدوده از ۶۰ و ۳۰ با چند واحد افزایش و

کاهش فریم، بازی اجرا شود، فریم با قاب عکس نیز یاد می شود، یعنی برای اجرای بازی مان در استاندارد ۶۰ یا ۳۰ فریم، کامپیوتر یا موبایل مان هر ثانیه ۶۰ یا ۳۰ فریم را باید پردازش کند و چند واحد کاهش و افزایش فریم طبیعی است، اگرچه همیشه توصیه می شود تا جایی که می توانید نسبت فریم ریت و گرافیک را در سطح ۶۰ یا ۳۰ نگه دارید اما هر چه این اعداد افزایش یابد به طوری که بازی گرافیک دلچسبی داشته باشد، نشان می دهد شما مهارت بیشتری را برای ساخت بازی ها به دست آورده اید.

۵- به فرد یا افرادی که مرحله بازی را با مجموعه ای از اشیا چیدمان می کنند و این اشیا باید به گونه ای قرار بگیرد که با سناریو بازی و ادامه مرحله مطابق کامل داشته باشد، شاید این کار ساده به نظر برسد اما این کار خود نیز یک تخصص است، مثلاً شما درختانی که در جنگل های سیبری هستند را نمی توانید در کنار لب دریا با آب و هوای گرم قرار دهید، طراحان مرحله باید مطالعه خوبی در رابطه با طبیعت واقعی و دنیای انسان ها داشته باشند و مانند سناریو نویس ها باید تخیل خوبی داشته باشند.

۶- هنگامی که برنامه نویس یا طراح بازی در حال ساخت بازی و ایجاد منطق بازی با مجموعه ای از گراف ها است، این موضوع را پروگرافینگ (Prographing) گویند و هنگامی که برنامه نویس در حال ساخت بازی با یکی از زبان های برنامه نویسی است و در حال تایپ کد است، به این موضوع پروگرامینگ (Programming) می گویند.

محیط سندباکس کرای انجین ۵,۴ به شکل زیر است :



سندباکس با Layout پیش فرض از نوار منوها – نوار ابزارها و ۷ پنجره مختلف به شرح زیر تشکیل شده است :

نوار منوها که بر اساس نام هر منو و وظایف هر منو معلوم است :

File Edit Level Display Game Debug Tools Layout Help

File : ایجاد کردن، باز کردن، ذخیره کردن مراحل و خروجی گرفتن مراحل برای زبان C++ و غیره

Edit : عملیات نمونه برداری، تکثیر، حذف و جستجوی اینتیتی ها و مشخص نمودن مختصات های دکارتی مختلف و غیره

Level : ارتباط و گروه بندی اینتیتی ها، ایجاد پریفب ها، نوع انتخاب اینتیتی ها، ارتباطات بین متریال ها و اینتیتی ها و همچنین شبیه سازی و اجرای قوانین فیزیک در در هنگام طراحی مرحله

Display : کنترل و مدیریت گرافیک بازی، کنترل دوربین، جهان هندسی بازی، تعیین نقاط پرش مختلف در جهان بازی

Game : کنترل پخش صداها، محیطی، هوش مصنوعی، اجرای بازی در ادیتور

Debug : لود کردن دوباره اسکریپت های زبان لوا ، عوارض زمین ، تکسچرها و

نمایش لاگ و ثبت وقایع زمان طراحی بازی

Tools : قدرت کرای انجین در این منو است، کلیه ابزارهای مختلف و البته متفاوت در

این منو وجود دارد (فرانک ویدز کارت مثل همیشه عالیه)، این منو را در ادامه این

فصل به خوبی توضیح خواهیم داد

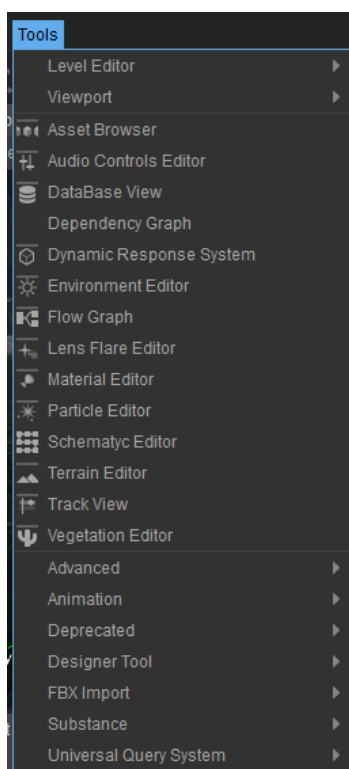
Layout : نحوه نمایش Layout های مختلف، ایجاد Layout های جدید و ذخیره

Layout های مختلف

Help : کمکیار آموزشی برخط و دسترسی به تمامی دستورات (کاماندها) و متغیرهای

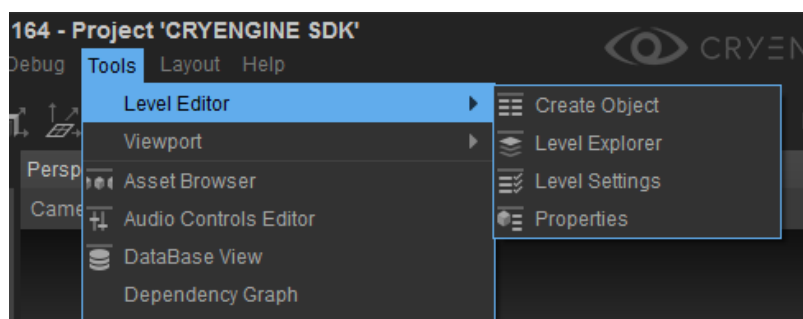
کنسولی

منوی **Tools** (قدرت کرای انجین اینجاست!)



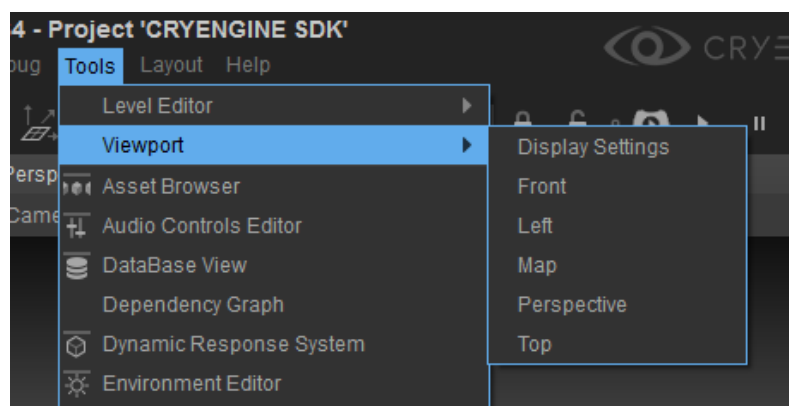
: Level Editor

برای دسترسی به پنجره های "Level Explorer"، "Create Object" Level "، "Settings"، "Properties" که پنجره Level Settings پارامترهای مختلفی برای کنترل و تغییر محیط در سطح مرحله را انجام می دهد، مثلاً افزایش و کاهش شدت باد و جهت باد و بقیه پنجره ها در ادامه توضیح دادم.



: Viewport

برای دسترسی به پنجره های "Left"، "Front"، "Display Settings"، "Top"، "Perspective"، "Map"



پنجره Display Settings : برای نحوه فیلتر و پیمایش انواع و اقسام آبجکت ها با انواع پارامترهاست

پنجره Front : آبجکت ها را در نمای جلو از مختصات جهانی در مرحله نمایش می دهد

پنجره Left : آبجکت ها را در نمای چپ از مختصات جهانی در مرحله نمایش می دهد

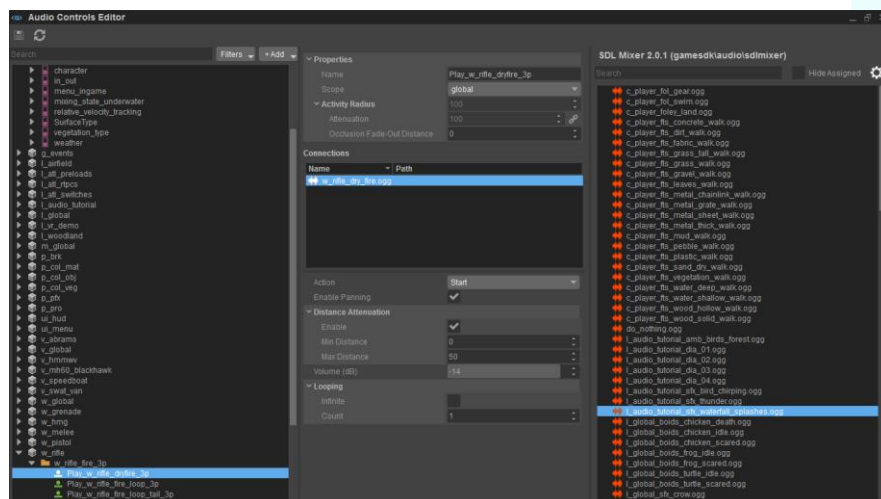
پنجره Map : آبجکت ها را در نمای مپ از مختصات جهانی در مرحله با نقشه تکسچری عوارض زمین نمایش می دهد

پنجره Perspective : آبجکت ها را در نمای پرسپکتیو از مختصات جهانی در مرحله نمایش می دهد، نمای پنجره بازی در زمان اجرا و در زمان طراحی مراحل به صورت پیش فرض در این حالت قرار گرفته است.

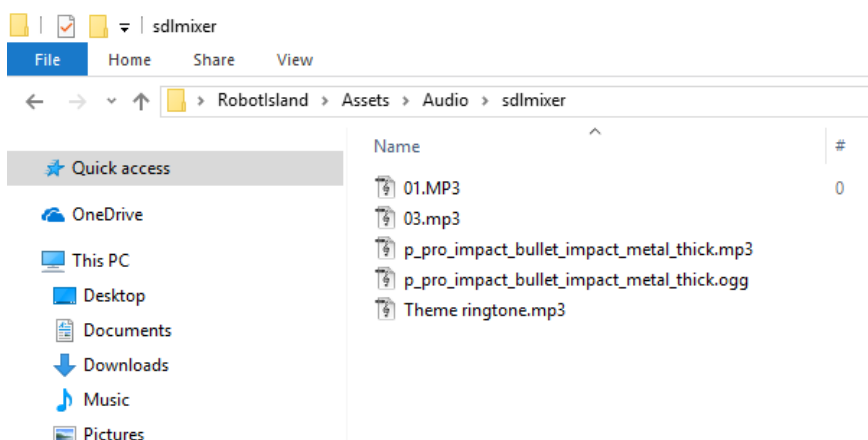
پنجره Top : آبجکت ها را در نمای بالا از مختصات جهانی در مرحله بدون نقشه تکسچری عوارض زمین نمایش می دهد

Assets Browser : این پنل را در ادامه فصل توضیح خواهیم داد.

Audio Controls Editor : برای کنترل و مدیریت کلیه اصوات، موسیقی، موزیک و دیگر صداها در بازی و در ادیتور سندباکس مورد استفاده قرار می گیرد، شما می توانید داخل ده ها یا صداها موزیک، صوت و آهنگی که به داخل پروژه بازی تان اضافه کردید، جستجو کنید و آنچه که در نظر دارید را در لیست طولانی از اصوات پیدا کنید، میزان شدت صداها را بر حسب واحد صوتی دسیبل افزایش و کاهش دهید، صداها سه بعدی (نسبت مسافت از پلیر) و دو بعدی (ثابت از کامبونت Listener-گوش کننده در دوربین پلیر) را با ابزارهای دیگر تولید کنید.

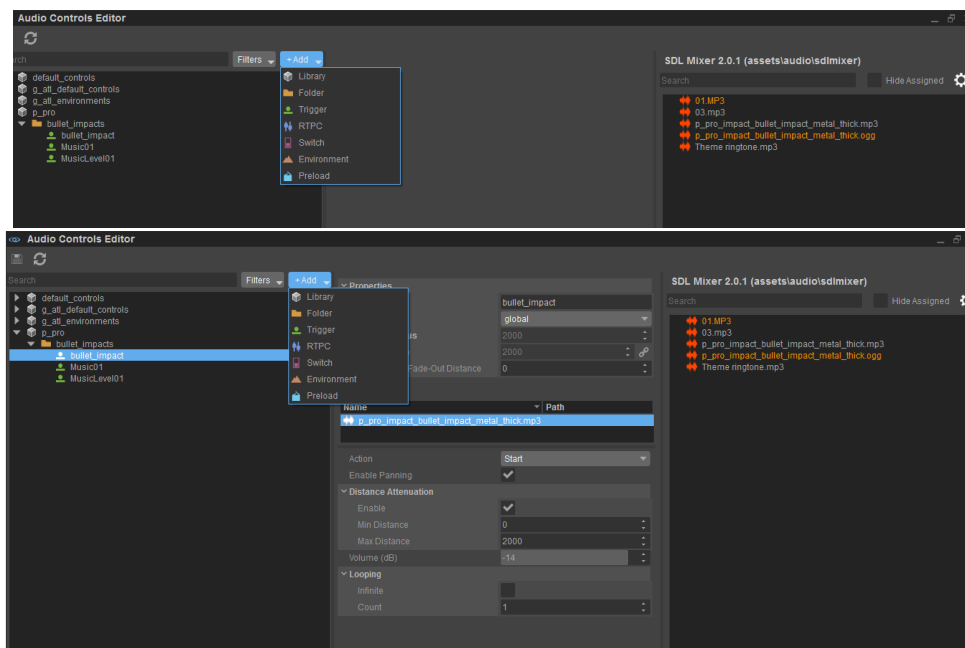


برای وارد کردن فایل های صوتی ogg , wav , mp3 و مشاهده این فایل ها در پنجره Audio Controls Editor کافیست به مسیر بروید و در صورتی که پوشه ها وجود ندارند، پوشه ها را ایجاد کنید و فایل های صوتی تان را با پسوندهای مورد اشاره در این مسیر Copy/Paste نمایید :

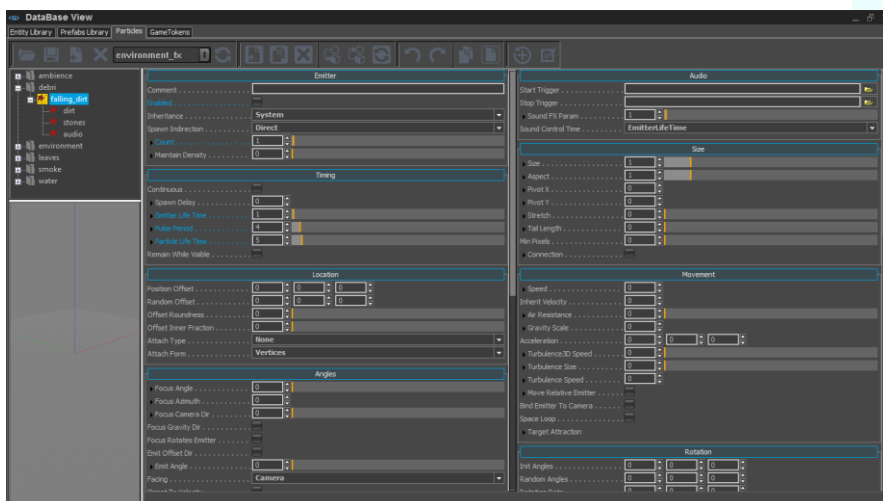


بعد کلیک برروی دکمه refresh در پنجره Audio Controls Editor، فایل های copy/paste شده با رنگ متفاوت در سمت راست پنجره نمایش داده می شود، شما باید قادر باشید که Trigger های مختلف صوتی را ایجاد کنید، کافیست برروی دکمه

Add در این پنجره کلیک کنید و اسم تریگرهای صوتی تان را وارد کنید و فایل های مختلف صوتی به تریگرها اختصاص دهید، لازم به ذکر است که در سمت راست گوشه بالا، مسیر پوشه ها و موقعیت فایل های صوتی در کنار کلمه ۲,۰,۱ SDL Mixer آمده است



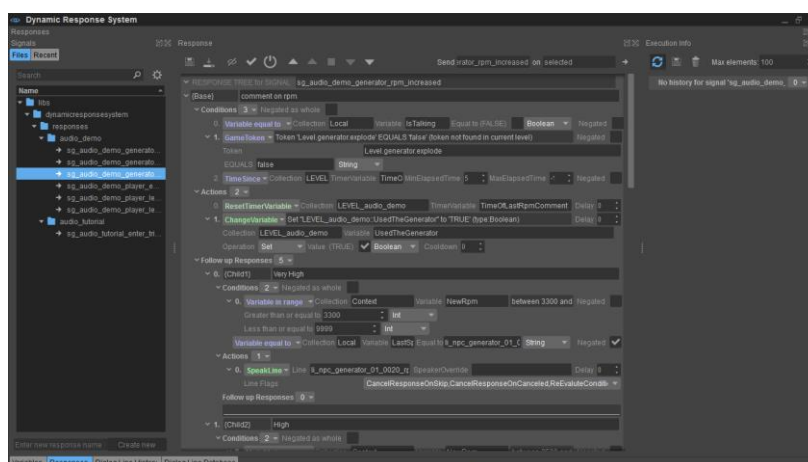
Database View : این پنجره به شما امکان دسترسی به ساخت و ویرایش و حذف اینتیتی ها، پریفب ها، پارتیکل سیستم (میراث کرای انجین ۳)، گیم توکن ها را می دهد و با ظهور کرای انجین ۵، استفاده از این پنجره کمتر شده است.



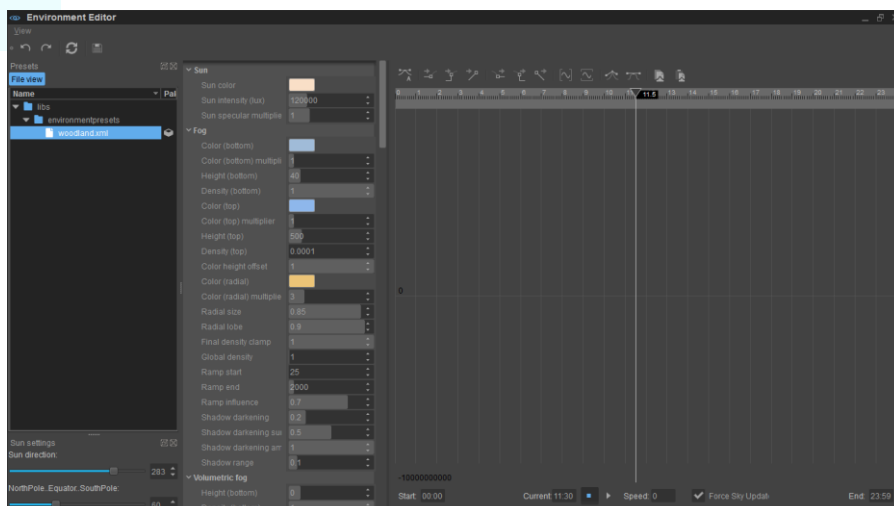
Dependency Graph : این پنجره در واقع فضای گراف و نودها را برای دیگر ابزارها آماده می کند و به صورت پیش فرض این پنجره یک فضای خالی را نشان می دهد.

Dynamic Response System :

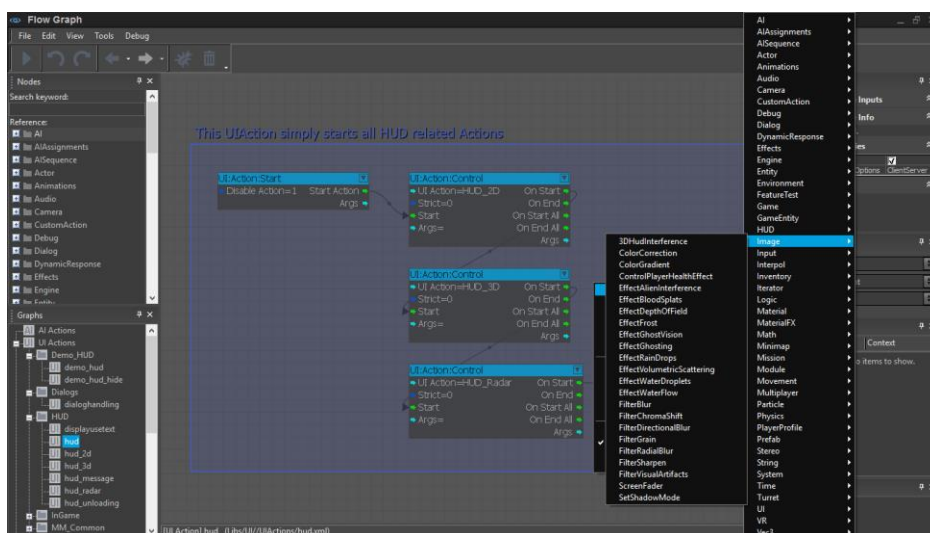
سیستم ارسال و دریافت پیام ها با استفاده از متغیرهای مختلف و قابل تعریف در این سیستم است



Environment Editor : برای تغییرات زمان شب و روز، ابرها، سایه ها، مه، میزان روشنایی نور آسمان و غیره استفاده می شود.

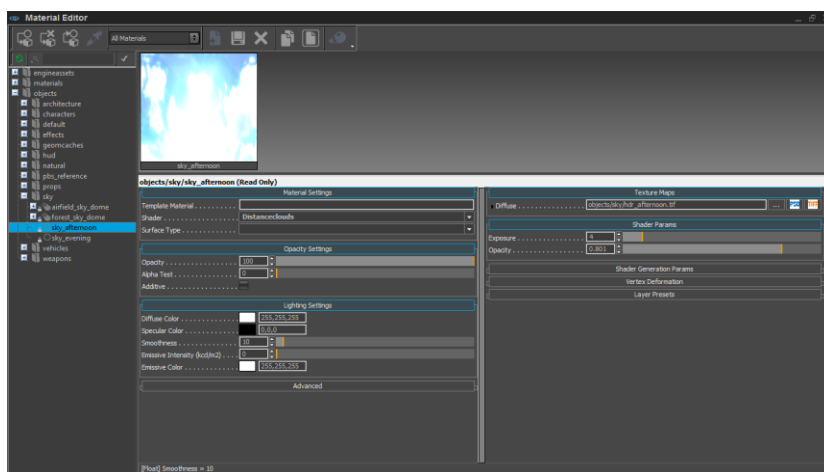


Flowgraph : این سیستم (که میراث کرای انجین ۳ است) با توجه به وجود نودها و گراف ها باعث کنترل و تغییرات در سطح مرحله در بازی می شود، مباحث پست پراسسینگ افکت (Post Processing Effect) یا امیج افکت ها (Image Effect Post Processing Filter) در اینجا وجود دارند، همچنین برای نمایش و کنترل منوهای ساخته شده در نرم افزار Adobe Flash و لود آن در کرای انجین برای استفاده در بازی از فلوگراف استفاده می کنیم، در واقع از این سیستم به برنامه نویسی بصری یا ویژوال اسکریپتینگ یاد می شود.



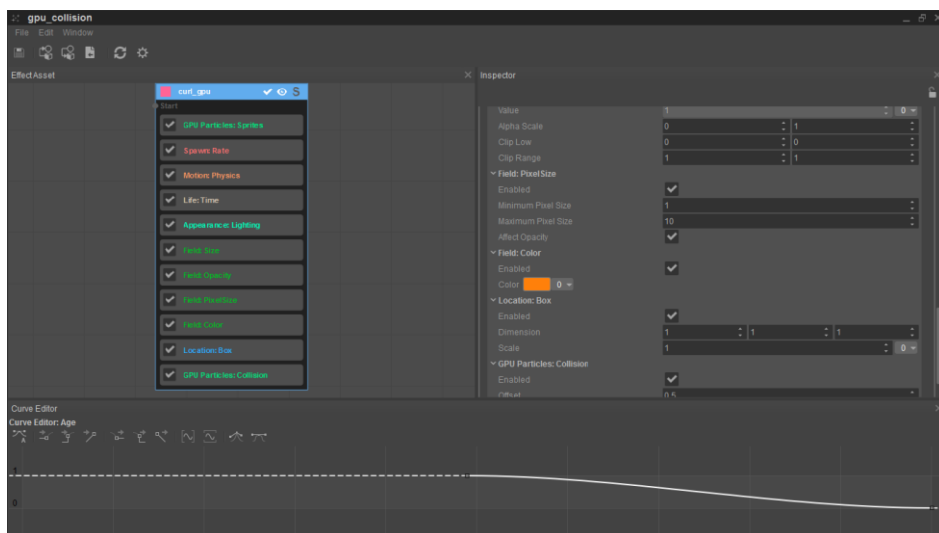
Lens Flare Editor : برای ساخت انواع لنزها برای نور خورشید و غیره از این پنجره استفاده می شود.

Material Editor : برای ساخت انواع مواد با توجه به وجود تکسچرها و شیدرها استفاده می شود، شما می توانید موارد مختلف مانند گچ، چوب، پلاستیک، نقره، آب، گل و لای و غیره بسازید، هر چه کیفیت تکسچرها بالاتر رود متریال ها بیشتر به واقعیت نزدیک می شوند، اگرچه نورپردازی و نوع شیدرها برای ساخت محیط های واقعی در کرای انجین پارامترهای دیگری هستند که شما را به ساخت محیط های واقعی کاملاً نزدیک می کند، مباحث ساخت مواد و آموزش آن در بحث این کتاب نمی گنجد.



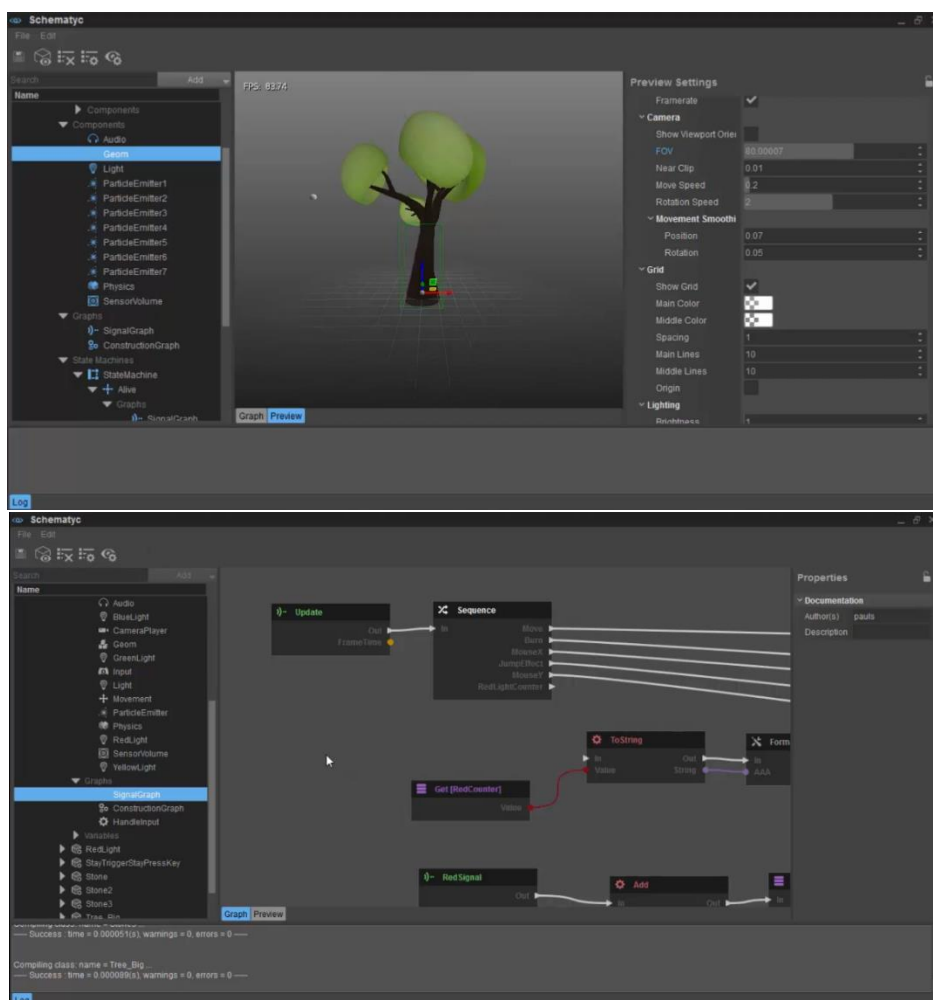
Particle Editor : وجود پارتيكل سيستم ها در هر بازی لازم و ضروری است، شما همیشه با این گونه از آبجکت ها برخورد می کنید، پارتيكل سيستم ها از ذرات ریزی تشکیل شده است که می توان این گونه ذرات را پارامترهای مختلف آن را تغییر داد، مثلاً ذراتی که باعث پدید آمدن دود شوند، یا ذراتی که آتش را به وجود می آورند، ذراتی مانند گردباد، ریزگرد، آبشار، بخار و غیره همگی آبجکت هایی از نوع پارتيكل سيستم هستند، در کرای انجین ۳ برای

ایجاد پارتیکل سیستم ها از پارامترهای مختلف استفاده میشد اما با ظهور کرای انجین ۵، سیستم جدیدی با نام Wavicle اضافه شده است که بسیار قدرتمند تر از کرای انجین ۳ است، این سیستم به صورت گراف عمل می کند و پارامترها را بر اساس اولویت انجام عملیات دسته بندی می شود، مباحث این سیستم در این کتاب نمی گنجد.



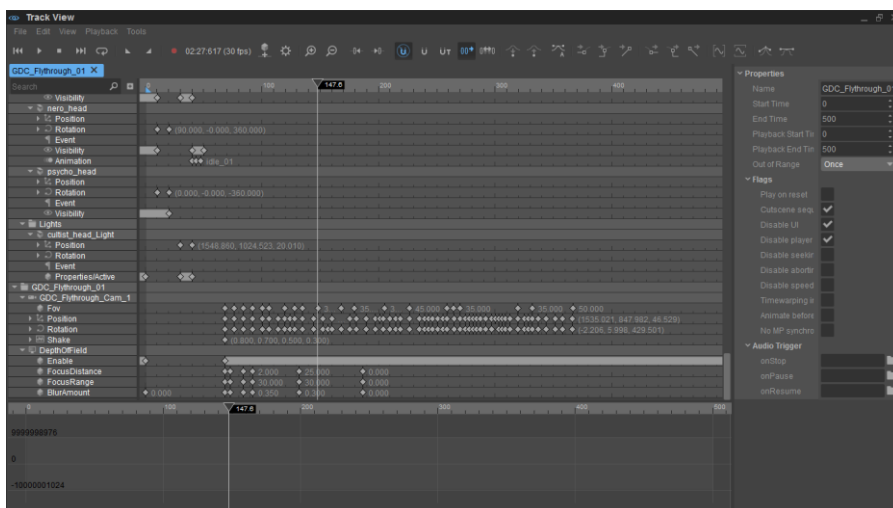
Schematyc Editor : با انتشار کرای انجین ۵،۳ انقلابی جدید در بحث برنامه نویسی بصری به وجود آمد، ساخت بازی ها در کرای انجین بدون کدنویسی! (واقعا Paul Slinger زیرکانه این سیستم را توسعه داد، Paul تو فوق العاده ای، من قبلش در توئیتر هم از ایشون تشکر کردم)، تبریک میگم به کسانی که حوصله کدنویسی را ندارند، چون سیماتیک اینجاست، در این کتاب سعی میکنم سیماتیک را نیز پوشش دهم و شما را با سیماتیک آشنا کنم، Template هایی در لانچر کرای انجین ۵،۵ برای سیماتیک تهیه شده است، این باعث می شود شما بسیار سریع بازی هایتان را با سیماتیک بسازید، البته دل سرد نشوید شما نمی توانید بازی کرایسیس ۳ را با سیماتیک

تولید کنید، شما باید زبان C++ را بدانید، اگرچه کرایتک سعی می کند
سیماتیک اولویت اول برای همه توسعه دهندگان بازی باشد.



Terrain Editor : برای ساخت، ویرایش و حذف، نرم کردن یا تغییر عوارض زمین
است، به این ترتیب می توانید هر آنچه از کوه، تپه، دره، جاده، کوهستان ها و غیره را
ایجاد کنید و تغییرات لازم را بر روی زمین بازی تان اعمال کنید.
Track View : اینجا برای ساخت انیمیشن های مربوط به مفهوم کات-سین
است، شما در بازی لازم دارید که قبل از شروع مراحل یا بین مراحل، نمایشی از سناریو

بازی^۲ را برای قهرمان داستان ارائه نمایید، مباحث مربوط به ساخت انیمیشن ها در این پنجره خارج از بحث این کتاب است.



Vegetation Editor: پوشش گیاهی و درختان و جنگل ها در بیشتر بازی ها وجود دارد، این پنجره به شما بیشترین امکان را می دهد که بر اساس نیازتان جنگل، باغ و مرتع را بر روی عوارض زمین بسازید، مراقب مثلث ها و چندضلعی ها درختان باشید، استفاده بیش از حد از درختانی که لوپولی^۳ نیستند باعث می شود سرعت اجرای بازی^۴ به شدت پایین بیاید.

Advance: دسترسی به پنجره های "Console"، "Notification"، "Python Scripts"، "Python Interactive Console"، "Center" برای کنترل و مدیریت انواع دستورها در سطح پوسته و هسته کرای انجین کاربرد دارند

Animation: دسترسی به پنجره های "Character Tool"، "Facial Editor"، "Mannequin Editor"،

برای دسترسی به انیمیشن های کاراکترها و نحوه کنترل آنها مورد استفاده قرار می گیرد

Deprecated : دسترسی به پنجره های در حال منسوخ "Dialog Editor" ،
 "Vehicle Editor" ، "Smart Object Editor" ، "Equip Pack Editor"
 این پنجره ها میراث کرای انجین ۳ است و تیم کرای انجین در حال جایگزین کردن
 سیستم های جدید است، مثلاً دسترسی به وسایل نقلیه در سیماتیک
 Designer Tool : دسترسی به پنجره های "Modeling" ، "UV Mapping"
 برای ساخت انواع مدل های سه بعدی و به غیر از کاراکترها قابل استفاده است، تیم
 کرای انجین تصمیم گرفته است که این ابزار را بیشتر توسعه دهد مانند آنچه که در
 مادباکس موجود است در این ابزار نیز در آینده طبق نقشه راه قابل دسترسی باشد، در
 حال حاضر این ابزار برای ساخت مدل ها و نقشه بندی برای مطابق با تسکچرها قدرت
 ۳دی مکس و مایا را دارد.

FBX Import : دسترسی به پنجره های "Mesh" ، "Animation" ،
 "Skeleton"
 برای وارد کردن مدل هایی که از تکنولوژی "فیلم-باکس" یا به اصطلاح امروز
 FBX در کرای انجین استفاده می شود، مدل های FBX یا فیلم-باکس دارای
 سه تعریف بر اساس انیمیشن (پویانمایی)-میش (مدل توری)-اسکلیتون
 (استخوان بندی) دارند که اگر شما می خواهید مدل هایی که دارای انیمیشن
 است را وارد کنید از پنجره انیمیشن و مدل هایی که فاقد انیمیشن هستند از
 پنجره میش و مدل هایی که دارای انیمیشن و اسکلت بندی هستند مانند
 کاراکترها از پنجره اسکلتون باید استفاده کنید، این سیستم جدیداً در نسخه ۵
 کرای انجین اضافه شده است و باعث استقبال عموم کاربران قرار گرفته
 است، قبلاً شما باید پلاگین هایی را داخل مکس-مایا و دیگر ۳d پلیکیشن ها
 راه اندازی می کردید تا مدل ها به فایل های *.CGF و یا *.CGA تبدیل می
 شدند و به داخل کرای انجین وارد می کردید اما با ارائه این سیستم

جدید، واردسازی مدل های سه بعدی کارها بسیار ساده تر شده است و مدل های دیگری با پسوند فایل های *.OBJ ، *.3DS ، *.DXF ، *.DAE (کولادا) را می توانید داخل کرای انجین Import یا اصطلاح وارد کنید و به صورت اتوماتیک فایل های کرای انجینی مدلینگ *.CGF و *.CGA و غیره ایجاد می شوند (به همین سادگی!)

Substance : دسترسی به پنجره های "Archive Graph Editor" ، "Graph" ، "Instance Editor" ، "Default Mapping Editor"

خوش آمدید به تکنولوژی Substance در کرای انجین ، خوش آمدید

شما با گراف های بسیار قدرتمند Substance در هر طیف و وسعتی می توانید بهترین شیدرها را خلق کنید، بهترین متریال ها را استفاده کنید و زیباترین آثار را در کرای انجین منتشر کنید، این تکنولوژی محبوب کاربران دنیاست، هر دوی این تکنولوژی فوق العاده هستند، Substance و CryEngine

Universal Query System : دسترسی به پنجره های "UQS Editor" ، "UQS History" ،

هوش مصنوعی در رابطه با نقطه های تاکتیکی و نقطه های پوششی برای حملات مرگبار با پیاده سازی الگوریتم های جدید با قدرت C++ و Schematyc کاملاً امکان پذیر شده است، خوش آمدید به الگوریتم های Universal UQS ، دنیای پیچیده تر را با این الگوریتم ها در قالب C++ و Schematyc ببینید و هوش مصنوعی بسیار پیشرفته تر کرای انجین اینجاست، کمی تحمل کنید من در حال تحقیقات بیشتر هستم و نتایج آن را منتشر خواهم کرد.

نوار ابزارها که مهم ترین و پرکاربرد ترین ابزارها را از میان منوها نیز که قابل دسترس است با تزئین به صورت دکمه های آیکن دار قابل مشاهده است





برای انتخاب اشیاء استفاده می شود



برای جابه جایی اشیاء و عمل انتقال اشیاء از یک مکان به مکان دیگری است



برای چرخش اشیاء استفاده می شود



برای بزرگ کردن و کوچک کردن اندازه اشیاء استفاده می شود



برای جابجایی اشیاء و چرخش اشیاء در دو محور مختصات انجام می شود



برای جابجایی اشیاء و چرخش اشیاء به صورت محلی با سه محور مختصات انجام می شود



برای جابجایی اشیاء و چرخش اشیاء به صورت شی والد با سه محور مختصات انجام می شود



برای جابجایی اشیاء و چرخش اشیاء به صورت مختصات جهانی با سه محور استفاده می شود



برای فوکوس دوربین و جا به جایی دوربین بر روی شی ای که در فاصله و مسافتی دور یا نزدیک است استفاده می شود، ابتدا شی را انتخاب کنید و سپس بر روی این دکمه کلیک کنید، دوربین بر روی شی قرار می گیرد که اصطلاح فوکوس گفته می شود.

برای پیدا کردن یک شی از میان صدها شی یا هزاران شی در مرحله بازی کاربرد دارد.



برای ایجاد گروهی از اشیاء استفاده می شود، طراحان مرحله (Level Designer ها) به شدت علاقه مند هستند زیرا که کارهایش را آسان می کند.



برای خارج کردن گروه و شکستن گروه و توانایی ایجاد تغییرات و ویرایش گروه بندی استفاده می شود.



برای ارتباط دادن آبجکت ها و نسبت هر آبجکت به آبجکت دیگر استفاده می شود، اگرچه این ابزار قدیمی است اما هنوز کاربرد خود را از دست نداده است.



همیشه آبجکت ها یا اشیاء ارتباطات شان دائمی نیست و مواردی از انصراف از این ارتباطات وجود خواهد داشت



ساخت پریفب هایی که غیر سیماتیک هستند را در اینجا می توان یافت، اگرچه مفهوم سیماتیک مانند پریفب است و به مفهوم برنامه نویسی غیر کد و با گراف ها هم ارز است، ساخت پریفب ها هنوز در کرای انجین وجود دارد



منجمد کردن اشیاء در مرحله به صورتی که نتوان آن را انتخاب، جا به جایی، چرخش، تغییر اندازه داد



خارج کردن کلیه اشیاء منجمد شده در مرحله



اجرا کردن بازی



فعال سازی و اجرا کدهای فیزیک و کدهای هوش مصنوعی



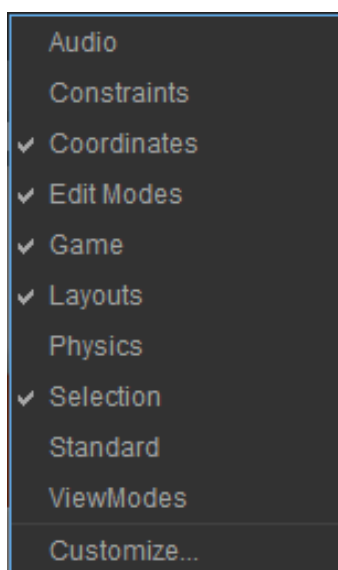
توقف موقت در اجرای کدهای فیزیک و کدهای هوش مصنوعی



برای اجرای یک فریم از کدهای فیزیک و هوش مصنوعی

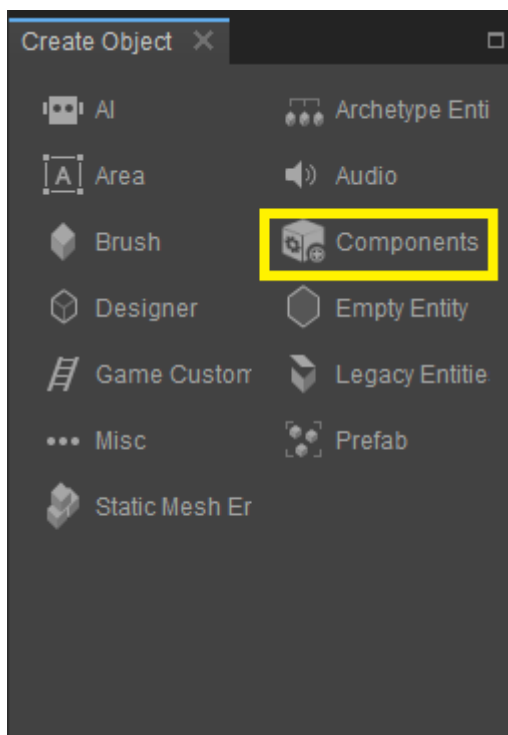


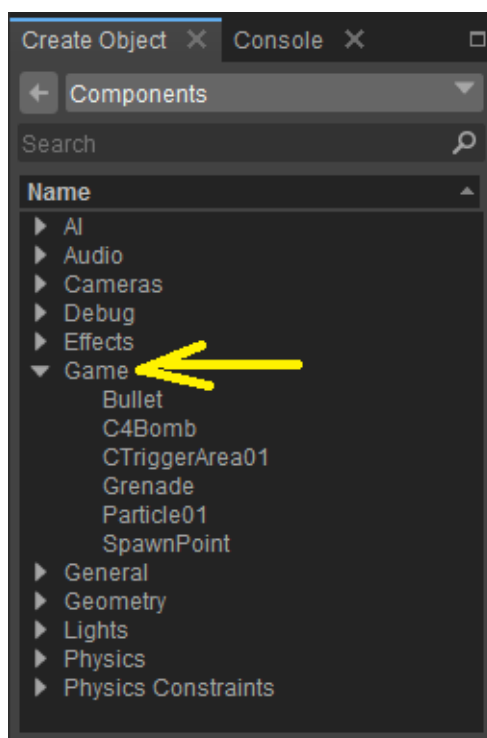
با راست کلیک کردن بر روی نوار ابزار می توانید به گزینه های بیشتری برای حذف و اضافه کردن ابزارها دسترسی داشته باشید.



و حالا به بررسی مختصر و مفید ۷ پنجره در Layout پیش فرض می پردازم:

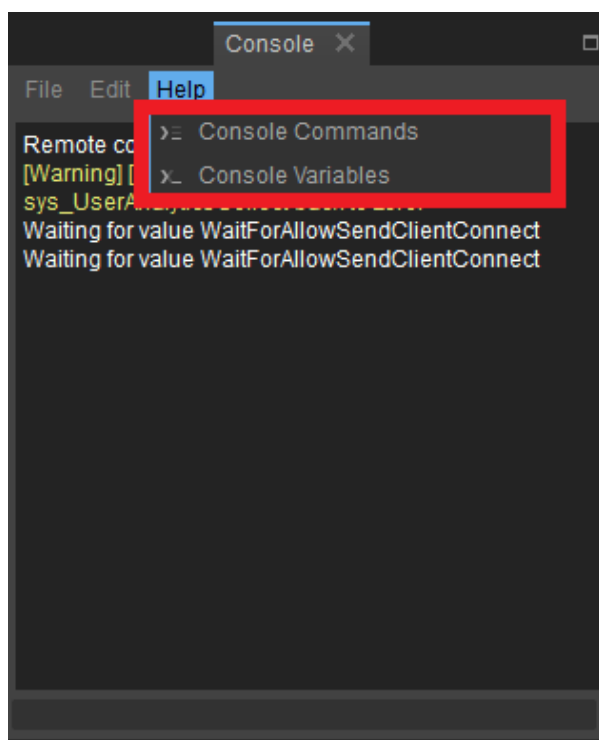
۱- پنجره Create Object : همانطور که از نام آن پیداست، هر آنچه که از آبجکت ها یا اینتیتی ها را از طریق این پنجره ایجاد می کنیم، کلیه کدهای نوشته شده C++ هایمان در دکمه کشویی Components به صورت کامپونت ها با کادر زردرنگ قابل نمایش است





بعد از کامپایل کدهای C++ یمان در کاتالوگ Game لیستی از کامپوننت هایی که بوسیله کدهای C++ ایجاد شده است را برای مان نمایان می شود.

۲- پنجره Console : با توجه به فرامین و متغیرها فراوان برای کنترل سندباکس در محیط ادیتور و کنترل بازی در سطح RunTime، کاربرد این پنجره در بازی بسیار مهم است و می توان با کدهای C++ به این فرامین و متغیرها دسترسی کامل در زمان اجرا بازی داشت، مثلاً تغییر زمان روز - فعال سازی یا غیر فعال سازی مه - تغییر ریزولیشن بازی و غیره




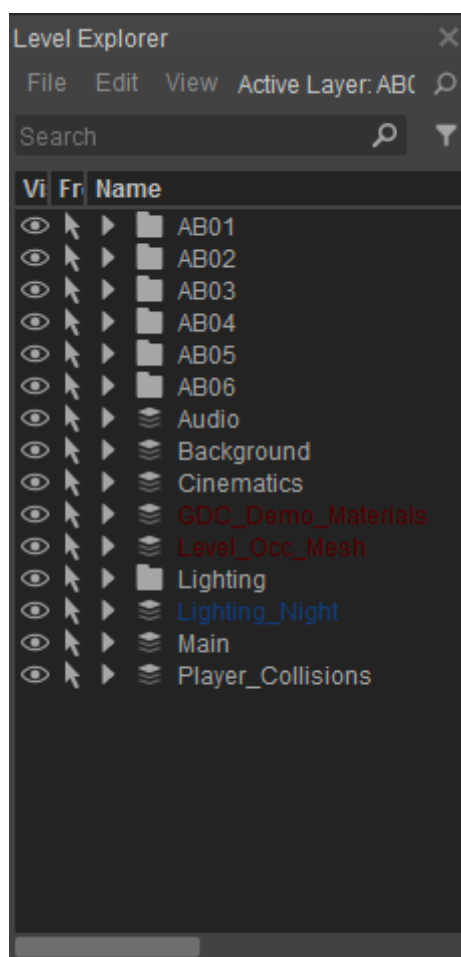
گزینه های مورد اشاره در توضیحات بالا در کادر قرمز رنگ نمایش داده شده است که با کلیک کردن برروی هر یک از این دو گزینه به انبوهی از دستورات پرکاربرد در سندباکس و بازی دسترسی خواهیم داشت، در این کتاب گزینه "Console Variables" را به خوبی پوشش خواهیم داد، زیرا در ارتباط با اجرای کدهای در سطح بازی است.

۳- پنجره Level Explorer : هر آبجکت یا اینتیتی که در بازی ایجاد می شود برای مدیریت به صورت دسته بندی پوشه ای، دسته بندی لایه ای، ویرایش اسم اینتیتی، حذف اینتیتی و جستجوی اینتیتی و غیره به وسیله این پنجره انجام می شود، در میان انبوهی از اشیاء بازی جستجو کنید و آنچه که در نظر دارید را پیدا

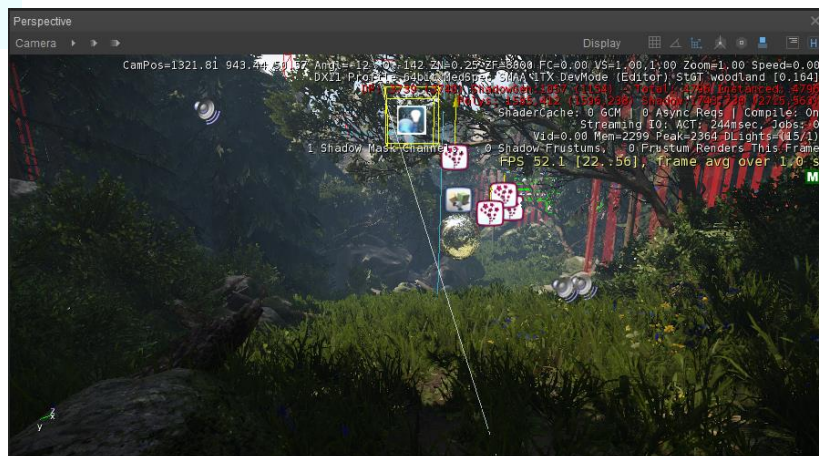
Search



کنید، شما می توانید از فیلترهای مختلف  برای این جستجو یا جستجو ها استفاده کنید.

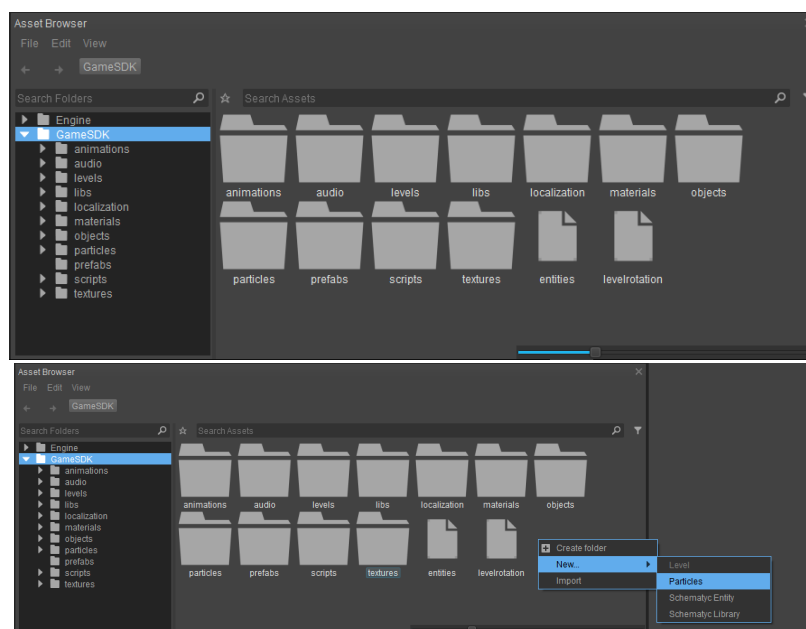


۴- پنجره Perspective : کلیه مراحل ساخت بازی با تنوع بسیار زیادی از اینتیتی ها با توجه به اجرا بازی داخل این پنجره انجام میشود و این پنجره از بخش های مختلفی تشکیل شده است.



نمایی از پروژه GameSDK را در پنجره Perspective مشاهده می کنید و اگر دقت کنید داخل این پنجره نیز شامل دکمه های مختلفی است (سمت راست گوشه بالا) که سبک و نحوه نمایش محیط بازی را برای ما تغییر می دهد.

۵- پنجره Asset Browser : این بخش در ارتباط با مدیریت کلیه فایل های صوتی-انیمیشن-پارتیکل-مدلها-تکسچرها-پریفب ها و غیره و در یک کلام برای حذف و اضافه و ویرایش فایل ها مختلف و متفاوت (به این فایل ها Asset گفته می شود) پروژه است، شما در یک جای خالی از این پنجره راست کلیک کنید تا منوی Asset مانند سیماتیک-پارتیکل سی شارپ یا هر چیزی که لازم دارید را نمایش دهد و سپس آن را انتخاب کنید (چقدر لذت بخش است)





در کرای انجین ۵,۵ تعداد گزینه ها بیشتر شده است و این تصویر در کرای انجین ۵,۴ است، همانطور که می بینید در یک جای خالی راست کلیک شده و لیستی از Asset ها نمایش داده شده است، در میان انبوهی از Asset ها که در پروژه تان وجود دارد می توانید جستجو کنید و آنچه که در نظر دارید را پیدا کنید، می توانید از فیلتر های مختلف استفاده کنید.


۶- پنجره Properties : این پنجره برای نمایش کلیه خصوصیات و پارامترهای هر اینتیتی موجود در پنجره Level Explorer است، وقتی که کدهای C++ را با عنوان اضافه کردن Member در بخش ثبت Reflect های کامپونت مد نظر قرار دهیم، پارامترها و خصوصیات در اینتیتی نوشته شده در C++، داخل این پنجره نمایش داده خواهد شد، در این کتاب به این مهم خواهیم پرداخت.


۷- پنجره Terrain Editor : این پنجره برای ویرایش عوارض زمین، اضافه کردن کوه ها، دره ها، دریاچه ها و غیره استفاده می شود.


این پنجره ها در Layout پیش فرض کرای انجین است، این Layout ها در نوار ابزارها واقع هستند که شکل پنج Layout مختلف را در زیر می بینید که به ترتیب از چپ به راست عبارتند از :

 Default Layout : چیدمان پنجره ها بر اساس تعریف نام گذاری شده به صورت پیش فرض است و بیشتر کاربران از این Layout استفاده می کنند

 Animation Layout : هنگامی که برروی کاراکترهای بازی در حال کار هستیم، چیدمان پنجره ها بهتر است برروی این Layout باشد.

 Cinematic Layout : در هنگام کار کردن برروی کات-سین و طراحی نمایش سینمایی اتفاقات بازی، چیدمان پنجره ها بهتر است برروی این Layout باشد.

 Level Design Layout : در زمانی که طراح یا طراحان مراحل بازی برروی پروژه کار می کنند، چیدمان پنجره ها بهتر است برروی این Layout باشد

 Flowgraph Layout : زبان فلوگراف نیز بسیار گسترده است و در سطح مرحله یا مراحل بازی کاربردهای بسیار زیادی دارد، اگرچه این زبان نیز میراث کرای انجین ۳ است اما هنوز نیز در کرای انجین ۵ کاربرد دارد، خصوصا هنگامی که برروی پست پراسسینگ افکت یا امیج افکت کار

می کنید، چیدمان پنجره ها بهتر است بر روی این Layout باشد زمانی که شما در حال پروگرافینگ بازی هستید.



بر روی هر یک از Layout ها کلیک کنید و نتیجه تغییر و چینش پنجره ها را مشاهده خواهید کرد، لازم به ذکر است که این Layout ها بر روی سه مانیتور و یا چهار مانیتور لود شده و به طراح بازی این کنترل را می دهد که هر مانیتور بخشی از محیط سندباکس کرای انجین را نمایش دهد و آنچه که در شرکت های بازی سازی می بینیم و استفاده می شود. اگر به دنبال دستور یا فرمان یا گزینه ای از میان منوها می گردید، ابزار جستجو بسیار مفید است



همچنین هر گونه پیغام های خطا (قرمز رنگ)، هشدار (زرد رنگ) (در کنار ابزار جستجو در سمت راست است، این پیغام ها به شما کمک می کند تا بهتر بتوانید مشکلات موجود در داخل پروژه تان را رفع کنید این فصل مروری سریع و البته مفید به ابزارها و گزینه های مختلف بود تا بهتر با کرای انجین ارتباط برقرار کنید، این مفاهیم شما را آماده می کند تا برنامه نویسی در کرای انجین را آغاز کنید.

برای درک مطالب این فصل می توانید به آموزش های ویدئویی بیشتر در اکانت یوتیوب من یا آپارات من و در صورت اینکه به اینترنت دسترسی ندارید به حداقل آموزش های من بر روی DVD در ضمیمه این کتاب مراجعه بفرمایید

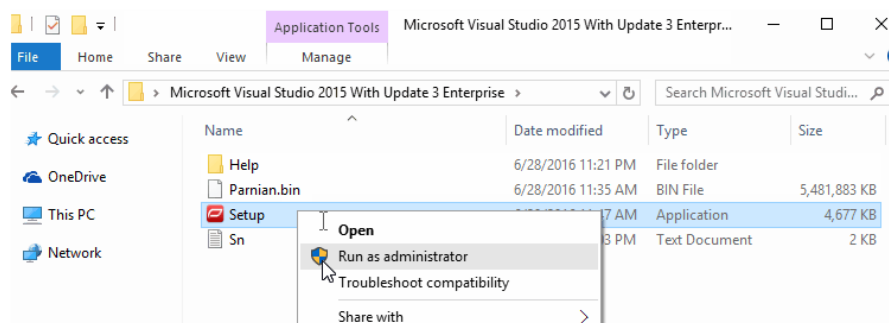
فصل چهارم

نحوه نصب Visual Studio ۲۰۱۵

Enterprise برای C++ CryEngine

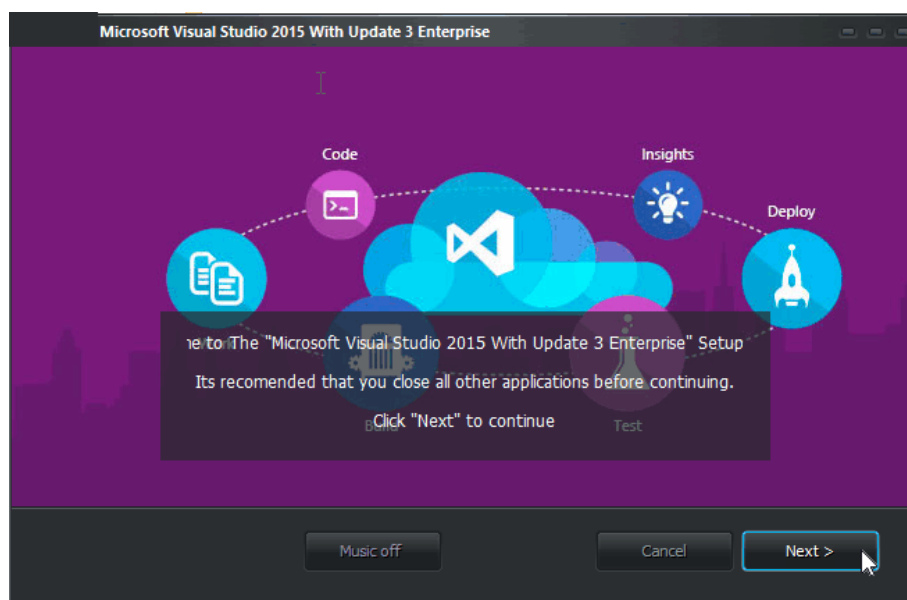
من نحوه نصب ۳ Visual Studio Enterprise ۲۰۱۵ Update را به شما آموزش می‌دهم، این کار ظاهراً خیلی ساده است اما همه کارهای ساده باید به درستی انجام شوند تا وارد کارهای متوسط و حتی سخت شویم، این یک استاندارد در کتاب‌های آموزشی است، مطمئناً نحوه نصب درست و ویزوال استودیو به شما کمک می‌کند تا وارد دنیای برنامه‌نویسی بازی‌ها با زبان قدرتمند C++ شوید، حتماً قبل از نصب آنتی‌ویروس را غیر فعال نموده و اینترنت را قطع کنید

بهتر است که دسترسی نصب را به بالاترین اولویت ریشه یا Administrator به برنامه اختصاص دهید پس طبق تصویر زیر بر روی فایل `setup` راست کلیک کرده و گزینه `Run as administrator` را انتخاب کنید، اگر دقت کنید یک فایل متنی `SN` وجود دارد که سریال‌های مختلف را برای فعال‌سازی ویزوال استودیو در خود دارد، روش کار برای نصب انواع ویزوال استودیو با `Version`‌های مختلف و متفاوت طبق تصاویر زیر تقریباً یکسان است

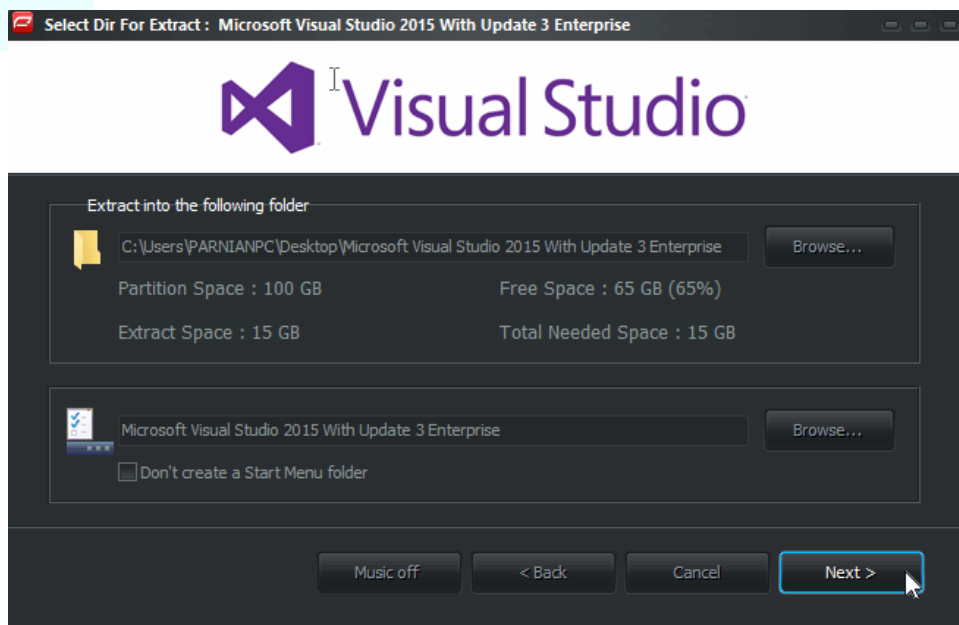


با اجرا شدن فایل `setup`، نوبت آن است که فایل‌های حجیم از حالت فشرده خارج شوند و طبق تصویر زیر، برنامه خارج کردن فایل‌ها از فشرده‌گی اجرا شود و فضایی را برای اکستراکت کردن فایل بر روی `Hard-disk` باید

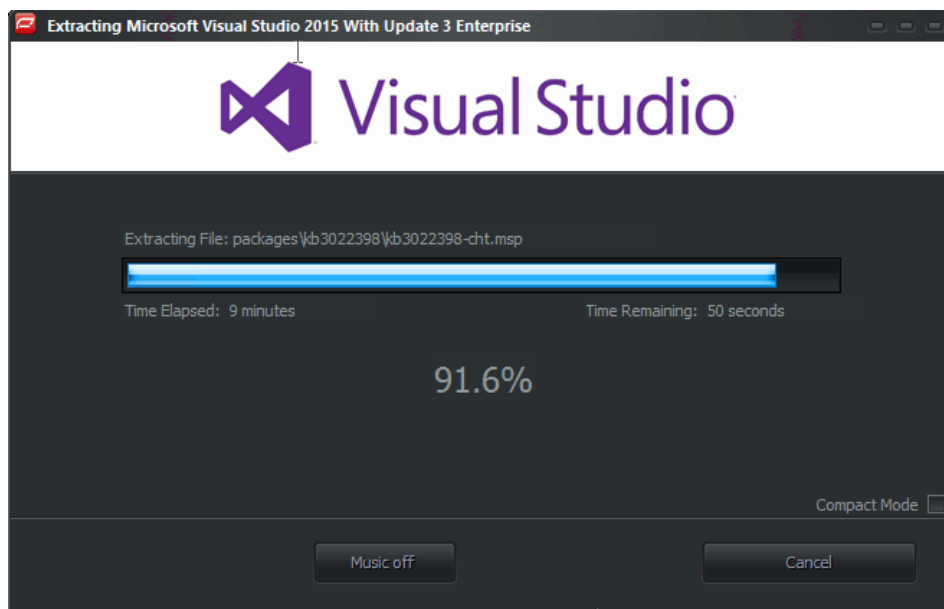
اختصاص یابد، پس بر روی دکمه Next کلیک کنید تا وارد مرحله بعدی شوید



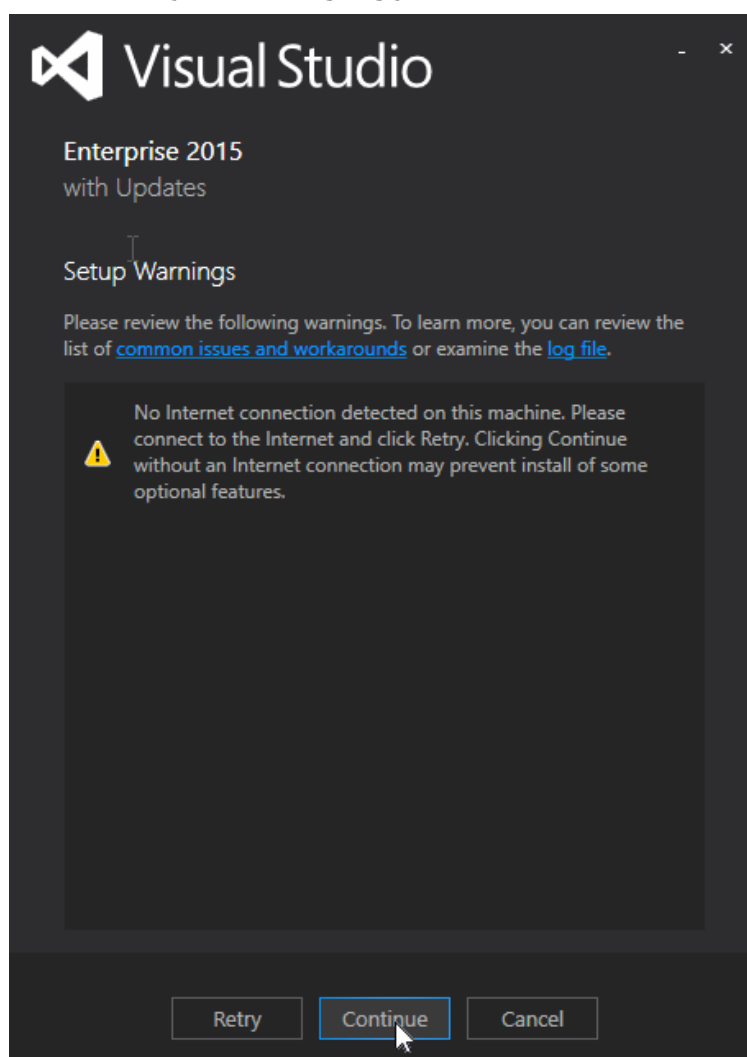
در تصویر زیر مشخصات فضای هارد دیسک را بر روی پارتیشنی که می خواهید ویژوال استودیو را از فشردگی خارج کنید را نشان می دهد که در این تصویر ۱۵ گیگا بایت فضا را برای نصب ویژوال استودیو می خواهد و ۶۵ گیگابایت فضای خالی در پارتیشن فعلی یعنی درایو C موجود است و کل حجم پارتیشن (درایو C) شامل ۱۰۰ گیگا بایت است و همانطور که می بینید در تصویر زیر مسیری از پوشه های تودرتو که فایل های ویژوال استودیو برای خارج شدن از فشردگی را نشان می دهد که می توانید مسیرش را عوض کنید، برای ادامه عملیات دکمه Next را کلیک کنید



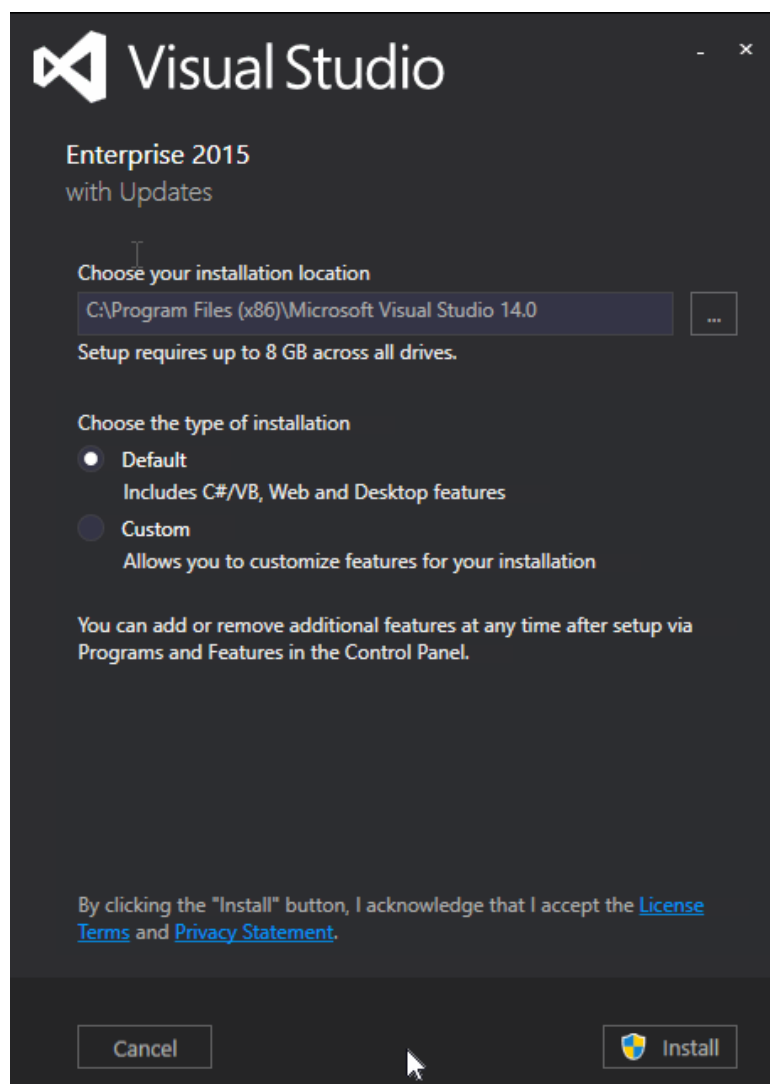
بعد از مدت چند دقیقه فایل ها از فشرده‌گی خارج شده و بر روی مسیر مشخص شده شما در هارد دیسک قرار می گیرند و انجام این عملیات مثل تصویر زیر خواهد بود



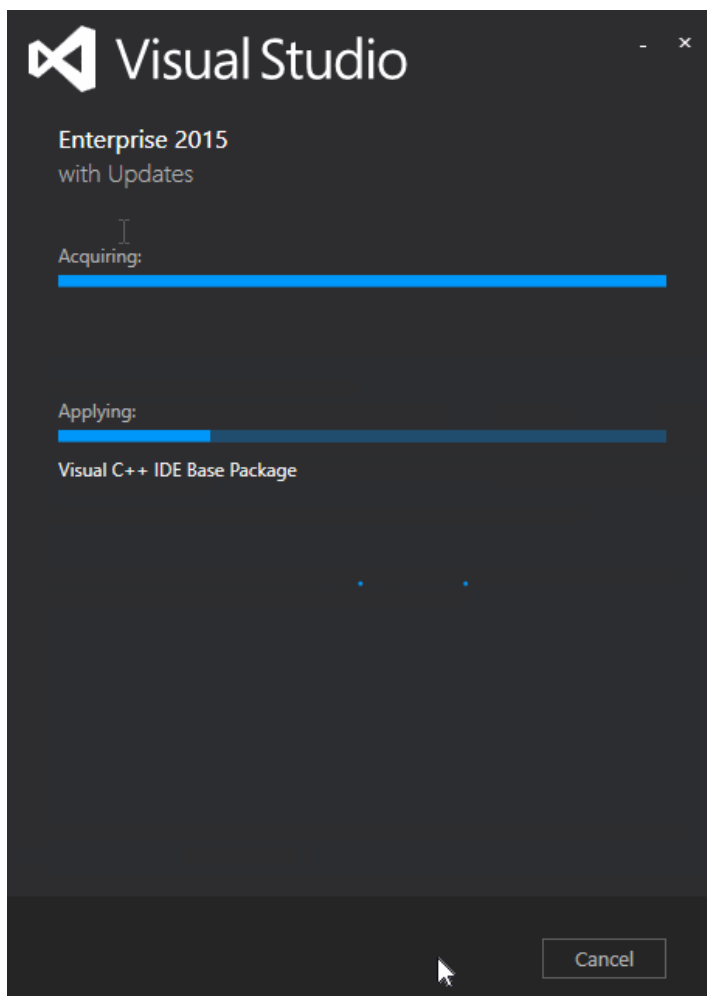
حالا وارد مرحله ای برای نصب ویژوال استودیو می شوید، بدون توجه به پیغام هشدار که به اینترنت وصل نیستید، باید کار را ادامه دهید، دقت کنید نباید به اینترنت متصل باشید و آنتی ویروس تان باید غیرفعال باشد، دلایل این کار به قانون کپی رایت بر میگردد که کشور ایران این قانون را اجرا نمی کند، حالا بر روی دکمه Continue کلیک کنید تا وارد مرحله بعدی شوید



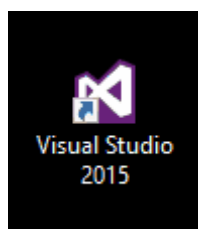
در تصویر زیر می بینید که مسیر نصب ویژوال استودیو به شما پیشنهاد داده می شود و در اینجا عدد ۱۴ یعنی ویژوال استودیو ۲۰۱۵ و عدد ۱۲ به معنی ویژوال استودیو ۲۰۱۳ است، دقت کنید که برای دسترسی به بیشترین تنظیمات نصب حتما گزینه Custom را انتخاب کنید و طبق تصویر زیر گزینه Default برای ما کارایی ندارد زیرا که باید تمام option ها یا همان گزینه های C++ در گزینه Custom باید تیک خورده شود، تمامی گزینه ها در ارتباط به کامپایلرها و نوع کامپایل و بیشترین ابزارهای کامپایل را با انتخاب تمامی تیک ها در ارتباط با C++ را هرگز فراموش نکنید، زیرا که ما بر روی برنامه نویسی C++ در کرای انجین تمرکز کرده ایم.



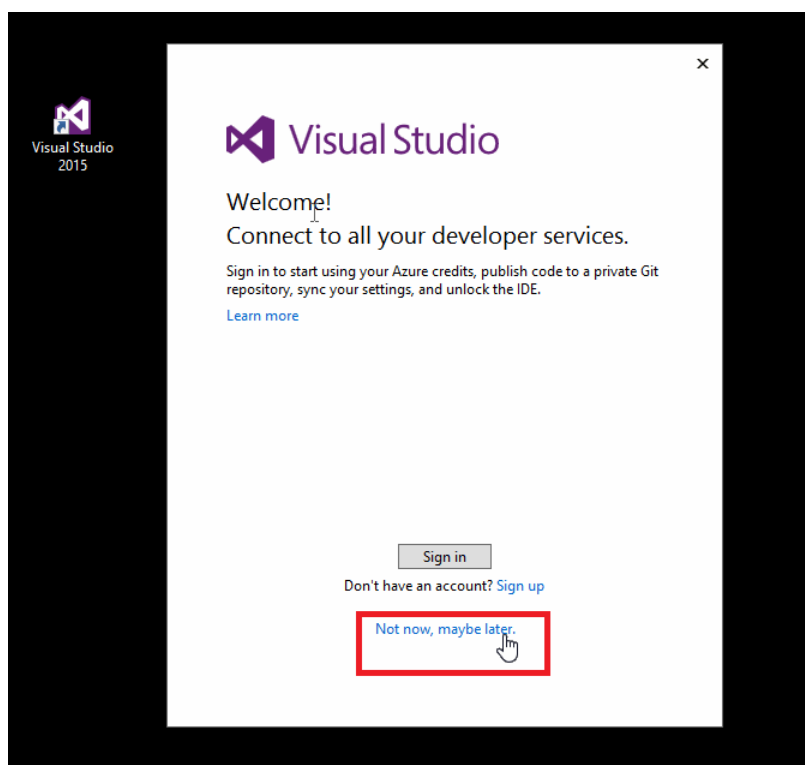
بعد از انتخاب همه تیک ها (checkbox) در ارتباط با زبان C++، نصب
ویژوال استودیو ۲۰۱۵ آپدیت ۳ آغاز می شود و همانند آنچه که در تصویر
زیر می بینید باید بروی کامپیوترتان نمایش داده شود.



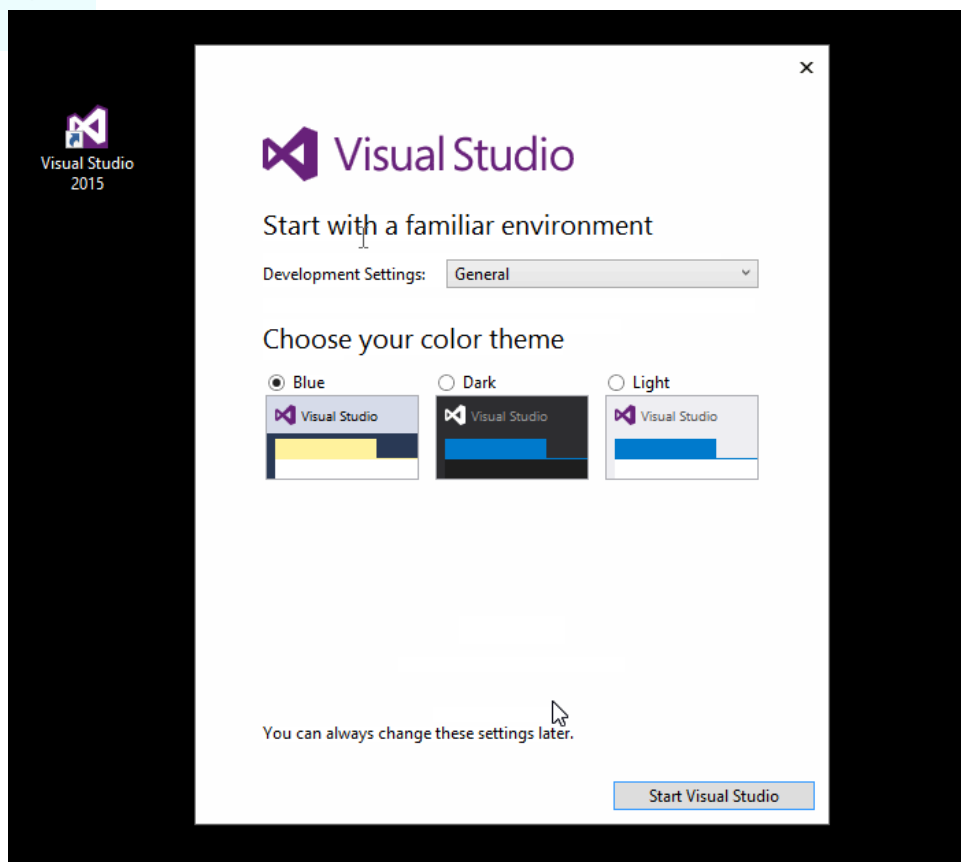
بعد از اتمام مراحل نصب و نصب تمام گزینه ها و در آخر کلیک کردن بر روی دکمه Finish، بر روی Desktop آیکن ویژوال استودیو ۲۰۱۵ زیر ایجاد شده و منتظر اجرا شدن است



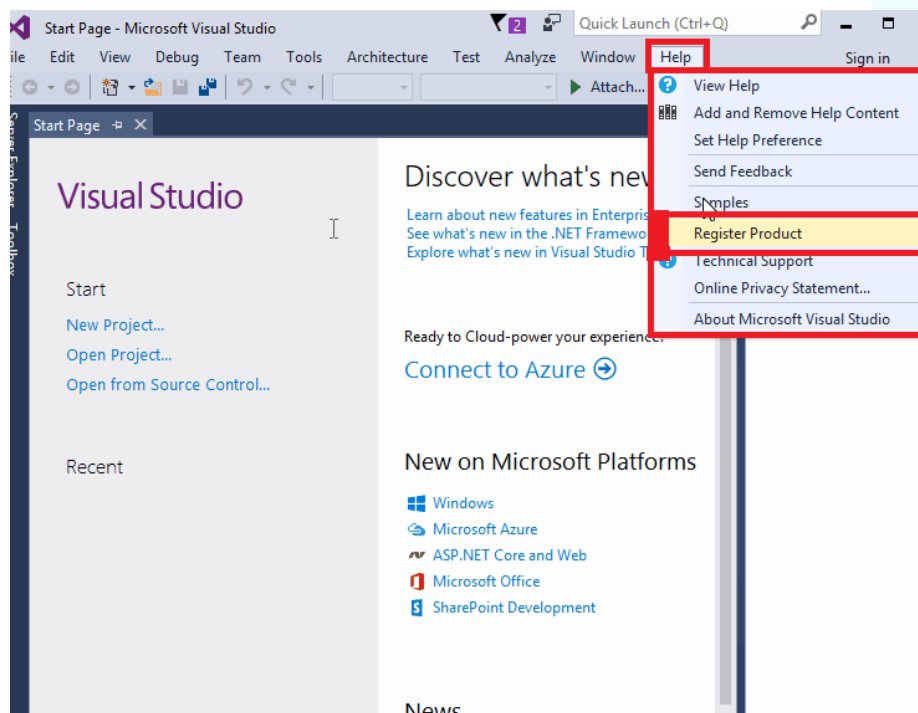
و حالا برروی آیکن دوبار کلیک کنید و بعد از اجرا ویژوال استودیو، پنجره زیر مشاهده می شود و در کادر قرمز رنگ لینک دکمه ای را می بینید با نام **Not now, maybe later** ، و برروی این جمله کلیک کنید تا مراحل رجیستر کردن ویژوال استودیو را انجام دهید



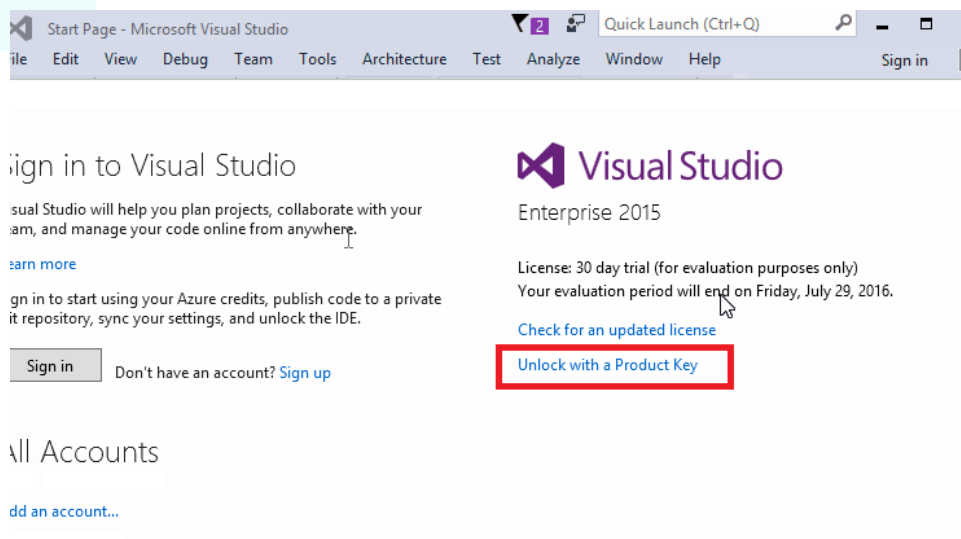
حالا طبق تصویر زیر با انتخاب تم (Theme) ویژوال استودیو وارد مرحله بعدی از رجیستر شدن ویژوال استودیو می شوید، در اینجا شما می توانید تم Blue (آبی) ، Dark (تیره) ، Light (سبک) را انتخاب کنید، در اینجا از تم پیش فرض یا همان Blue استفاده کنید و سپس برروی دکمه **Start Visual Studio** کلیک کنید تا وارد مرحله بعدی شوید.



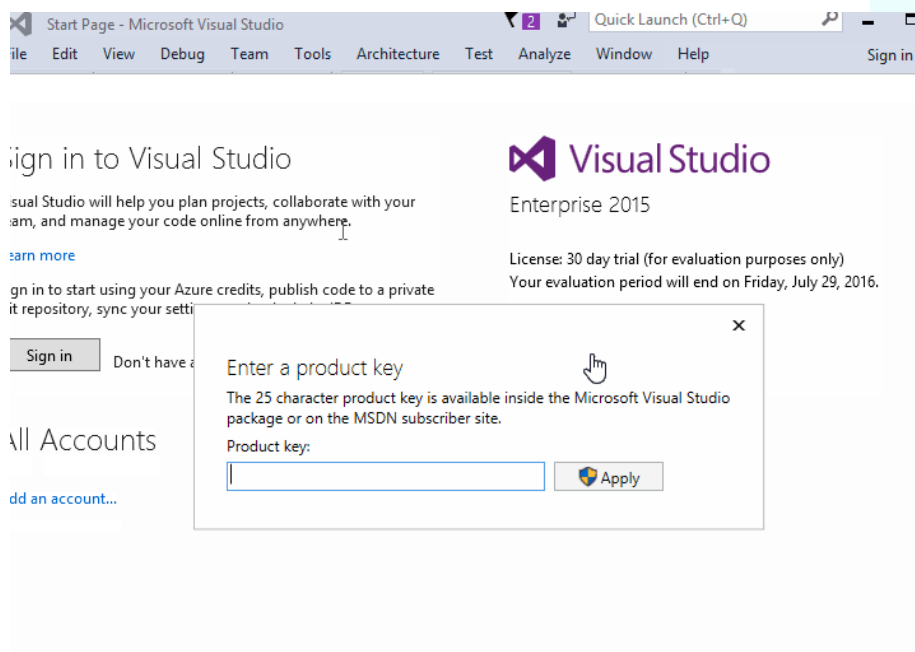
ویژوال استودیو برای شما باز می شود و به منوی Help مراجعه کنید و گزینه Register Product را انتخاب کنید



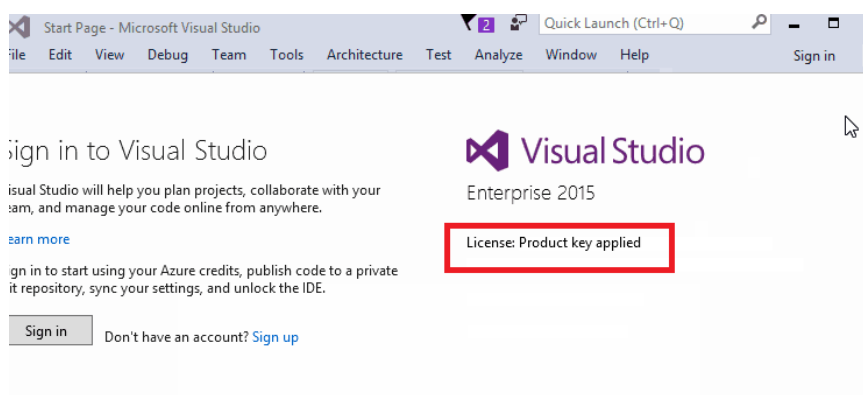
همانطور که در تصویر زیر مشخص است، ویژوال استودیو ۳۰ روز رایگان است و بعد از ۳۰ روز باید آن را خریداری کنید و نحوه رایگان و دائمی شدن ویژوال استودیو در مرحله بعدی توضیح داده ام، پس برروی دکمه لینکی Unlock with a Product Key کلیک می کنید تا وارد مرحله پایانی رجیستر شدن شوید.



داخل DVD ویزوال استودیو فایلی باید با اسم SN یا Serial وجود داشته باشد که با انجام Copy کد سریال متنی از آن فایل و سپس Paste کردن آن کد سریال داخل پنجره در تصویر زیر و کلیک کردن برروی دکمه Apply، عمل ثبت و ریجیستر کردن ویزوال استودیو برای همیشه انجام می شود و استفاده فقط ۳۰ روزه از این نرم افزار غیر فعال می شود و استفاده از نرم افزار دائمی می شود



همانطور که می بینید داخل کادر قرمز رنگ در تصویر زیر عمل ثبت و رجیستر کردن با موفقیت انجام شده است



تنها دو گام با شروع برنامه نویسی C++ باقی مانده است، شما باید بدانید:

۱- چیسیت و چگونه نصب می شود CMake

۲- ایجاد Solution ها با توجه به وجود Template های مختلف چگونه ممکن است؟

در فصل بعدی این دوگام را تا شروع برنامه نویسی C++ خواهید پیمود و به
جواب سوالات بالا خواهید رسید

فصل پنجم

نحوه نصب CMake و ایجاد

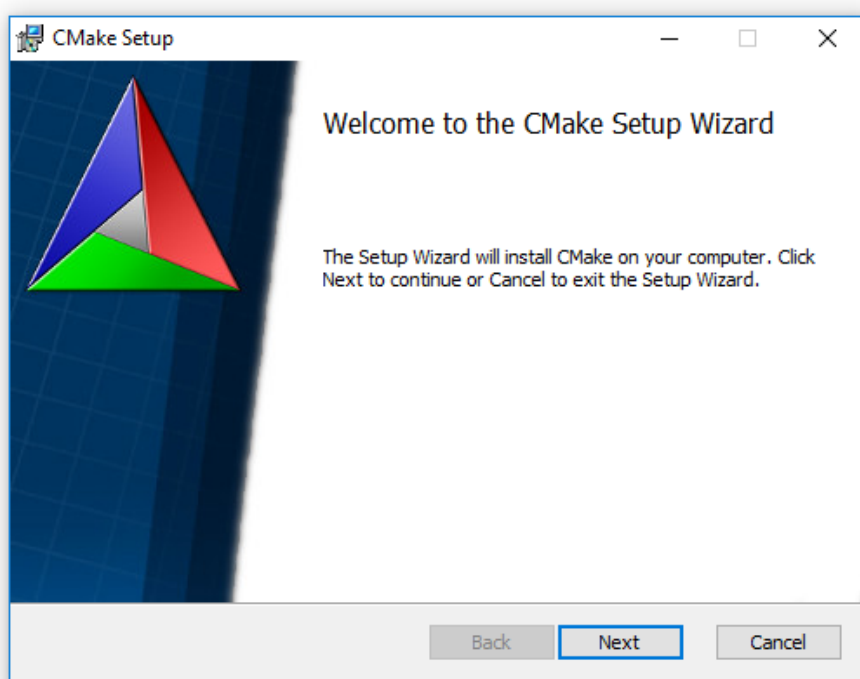
Solution ها برای Template های

مختلف

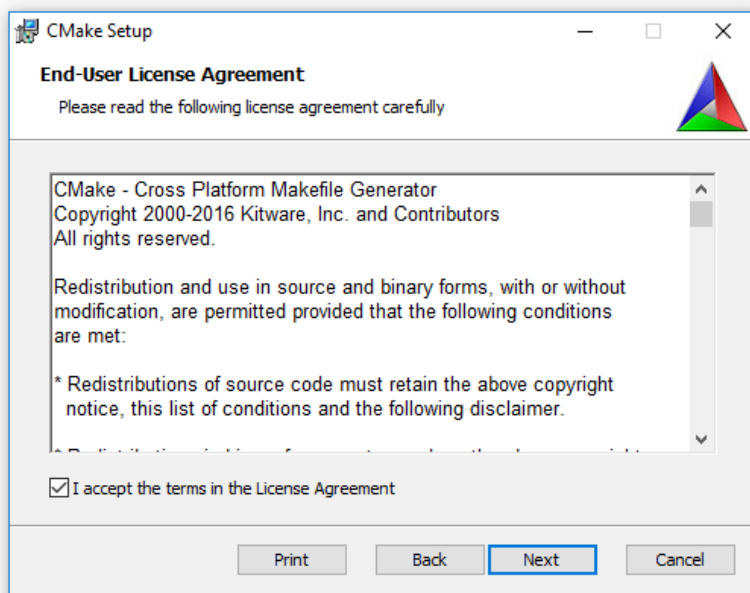
۱- به ترجمه کدهای زبان برنامه نویسی مانند C++ به زبان صفر و یک که زبان کامپیوتر است، کامپایل یا همگردانی گفته می شود، اگر خطا یا خطاها در کدها باشد لیست آن برای برنامه نویس یا برنامه نویسان در ویژوال استودیو نمایش داده می شود، ویژوال استودیو یک محیط توسعه برای کدها (IDE) برای تایپ، ویرایش، اضافه، کاهش، حذف کدها با قابلیت خطایابی است، دو محیط از توسعه کدها را می توان به نرم افزارهای MonoDevelop و Visual Studio اشاره کرد، ویژوال استودیو محبوب ترین و قدرتمندترین محیط برای توسعه کدها برای انواع دستگاه های سخت افزاری با سیستم عامل های مختلف است.

برای بازتولید کدهای C++/QT Sandbox و کدهای C++ پروژه ها در Template ها در ویژوال استودیو و همچنین راهنمایی و نحوه ایجاد Solution توسط ویژوال استودیو و لیست کامپایل شدن کدهای (C++ File) *.cpp و کدهای (Header) *.h از برنامه CMake استفاده می کنیم، برای نصب این برنامه و تولید فایل Solution ویژوال استودیو نسخه ۲۰۱۵ اینترپرایز آپدیت ۳ مراحل زیر را دنبال کنید:

ابتدا فایل CMake.msi را بر روی DVD در ضمیمه این کتاب را پیدا کنید و دوبار بر روی آن کلیک کنید تا پنجره خوش آمدگویی زیر نمایش داده شود، با کلیک بر روی دکمه Next به مرحله بعدی می روید و با کلیک بر روی دکمه Cancel انصراف خود را از نصب این برنامه به ویندوز اعلام می کنید و البته دکمه Back نیز غیر فعال است.

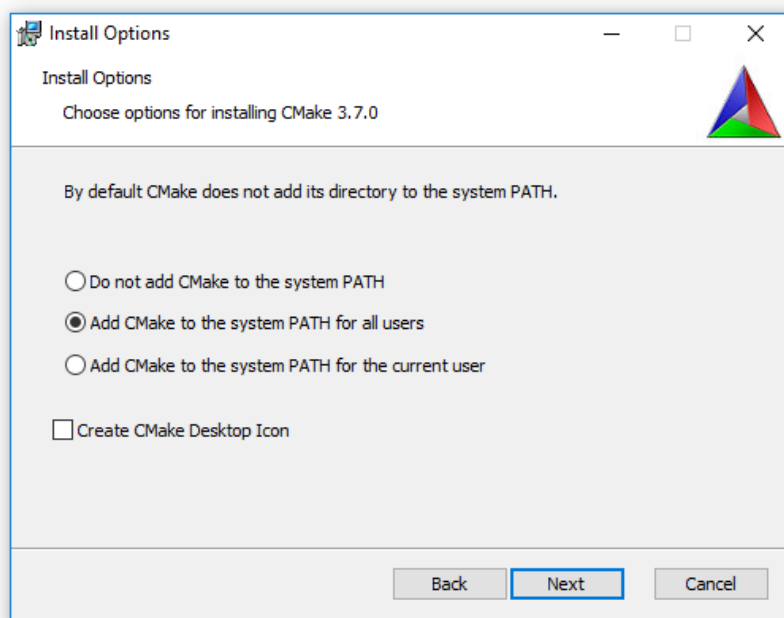


در این پنجره طبق تصویر زیر مشاهده می کنید لایسنس برنامه طبق قوانینی که از طرف شرکت سازنده وجود دارد را برای شما با جزئیات کافی و لازم نمایش می دهد و شما با انتخاب گزینه موافقت با کلمات و جملات این لایسنس و کلیک برروی دکمه Next، وارد مرحله بعدی از مراحل نصب برنامه CMake می شوید، شما می توانید با کلیک برروی دکمه Print، لایسنس این نرم افزار را چاپ و مطالعه کنید و با کلیک برروی دکمه Back وارد مرحله قبلی شده و با کلیک برروی Cancel، از نصب برنامه صرف نظر کنید

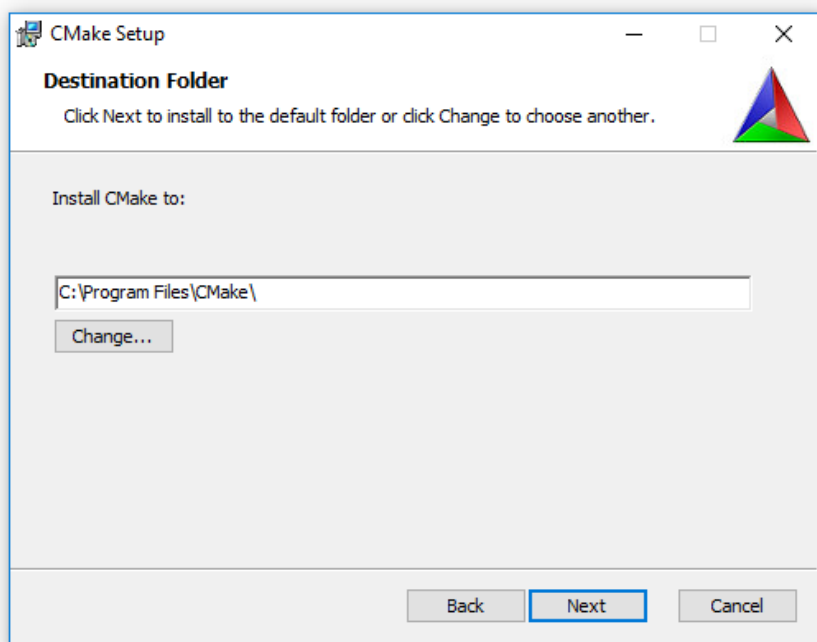


آخرین نسخه ای که شما می توانید در ضمیمه این کتاب از آن استفاده نمایید تا کدهای C++ تولید شود ۳,۷ Cmake است، در مرحله بعدی طبق تصویر زیر گزینه اول، شما مسیر را به سیستم مسیر در سیستم عامل ویندوز را ثبت نمی کنید و در این مورد باید صرف نظر کنید مگر در موارد خاص و گزینه دوم همه کاربران با همه اکانت ها می توانند به CMake دسترسی داشته باشند و گزینه دوم بهترین انتخاب است و در تصویر زیر نیز این گزینه انتخاب شده است و گزینه سوم CMake تنها در اکانت جاری قابل استفاده و ثبت خواهد شد و در اکانت های دیگر کاربران موجود نخواهد بود، مبین و روشن است که گزینه سوم مانند آنچه که در گزینه اول وجود داشت، برای ما انتخاب مناسبی نیست و شما با کلیک کردن بر روی گزینه وسطی و انتخاب CMake برای همه کاربران و کلیک بر روی دکمه Next، به مرحله بعدی مراجعه کنید، اگر می خواهید که یک فایل میانبر (shortcut) از برنامه CMake بر روی دیسکتاپ ویندوز قرار گیرد، گزینه Checkbox پایین را نیز

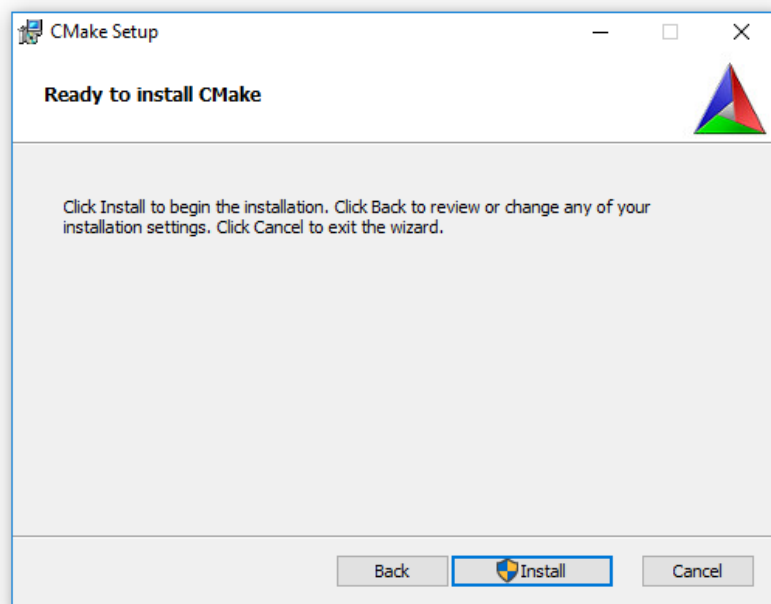
تیک بزنید، در غیر اینصورت آن را رها کنید و فایل میانبر بر روی دیسکتاپ ایجاد نخواهد شد.



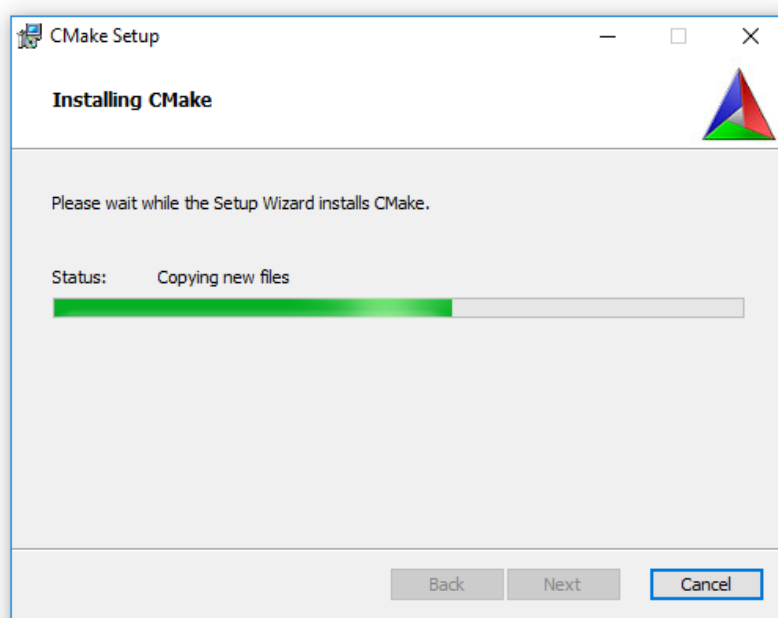
حالا با ورود به مرحله بعدی می توانید مسیر نصب برنامه را طبق آنچه که در تصویر می بیند با کلیک بر روی دکمه **Change** تغییر دهید و یا با کلیک کردن بر روی دکمه **Next** و عدم تغییر مسیر پیش فرض وارد مرحله بعدی شوید.



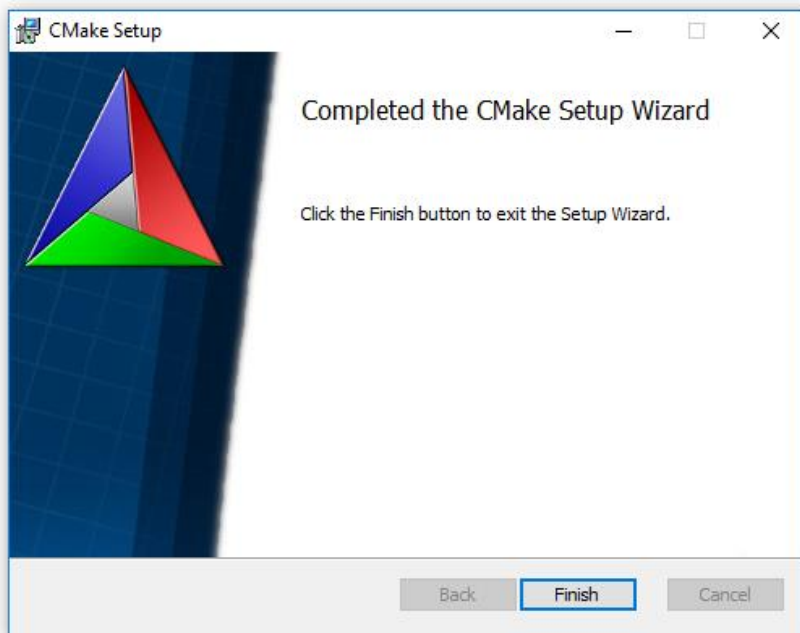
حالا می توانید بر روی دکمه Install کلیک کنید تا برنامه شروع به نصب شود.



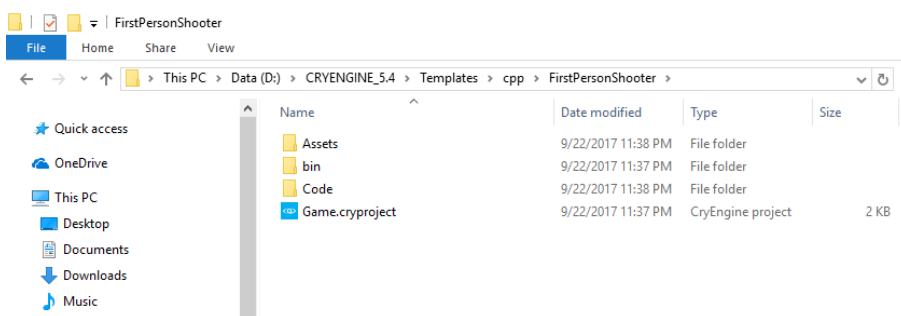
عملیات نصب ممکن است چند دقیقه طول بکشد اما زمان نصب کوتاه است و کلیه فایل های ضروری از برنامه برروی هارد کپی-پست می شود، در این فاصله می توانید یک فنجان چای یا یک لیوان آب بنوشید، خوشحال باشید که کم کم دارید شروع به برنامه نویسی با زبان C++ می کنید.



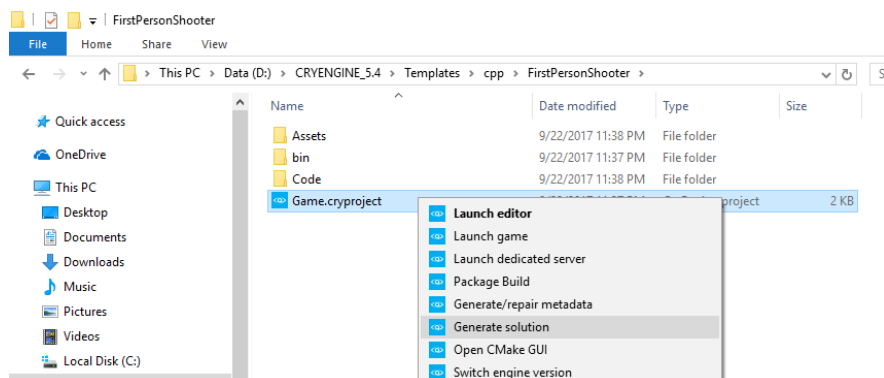
وارد مرحله پایانی شدید و پنجره زیر مانند آنچه که در تصویر می بیند را خواهید یافت، برروی دکمه Finish کلیک کنید، حالا نوبت انتخاب یکی از Template ها در پوشه CPP است.



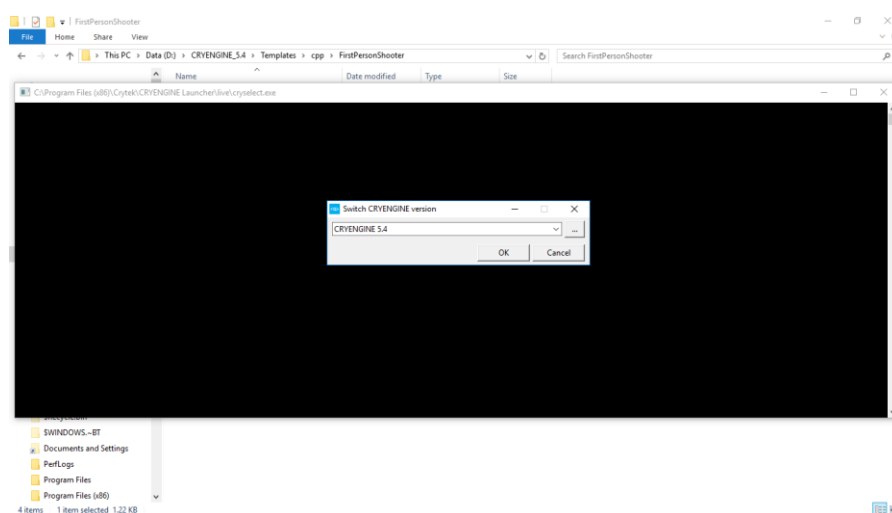
در مسیر آنچه که در تصویر زیر می بینید، در پوشه Template و زیر پوشه CPP و زیر پوشه مجدد "FirstPersonShooter" فایل پروژه ای با نام Game.cryproject وجود دارد، برای تولید فایل های C++ از این پروژه طبق وجود پوشه کد، مراحل زیر را انجام دهید



برروی فایل Game.cryproject راست کلیک کرده و در میان گزینه ها، گزینه Generate solution را انتخاب کنید

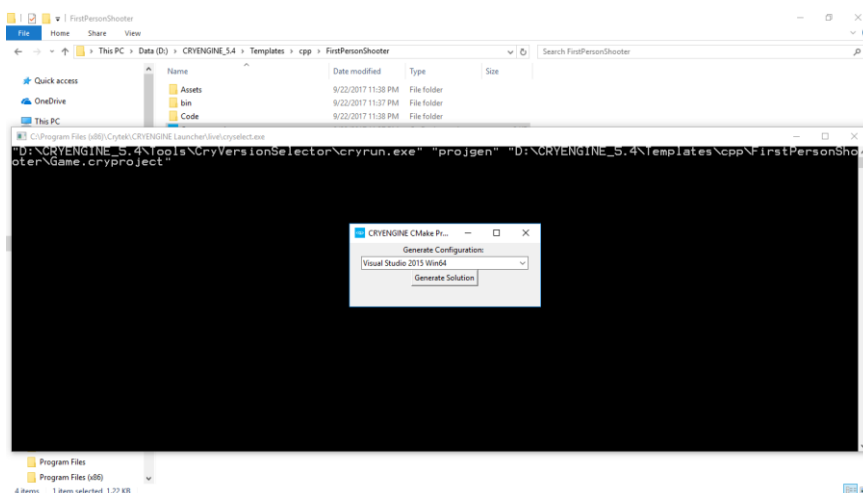


با انتخاب این گزینه، پنجره سیاه رنگی با نام `cryselect.exe` نمایش داده می شود که در وسط آن یک پنجره محاوره ای است و از شما می خواهد نسخه کرای انجین تان را انتخاب کنید و در اینجا کرای انجین ۵٫۴ را انتخاب کنید

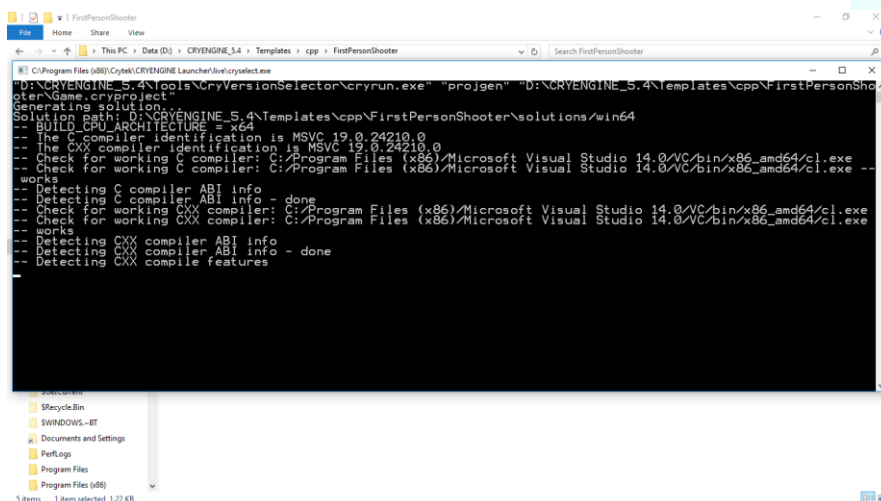


سپس با راست کلیک مجدد بر روی فایل پروژه آبی رنگ `Game.cryproject` و انتخاب گزینه `Generate solution`، پنجره ای دیگری ظاهر می شود و این بار از شما می خواهد نسخه ویژوال استودیو تان را انتخاب کنید و انتخاب شما در اینجا ویژوال استودیو ۲۰۱۵ است، ممکن است

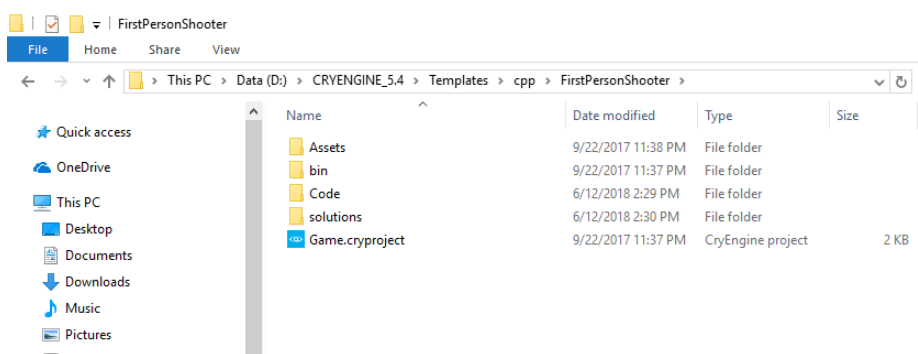
از آنجایی که کرای انجین را ثبت کرده باشید تنها نسخه ویژوال استودیو برای شما ظاهر شود، در هر حال تعجب نکنید که این مراحل انجام می شود، گام به گام رفتن این مراحل ضروری است تا شما بازی تان را با کرای انجین تولید کنید، ممکن است در این راه پراز فراز و نشیب به سیماتیک و فلوگراف و دیگر ابزارهای کدنویسی نیاز به دسترسی داشته باشید، من این راه پراز فراز و نشیب را برای شما هموار می کنم.



با انتخاب نسخه ویژوال استودیو، مراحل تولید solution و شناسایی کدها و ادغام در پروژه کمی زمان می برد، تنها برنامه نویسانی موفق می شوند که صبور باشند، برنامه نویسی زندگی شما را تحت تاثیر قرار می دهد، اگر برنامه نویسی را جزی از وجود خود بدانید.

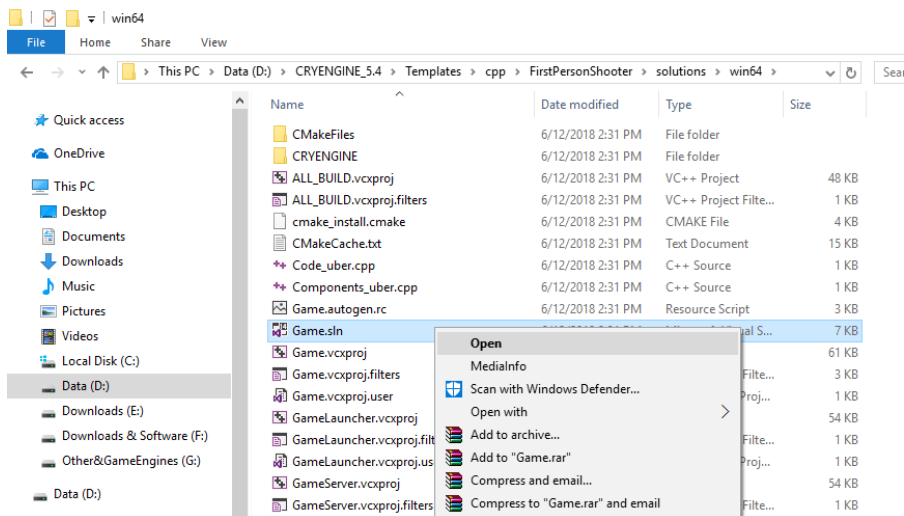


بسیار عالی، پوشه ای با نام solutions ایجاد شده است و به داخل این پوشه مراجعه کنید، این مراحل برای هر Template که در مسیر پوشه های Tutorials/cpp/Templates/ در کرای انجین است را انجام دهید، این مراحل به تولید و ایجاد یک پوشه با نام solutions در Template انتخابی شما ختم خواهد شد.

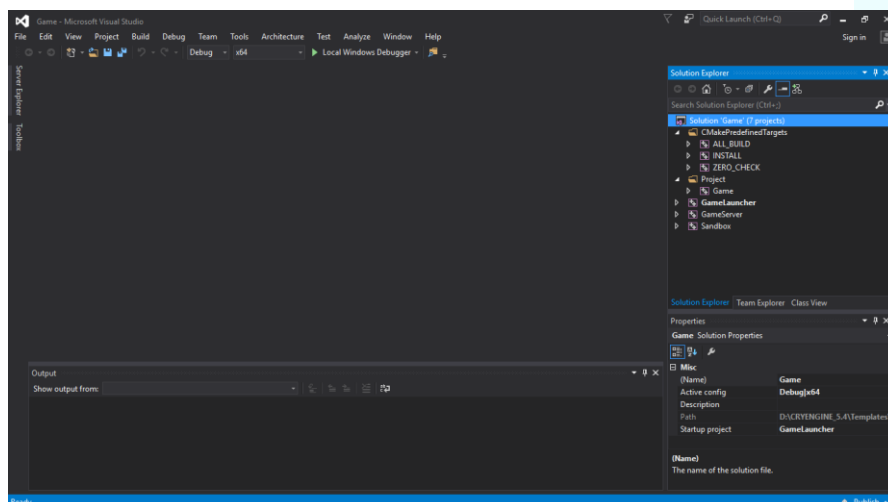


داخل پوشه solutions یک زیر پوشه دیگر با نام Win۶۴ وجود دارد و به داخل آن پوشه طبق تصویر زیر مراجعه کنید و حالا می توانید فایل Game.sln را که یک فایل solution با لیستی از کدهای C++ با تفکیک

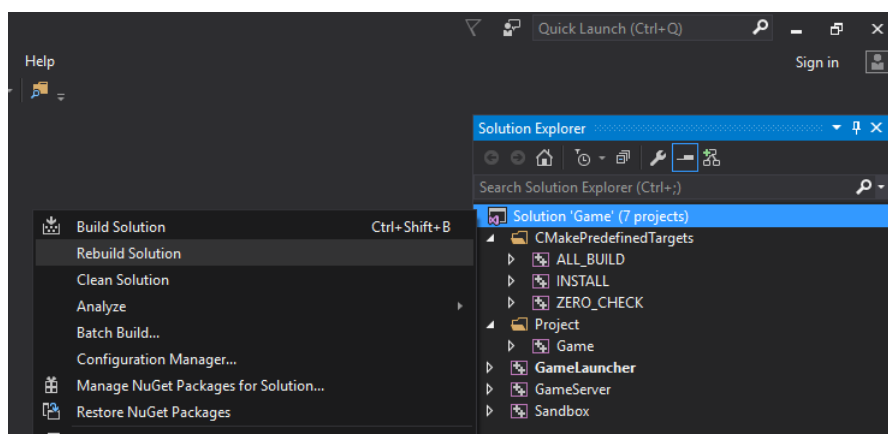
بندی CMake ببینید، بر روی فایل solution مربوطه با نام Game.sln راست کلیک کنید و گزینه open را انتخاب کنید.



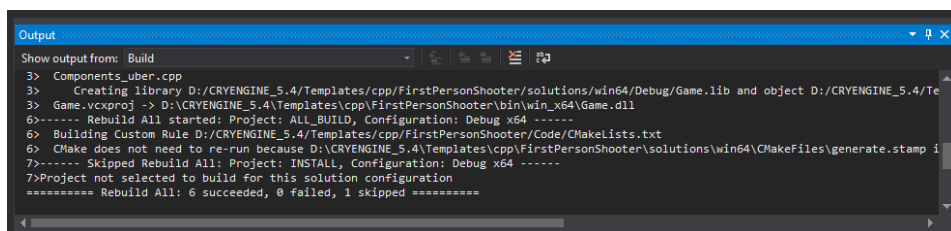
ویژوال استودیو ۲۰۱۵ با پروژه کدهای C++ باز می شود، این پروژه خود از ۷ زیر پروژه با تفکیک پذیری زیرکانه ای تشکیل شده است، شما بازی تان را می توانید از یک کامپیوتر Standalone تا تحت شبکه چند کامپیوتر Network تولید کنید و این به مهارت برنامه نویسی شما و کیت توسعه کرای انجین بستگی دارد، کرای انجین هر روز قدرتمند تر و هر روز بهینه سازی می شود، شما از ساخت بازی اول شخص شروع کنید و بگذارید بازی تان تحت شبکه نباشد، گام نخست حل مثال های ساده است و با درک مثال های کوچک زمینه حل مسائل پیچیده در دنیای بازی سازی را خواهید داشت، فراموش نکنید که زبان C++ یک زبان پویا و نسبتاً سخت است و یادگیری این زبان می تواند زمانبر یا حتی بسیار زمانبر باشد، این به مهارت و درک مطلب و هوش شما و صد البته مطالعه شما و علاقه شما بستگی دارد.



حالا زمان آن رسیده است، بازی اول شخص تان را امتحان کنید و آن را کامپایل^۱ کنید، پس طبق شکل زیر برروی ساختار درختی ریشه با نام Solution Game راست کلیک کرده و گزینه "Rebuild Solution" را انتخاب کنید تا عملیات کامپایل آغاز شود، به این عمل ساخت مجدد کل پروژه گفته می شود، با این عمل فایل های DLL پروژه باتولید می شود.



همانطور که از شکل پایین مشخص است، کل کدها و البته کل پروژه ها مجددا کامپایل شدند و ۶ پروژه با موفقیت کامپایل شده و صفر پروژه شکست خورده است و یک پروژه صرف نظر از کامپایل مجدد شده است.

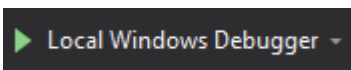


```

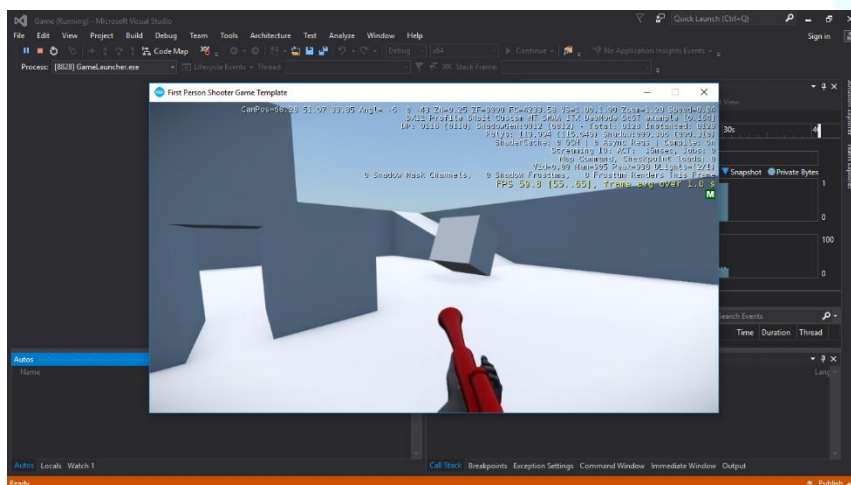
Output
Show output from: Build
3> Components_uber.cpp
3> Creating library D:\CRYENGINE_5.4\Templates\cpp\FirstPersonShooter\solutions\win64\Debug\Game.lib and object D:\CRYENGINE_5.4\Te
3> Game.vcxproj -> D:\CRYENGINE_5.4\Templates\cpp\FirstPersonShooter\bin\win_x64\Game.dll
6>----- Rebuild All started: Project: ALL_BUILD, Configuration: Debug x64 -----
6> Building Custom Rule D:\CRYENGINE_5.4\Templates\cpp\FirstPersonShooter\Code\CMakeLists.txt
6> CMake does not need to re-run because D:\CRYENGINE_5.4\Templates\cpp\FirstPersonShooter\solutions\win64\CMakeFiles\generate.stamp i
7>----- Skipped Rebuild All: Project: INSTALL, Configuration: Debug x64 -----
7>Project not selected to build for this solution configuration
===== Rebuild All: 6 succeeded, 0 failed, 1 skipped =====

```

حالا برروی دکمه زیر آنچه که در ویژوال استودیو می بینید کلیک کنید تا اجرا بازی آغاز شود



برای خارج شدن از بازی دکمه های Alt+F4 را برروی صفحه کلید فشار دهید تا از بازی خارج شوید و به محیط ویژوال استودیو برگردید، موفق شدید، شما گام نخست را برای یادگیری زبان C++ در کرای انجین برداشتید و کامپایل پروژه بازی را با موفقیت انجام دادید، به شما تبریک می گویم، چالش ها و سورپرایزهای هیجان انگیز دیگری در راه است و باید اراده تان را قوی تر کنید



توضیحات کوتاهی در رابطه با این پروژه :

کلید W : حرکت پلیر به جلو

کلید S : حرکت پلیر به عقب

کلیک سمت چپ ماوس : باعث شلیک گلوله

و حرکات ماوس باعث چرخش دوربین پلیر می شود.

عمل پرش پلیر کدنویسی نشده است و پرتاب شلیک گلوله در همه جهات امکان پذیر نیست.

نباید فراموش کنید که این پروژه گام نخست برای ساخت بازی تان در سبک اول شخص است و حتما لازم است که آنالیزی درست از کدهای C++ این پروژه داشته باشید، من به شما کمک میکنم که آنالیزهای دقیقی بر روی این پروژه انجام دهید تا بتوانید بر پایه این پروژه بازی اول شخص تان را بسازید، این کدها می توانند به شدت رشد کنند اما این تازه آغاز کار است و همت شما و انگیزه شما می تواند شما را یک برنامه نویس برتر نزد دیگران نشان دهد، فراموش نکنید که یادگیری و توسعه پروژه های بازی سازی، سال ها طول می کشد و بازی های امروزی در کلاس جهانی حاصل ده ها سال

تحقیق و پژوهش بوده است، پس روی کدهای C++ اصرار داشته باشید، شما نمی توانید راه یک ساله را در یک شب بپیمایید، آهسته و پیوسته پیش به سوی C++ Programming در تکنولوژی CryEngine، من راه را به شما نشان خواهم داد

فصل ششم

کدهای راکتوری C++ و API های

CryEngine

به قلب راکتور کرای انجین خوش آمدید، در این راکتور به جای اورانیوم، کدهای C++ شکسته و بازتولید می شوند، تمرکز ما بر روی کدهای C++ است، دلایل اینکه زبان C++ را برای کرای انجین تشریح میکنم، این است که اگر شما درک خوبی از این زبان داشته باشید، می توانید بازی های خوبی را حتی با سیماتیک و فلوگراف بسازید، پیش نیاز زبان سیماتیک و زبان فلوگراف زبان C++ است، در این کتاب اجازه دهید که پیش نیازهای سیماتیک و فلوگراف بوسیله زبان C++ رفع شود، زبان C++ زبان پیچیده ای است اما کافیسست ساختار دستورات آن را درک کنید، با درک خوب این ساختارها، پیچیدگی زبان C++ حذف خواهد شد و مباحث آن آسان تر و جذاب تر خواهد شد، قبلا اشاره کردم که موتور بازی سازی کرای انجین با زبان C++ ساخته شده است و مبحث اشاره گر ها در کرای انجین بیشترین کاربرد را برای ساخت بازی ها دارد، البته مباحث دیگری مانند پیاده سازی الگوها و لاندهای مجموعه ای دستوری نیز باعث توسعه کیت های بازی خواهد شد، یکی از مشکلات و پیچیدگی های که به زبان C++ تحمیل شده است، درک نادرست اساتید و معلمان در عدم کاربرد این زبان فوق العاده قدرتمند است، در مدارس و دانشگاه ها کاربرد این زبان به صورت ساخت بازی های ویدئویی بیشترین کمک را به دانش آموزان و دانشجویان خواهد کرد که زبان C++ را جزء دروس جذاب و هیجان انگیز بدانند، تدریس اشتباه این زبان باعث پیچیدگی دیگر شده و تحمیل این پیچیدگی ها به زبان C++ در ایران مضاعف شده است، در این کتاب سعی کردم که این پیچیدگی ها و مشکلات موجود تحمیلی از زبان C++ را حذف نموده و کرای انجین را با این زبان قدرتمند توضیح دهم، نکته ای که نباید فراموش کنید این است که در این

کتاب هیچ مبحثی از توسعه Sandbox و ایجاد پلاگین های جدید را توضیح نداده ام و از توضیحات زبان QT کاملاً صرف نظر کرده ام

مروری بر مباحث C++

همیشه خروجی دستورات زبان C++ را در محیط سیاه رنگ عقب افتاده با خطوط کدهای سفید رنگ تجربه کرده اید، در واقع این فقط آغاز است اما ادامه آن اوج هیجان را خواهد داشت، در اینجا خبری از این محیط عقب افتاده نیست، اینجا دنیای کرای انجین به کمک شما می آید تا محیط جذابی را تجربه کنید، همیشه با این عبارت و یک بدنه تابع main زبان C++ شروع می شد

```
int main ()
{
cout << "Hello , World";
return 0;
}
```

این تابع اصلی برنامه بود و با استفاده از تابع cout یک رشته (مجموعه ای از اعداد و حروف) برای ما همانند آنچه که در دو علامت کوتیشن-مارک "" بود چاپ میشد و سپس با استفاده از کلمه کلیدی return به سیستم عامل عددی مانند صفر بر میگرداند که نشان می داد که برنامه با صحت و سلامت اجرا شده و به پایان رسیده است، همانطور که قواعد ریاضی به ما کمک می کند طبق ساختار و مفهوم عملگر + عمل جمع را انجام دهیم و با عملگر - عمل تفریق را انجام دهیم، در ساختار بالا نیز باید برنامه نویسی را انجام می دادیم و با این ساختار برنامه شروع و پایانی داشت، این الگوریتم و فلوجارت ساده از برنامه در ساختار بالا به این شکل باید باشد، هر یک از علامت های { } [] , معنا دارند و برای کامپیوتر و البته کامپایلر مفهوم خاصی

دارند، مثلاً ما انسان‌ها از ساختار گرامری در یک زبان یا چند زبان باید تبعیت کنیم تا بتوانیم با انسان‌های دیگر ارتباط برقرار کنیم، اینجا نیز گرامر و ساختار زبانی C++ به ما کمک می‌کند که ما با کامپیوتر ارتباط برقرار کنیم، همانطور که کلمه خداحافظی به معنی پایان است در اینجا نیز کلمه `return` به معنی پایان برنامه یا پایان یک بلوک از کدها است، ما از جملات و دیالوگ‌های مختلف استفاده می‌کنیم و برای پایان هر جمله و شروع جمله دیگر مکثی یک یا چند ثانیه ای انجام می‌دهیم، در اینجا علامت `;` به معنی پایان جمله یا دستور است و ما انسان‌ها که آغاز و پایان دیالوگ‌ها در ارتباطات مان نسبت به همدیگر داریم، در اینجا علامت `{` و `}` برای آغاز و پایان دیالوگ برنامه با کامپیوتر یا آغاز و پایان یک گفتگو با کامپیوتر معنی دارند، پس هر چه این حروف، گرامر، علامت‌ها را در C++ بیشتر بدانید، به درک درست و ارتباط درست با C++ می‌رسید، تجربه شما و سال‌ها مطالعه بر روی این زبان به شما کمک می‌کند، درک شما و ارتباط شما با C++ قوی‌تر شود، پس ارتباط تان را با C++ قوی‌تر کنید، مکالمه فقط با عبارت خداحافظی تمام نمی‌شود، این عبارت می‌تواند هر چیزی باشد، یک مکالمه دلچسب می‌تواند به عبارت‌های مختلف و کلمات شیرین تمام شود، در اینجا `return` ۰؛ پایان یک مکالمه دلچسب با کامپیوتر است که به جای خداحافظی از عدد صفر استفاده شده است، مقدار برگشتی به تابع `main` بر می‌گردد، عدد صفر صحیح است پس خروجی تابع اصلی (`main`) باید `int` باشد (ما توابع از قبل نوشته شده دیگری مانند توابع ریاضی `sinx`، `cosx` و غیره داریم به زودی توابع فیزیک را نیز در این کتاب تشریح خواهیم کرد)، یعنی خروجی تابع `main` می‌تواند یک عدد صحیح داشته باشد، عدد - ۱ نشان می‌دهد که پایان مکالمه برنامه با کامپیوتر یا با سیستم عامل بد تمام شده است، یعنی داخل برنامه خطایی اتفاق افتاده است، این مفاهیم باید

به این صورت در دانشگاه ها تدریس می شد اما اکنون این مفاهیم را داخل این کتاب خواهید یافت، در اینجا دیگر تابع `cout` برای چاپ خروجی و تابع `cin` برای دریافت ورودی کاربرد ندارد، زیرا که کرای انجین دارای مدل های ورودی و خروجی منحصر به فردی است و دارای توابع ورودی و خروجی خود است، مثلاً برای چاپ یک رشته یا پیغام در پنجره `Output` در ویژوال استودیو و یا در پنجره `Console` در کرای انجین از تابع `CryLog` باید استفاده کنید :

```
CryLog(const char *format,...);
CryLog("Welcome to CryEngine Technology");
```

نحوه استفاده از این تابع در ساختار بالا مشخص شده و مثال آن در خروجی که می تواند نمایش داده شود `Welcome to CryEngine Technology` است.

به یاد بیاورید که دستورات `if` و `switch` چگونه کار میکردند، `if` دستوری است که هرگاه شرط داخل پرانتز درست بود مجموعه یا بلوک کد را اجرا می کند و `switch` زمانی دستورات را اجرا می کند که پارامتر (پرانتز کنار) `switch` برابر مقدار باشد

```
if(شرط)
{
مجموعه دستورات
}
```

```
switch(پارامتر)
{
```

case issue ۰۱ :

مجموعه دستورات

break;

case issue ۰۲ :

مجموعه دستورات

break;

.
.
.

case issue n :

مجموعه دستورات

break;

}

کلمات برچسب دار عددی که با نام enum ها شناخته می شود، می تواند با یک کلمه یا چند کلمه نماینده ی یک عدد باشد مثلاً

enum { lora , mohammad ,sara, ahmad, jason }

که به ترتیب lora مقدار صفر ، mohammad مقدار یک ، sara مقدار ۲ و الی آخر

متغیری که تغییر نکند و مقدار آن ثابت بماند const گفته می شود، متغیرها مانند ظرف هایی در حافظه (بیشتر منظور من حافظه RAM است) که می توانند مقادیر مختلفی را در خودنگه دارند و این مقادیر می توانند عوض شوند، مثلاً متغیر زیر را در نظر بگیرید :

int myNumber;

اسم این متغیر myNumber است مثل یه لیوان که من و شما داریم، می تواند لیوان هر اسمی داشته باشد، لیوان ها هم اندازه ها و محتوای مختلفی را می توانند در خود داشته باشند، باید قوانین نامگذاری متغیرها را اعمال کنید، مثلا نمی توانید به یک بطری لیوان بگویید، این قوانین را مطالعه کنید، حالا مثل یک لیوان که می تواند آب را در خود نگه دارد، مقادیر نیز اینگونه هستند، مثلا متغیر myNumber فقط عدد صحیح را در خود نگه می دارد، شما نمی توانید متغیر myNumber را داشته باشید و داخل آن رشته قرار دهید، مگر اینکه از تعریف زیر استفاده کنید:

```
string myNumber;
```

حالا می توانید هر عددی را در داخل این متغیر چه صحیح و چه اعشاری و یا حتی حروف و متن قرار دهید اما دیگر عمل محاسبه بر روی این متغیر امکان پذیر نیست، متغیر myNumber به لیوانی تبدیل شده که می توان آب، شیر، شربت، میخ یا حتی گچ! و هر چیزی را در خود نگه دارد. حالا تعریف زیر را در نظر بگیرید:

```
float myNumber;
```

حالا متغیر myNumber تنها می تواند اعداد اعشاری را در خود نگه دارد، شما نمی توانید حروف یا متون یا اسامی و اعداد صحیح را در آن بریزید، این متغیر مثل لیوان بزرگی است که تنها برای نگه داشتن شیرموز کاربرد دارد، شما می توانید با لیوان شیرموز چایی بنوشید؟! به مجموعه ای از کلمه های کلیدی مانند float , string , int نوع داده یا data type گفته می شود، نوع داده می تواند بیشتر باشد مثلا نوع داده enum که کلمه یا کلمات برچسب دار عددی را در خود نگه می دارد، در صفحه قبل مثالی برای enum ذکر کردم.

متغیرها می توانند گسترده تر از چیزهایی که گفتم باشند، مثل دسترسی به این متغیرها که با استفاده از کلمات کلیدی private ، public و protected امکان پذیر است، این کلمات کلیدی در مباحث کلاس و ارث

بری بیشترین معنی را به خود می گیرند اما کوتاه به توضیح این کلمات می پردازم:

private : متغیرها یا توابع خارج از کلاس توسط **entity** دیگر یا **component** دیگر و غیره قابل دسترسی نیستند و نمی توانید از این متغیرها و یا توابع خارج از کلاس استفاده کنید، تنها شما می توانید داخل کلاس جاری از این متغیرها و توابع استفاده کنید

public : متغیرها یا توابع خارج از کلاس توسط **entity** دیگر یا **component** دیگر و غیره قابل دسترسی هستند و می توانید از این متغیرها و یا توابع خارج از کلاس و همچنین داخل کلاس استفاده کنید

protected : متغیرها یا توابع خارج از کلاس توسط **entity** دیگر یا **component** دیگر و غیره قابل دسترسی نیستند و تنها توسط کلاس ارث بری شده و کلاس جاری قابل استفاده هستند

کلمه **virtual** نیز به این معنی است که تابعی که در کلاس ریشه تعریف شده و عمل ارث بری بر روی کلاس انجام شده، با این کلمه می توانید تابع را مجددا تعریف کنید که عملکرد تابع متفاوت از عملکرد تابع در کلاس ریشه (کلاس پدر) باشد، تابع چه در کلاس ریشه چه در کلاس فرزند هنگامی **virtual** می شود باید هم نام باشند

ساختار (**struct**) ساختمان داده ای است که برای ساخت نوع داده جدید کاربرد دارد، مثلاً ساختار زیر نوع داده زمان را برای متغیرها می سازد :

```
struct Time {
    int hour;
    int minute;
    int second;
}
```

حالا می توانید متغیر با نام **myTime** را مثل زیر تعریف کنید، ساختارها در کرای انجین بسیار کاربرد دارند :

```
Time myTime;
```

فراموش نکنید که struct ها هم می توانند مفهوم ارث بری را پیاده کنند

برای شروع بازی در پروژه جزیره ربات ها (Robot Island) بر روی DVD در ضمیمه این کتاب ، نقش تابع main توسط دو سورس کد صفحه با فایل های GamePlugin.cpp و GamePlugin.h را می توان دید،نقطه شروع بازی در رویداد سیستمی در کلاس CGamePlugin اتفاق می افتد،با استفاده از دستور switch و پارامتر event باید مشخص نمود که اگر شما می خواهید ریجیستر کنید تمامی کامبونت ها را در سیماتیک کدهای زیر با استفاده از نوع داده enum به نام ESYSTEM_EVENT_REGISTER_SCHEMATYC_ENV اجرا می شود اما شروع بازی از کدهای زیر با استفاده از نوع داده enum با نام ESYSTEM_EVENT_GAME_POST_INIT اجرا می شود،اینجا نقش تابع main را در case ی به نام ESYSTEM_EVENT_GAME_POST_INIT را می توان مشاهده کرد،دقت کنید که این رویه تنها یکبار اجرا می شود،یعنی تعداد اجرا یکبار اتفاق می افتد

If (!gEnv->IsEditor())

دستور بالا تاکید می کند که اگر در Sandbox کرای انجین نبودید بلوک (مجموعه ای) از دستورات را در بین { و } اجرا شود،این دستور تایید می کند که اگر بازی تان در Visual Studio یا در فایل GameLauncher.exe بود ، بلوک (مجموعه ای) از دستورات را در بین { و } اجرا شود،پس طبق دستور if بالا نمی توانید کدهای داخل بلوک را در ادیتور کرای انجین در خطوط بعدی را اجرا کنید اما این کدها چی هستند؟ قبل از ادامه دادن به این بحث،میخوام با کدهای سبز رنگ آشنا شوید، این

دستورات مجموعه توضیحات یا توضیحات تک خط با رنگ سبز نمایش داده می شود، یعنی این توضیحات برای برنامه نویس یا برنامه نویسان مفید است و در خروجی فایل exe حذف خواهد شد و در بازی اجرا نخواهد شد، پس اگر می توانید توضیحات را به کار ببرید تا دیگر برنامه نویسان نیز متوجه کد شما شوند، دستوراتی که به رنگ سبز توضیح تبدیل می شوند را می توان بین علامت // برای توضیحات تک خط و علامت های /* */ برای توضیحات چند خط استفاده می شود، به این کد نگاه کنید

```
void CGamePlugin::OnSystemEvent(EsystemEvent
event, UINT_PTR wparam, UINT_PTR lparam)
{
switch (event)
{
case ESYSTEM_EVENT_REGISTER_SCHEMATYC_ENV:
{
// Register all components that belong to this
plug-in
auto staticAutoRegisterLambda =
[] (Schematyc::IenvRegistrar& registrar)
{
// Call all static callback registered with
the CRY_STATIC_AUTO_REGISTER_WITH_PARAM
Detail::CStaticAutoRegistrar<Schematyc::IenvRe
gistrar&>::InvokeStaticCallbacks(registrar);
};

if (gEnv->pSchematyc)
{
```

```

gEnv->pSchematyc-
>GetEnvRegistry().RegisterPackage(

    stl::make_unique<Schematyc::CenvPackage>(
        CgamePlugin::GetCID(),
        "EntityComponents",
        "Crytek GmbH",
        "Components",

        staticAutoRegisterLambda
        )
    );
}
}
break;
// Called when the game framework has
// initialized and we are ready for game logic to
// start
case ESYSTEM_EVENT_GAME_POST_INIT:
{
    // Don't need to load the map in editor
    if (!gEnv->IsEditor())
    {
        gEnv->pConsole->ExecuteString("map
example.۱", false, true);
        gEnv->pConsole->ExecuteString("e_TimeOfDay
۱,۲۳");
        //gEnv->pConsole-
>ExecuteString("r_Fullscreen ۱");
        //gEnv->pConsole->ExecuteString("r_width
۱۲۸۰");
    }
}

```



```

//gEnv->pConsole->ExecuteString("r_height
۷۲۰.");
    gEnv->pConsole-
>ExecuteString("sys_DeactivateConsole ۱");
    gEnv->pConsole-
>ExecuteString("r_DisplayInfo .");
    //gEnv->pConsole->ExecuteString("quit");

    CryLog("gEnv->pConsole-
>ExecuteString(e_TimeOfDay ۱,۲۳) 😊
GamePlugin");

    // gEnv->pHardwareMouse-
    >IncrementCounter();

    /*
    CryAudio::ControlId const MyTriggerId =
    CryAudio::StringToId("MusicLevel.۱");
    gEnv->pAudioSystem-
    >LoadTrigger(MyTriggerId);
    gEnv->pAudioSystem-
    >ExecuteTrigger(MyTriggerId);
    */
    }
}
break;
}
}

```

توضیحات مربوط به مجموعه ای از دستورات در بلوک if در case ی با مقدار

: `ESYSTEM_EVENT_GAME_POST_INIT`

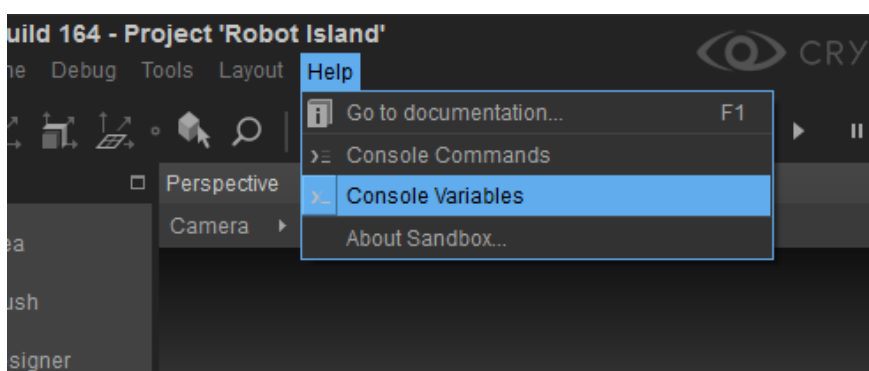
تابع `ExecuteString` برای اجرا دستورات قابل تغییر کنسولی با نام

`Console Variable` است و تعداد این دستورات بسیار زیاد است که

در اینجا چند مورد را با استفاده از این تابع بررسی می کنیم و همچنین برای

دسترسی به همه `Console Variable` ها به منوی `Help` و به

گزینه `Console Variables` مراجعه کنید



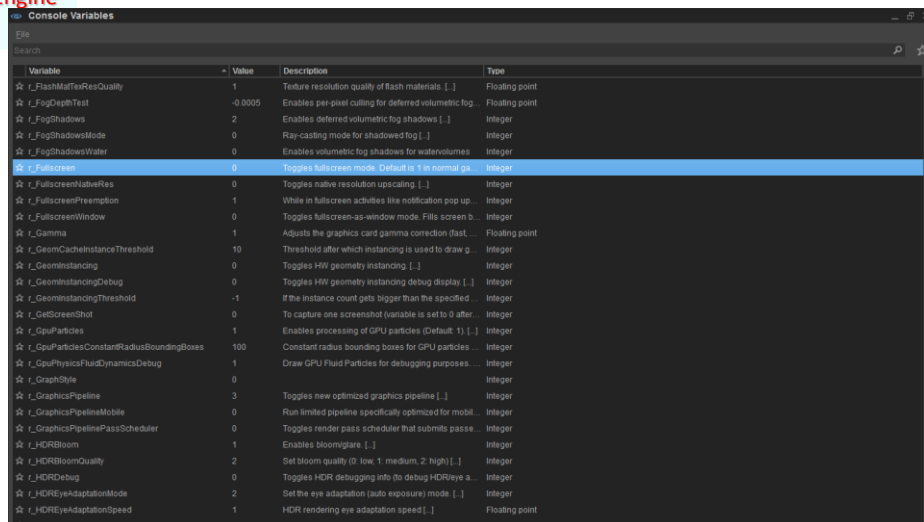
در تصویر بعدی می بینید که تعداد دستورات `Console Variables`

ها بسیار زیاد است و هر کدام از این دستورات می تواند عمل مهمی را در

بازی انجام دهد، ساده ترین مثال ها می تواند تغییر زمان شب و روز، فعال یا

غیرفعال سازی مه، تمام صفحه کردن پنجره بازی و غیره، پیشنهاد می کنم که

تک تک این دستورات را نگاه کنید و همه دستورات را بررسی کنید.



بررسی اینکه کدها در ادیتور اجرا می شود یا نه؟ این دستور مقدار true زمانی که در ادیتور بازی اجرا می شود را برمیگرداند و زمانی که در ادیتور بازی اجرا نشود (در فایل GameLauncher.exe یا در ویژوال استودیو بازی اجرا شود) مقدار false را برمیگرداند

`gEnv->IsEditor()`

با این دستور مرحله (map) مربوطه با نام example۰۱ را لود می شود

`gEnv->pConsole->ExecuteString("map example۰۱",
false, true);`

زمان شبانه روزی بازی (e_TimeOfDay) به ساعت ۱:۲۳ نصف شب تغییر می کند

`gEnv->pConsole->ExecuteString("e_TimeOfDay
۱,۲۳");`

برای تمام صفحه کردن بازی (r_Fullscrren) استفاده می شود که صفر برای انصراف از این عمل و برای فعال سازی از یک استفاده می شود.

```
gEnv->pConsole->ExecuteString("r_Fullscreen  
۱");
```

عرض صفحه نمایش به ۱۲۸۰ پیکسل تغییر می کند، اینجا تغییر resolution اتفاق می افتد

```
gEnv->pConsole->ExecuteString("r_width ۱۲۸۰");
```

ارتفاع صفحه نمایش به ۷۲۰ پیکسل تغییر می کند، اینجا تغییر resolution اتفاق می افتد

```
gEnv->pConsole->ExecuteString("r_height ۷۲۰");
```

در بازی نیز می توان به متغیرهای کنسولی با فشار دکمه ~ دسترسی داشت که در اینجا با دادن عدد یک به متغیر کنسولی **sys_DeactivateConsole** کنسول در بازی غیر فعال می شود و با دادن عدد صفر دوباره به این متغیر کنسول در بازی دوباره فعال خواهد شد

```
gEnv->pConsole->ExecuteString("sys_DeactivateConsole ۱");
```

مجموعه اطلاعاتی متنی در بازی همیشه نشان داده می شود، مثلاً از بین همه این اطلاعات نمایش داده شد، میزان سرعت اجرای بازی (frame rate) نیز نشان داده می شود، با دادن عدد صفر به متغیر کنسولی **r_DisplayInfo**، نمایش اطلاعات متنی به پایان می رسد.

```
CamPos=24.49 28.85 35.19 Angl= 44 0 -110 ZN=0.05 ZF=8000 FC=0.52 VS=1.00,1.00 Zoom=1.00 Speed=0.00
DX11 Profile 64bit MedSpec SMAA ITX DevMode (Editor) StGT example01 [0.164]
DP: 0094 (0094) ShadowGen:0005 (0005) - Total: 0099 Instanced: 0099
Polys: 019,041 (019,435) Shadow:001,090 (001,079)
ShaderCache: 0 GCM | 0 Async Reqs | Compile: On
Streaming IO: ACT: 38msec, Jobs: 0
Vid=0.00 Mem=1253 Peak=1316 DLights=(3/1)
1 Shadow Mask Channels, 0 Shadow Frustums, 0 Frustum Renders This Frame
FPS 356.7 [308..399], frame avg over 1.0 s
M
```

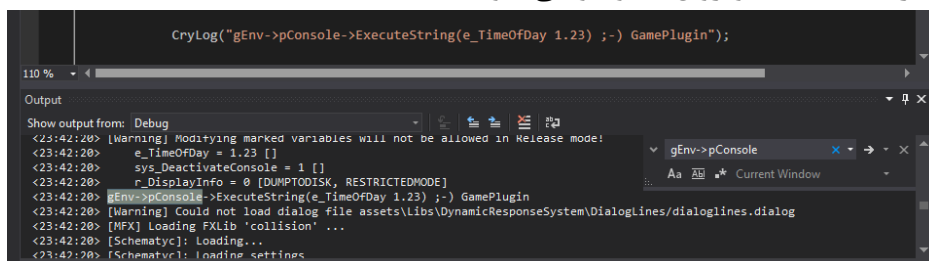
```
gEnv->pConsole->ExecuteString("r_DisplayInfo
.");
```

برای خروج از بازی از پارامتر رشته ای quit استفاده کنید

```
gEnv->pSystem->Quit();
gEnv->pConsole->ExecuteString("quit");
```

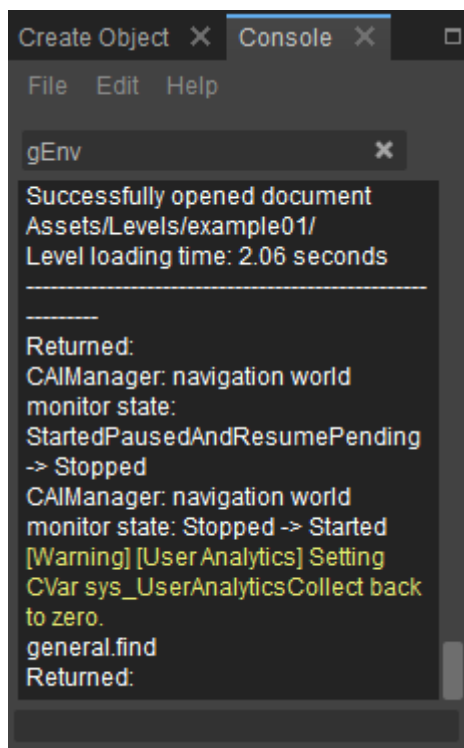
```
CryLog("gEnv->pConsole-
>ExecuteString(e_TimeOfDay ۱,۲۳) 😊
GamePlugin");
```

در تصویر نیز می بینید که CryLog باعث نمایش دادن پیغام یا متنی در پنجره Output و ویژوال استودیو می شود



همچنین برای نمایش دادن پیغام ها یا متن چاپی در هنگام اجرای بازی در ادیتور می توان از تابع CryLog استفاده کرد، مطمئناً این دستور برای دیباگ کردن کدها به صورت سریع کاربرد دارد و می توان با استفاده از تابع

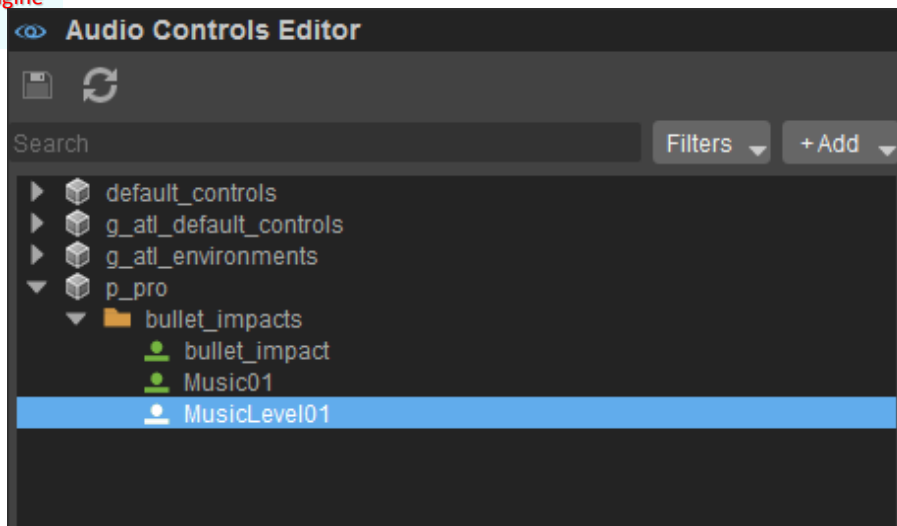
CryLog مشخص کرد که آیا بلوک دستورات اجرا می شود یا نه؟ مثلاً آیا بلوک `if` یا `switch` اجرا می شود یا نه؟



با استفاده از این دستور ماوس کرای انجین در بازی یا حتی در ادیتور فعال می شود

`gEnv->pHardwareMouse->IncrementCounter();`

از آنجایی که تریگرهای صوتی را برای بازی تان ساخته اید، باید قادر باشید که به وسیله کدهای C++ این تریگرها را بخش کنید، من قبلاً یک تریگر صوتی با نام `MusicLevel01` ساخته ام، این کار بسیار ساده است

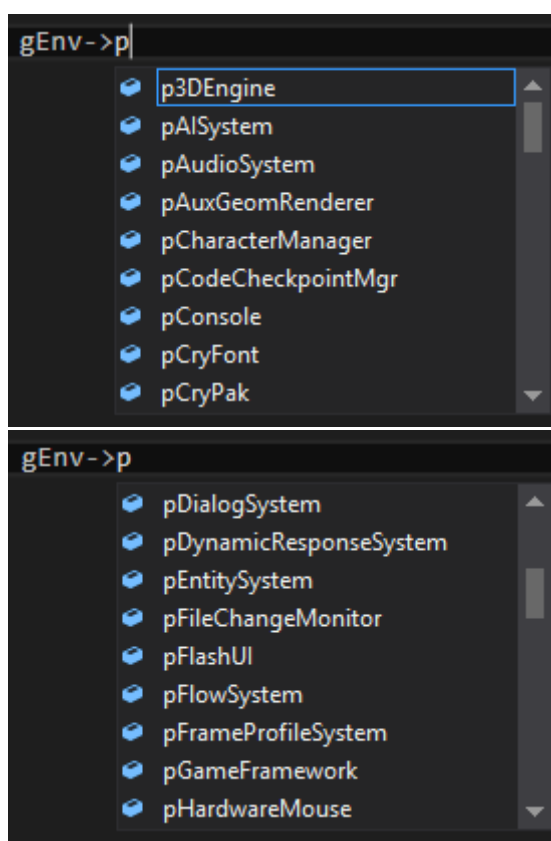


با استفاده از این کدها ابتدا شما یک **Id** با استفاده از کلاس **CryAudio** برای تریگر صوتی تان می سازید و سپس با استفاده از سیستم صوتی کرای انجین، آن تریگر را لود یا همان بارگذاری می کنید و در آخر تریگر مربوطه را به حالت اجرا (**Execute**) در می آورید، در اینجا **Id** همان دستگیره ای است که باعث دسترسی به تریگرهای مختلف در پنجره **Audio Controls Editor** می شود

```
CryAudio::ControlId    const    MyTriggerId    =
CryAudio::StringToId("MusicLevel۰۱");
gEnv->pAudioSystem->
LoadTrigger(MyTriggerId);
gEnv->pAudioSystem->
ExecuteTrigger(MyTriggerId);
```

API های کرای انجین

هر گاه شما به دستور `GetEntity` یا `m_pEntity` برخورد کردید، منظور شی جاری است که کدها بر روی آن اعمال خواهد شد، با استفاده از کلمه دستوری `gEnv` به حجم عظیمی از تمامی کتابخانه های کرای انجین می توانید دسترسی داشته باشید :



gEnv->p|

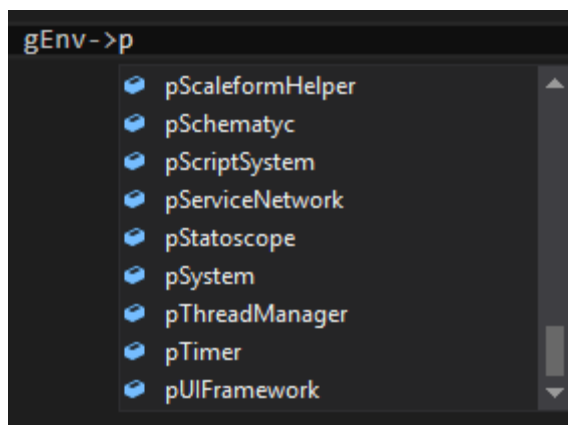
- pi
- pInput
- pJobManager
- pLiveCreateHost
- pLiveCreateManager
- pLobby
- pLocalMemoryUsage
- pLog
- pMaterialEffects

gEnv->p

- pMonoRuntime
- pMovieSystem
- pNameTable
- pNetContext
- pNetwork
- pOpticsManager
- pOverloadSceneManager
- pParticleManager
- pPhysicalWorld

gEnv->p|

- pProfileLogSystem
- pProtectedFunctions
- pRemoteCommandManager
- pRenderer
- pScaleformHelper
- pSchematyc
- pScriptSystem
- pServiceNetwork
- pStatoscope



من به بررسی تعدادی از این کتابخانه ها و توابع آن می پردازم، درک تعدادی از این توابع و کتابخانه ها از روی نام آنها مشخص است و تعدادی دیگر نیز کاملاً دشوار است و هنوز اسناد آموزشی برای آن از طرف کرایتک منتشر نشده است.

سرعت اجرای زمان در بازی با استفاده از دستور زیر است، این دستور باعث می شود که ۱۰ برابر سریع تر ثانیه ها در بازی اجرا شود، یعنی عمل FastMotion اتفاق می افتد

```
gEnv->pTimer->SetTimeScale(۱۰,۰);
```

با این دستور زمان ۱۰ برابر آهسته تر اجرا می شود و به این عمل SlowMotion گفته می شود

```
gEnv->pTimer->SetTimeScale(۰,۱);
```

با دستور زیر زمان به صورت عادی (Normal) اجرا می شود

```
gEnv->pTimer->SetTimeScale(۱,۰);
```

و اگر بخواهیم زمان متوقف شود از دستور زیر استفاده کنید:

```
gEnv->pTimer->SetTimeScale(۰,۰);
```

نمایش کنسول دستورات و پیغام های کنسولی در بازی با کلمه کلید true و برای مخفی شدن کنسول در بازی از کلمه کلیدی false استفاده کنید

```
gEnv->pConsole->ShowConsole(true);
```

با استفاده از مفهوم اشاره گر (*) یک متغیر با نام ppe از نوع داده IParticleEffect تعریف کنید و سپس با استفاده از اشاره گر ریشه به تمامی دستورات انجین (gEnv) می توانید اقدام به پیدا کردن یک پارتيكل سیستم در پنجره Assets Browser و لود کردن پارتيكل سیستم در Entity جاری نماید، البته با استفاده از GetEntity() نیز می توانید به تابع LoadParticleEmitter دسترسی داشته باشید:

```
IParticleEffect *ppe = gEnv->pParticleManager->FindEffect("fire.pfx");
m_pEntity->LoadParticleEmitter(۰, ppe);
```

نکته بسیار مهم و حیاتی این است که عدد ۰ در اینجا برای slot۰ است و شما نمی توانید همزمان از slot۰ برای لود مدل هندسی و لود پارتيكل استفاده کنید، زیرا که هر slot تنها یک عمل لود را انجام می دهد، مثلا اگر می خواهید همزمان هم لود پارتيكل و هم لود مدل هندسی را انجام دهید باید دو slot در نظر گرفته شود، slot۰ برای لود مدل هندسی و slot۱ برای لود پارتيكل سیستم، به مثال زیر دقت کنید :

```
m_pEntity->LoadGeometry(۰ ,
"MyModels/Grenade/grenade۰۱.cgf");
```

```
IParticleEffect *ppe = gEnv->pParticleManager-
>FindEffect("fire.pfx");
m_pEntity-
>LoadParticleEmitter(۱, ppe);
```

عدد ۰ به slot۰ و عدد ۱ به slot۱ اشاره می کند، همزمان عمل لود مدل سه بعدی (یک نارنجک) و لود پارتيكل با نام fire.pfx انجام می شود و هر دو در شی جاری نمایش داده می شوند

```
pParticleEntity->FreeSlot(۰);
```

آزادسازی slot۰ با مقدار صفر انجام میشود

```
pParticleEntity->FreeSlot(۱);
```

آزادسازی slot۱ با مقدار یک انجام میشود

تغییر نور خورشید به رنگ قرمز

```
gEnv->p3DEngine->SetSunColor(Vec3(۲۵۵,۰,۰));
```

نمایش Ahmad Karami بر روی صفحه نمایش بازی با رنگ قرمز و موقعیت $x=۱۰۰$ و $y=۵۰$ با اندازه فونت ۳ ، دقت کنید که دستور ColorF(R,G,B) با پارامترهای R یعنی Red و G یعنی Green و B یعنی Blue است و در اینجا مقدار پارامتر قرمز فعال شده است یعنی عدد ۱ به آن داده شده است، شما می توانید به هر سه پارامتر در بازه صفر تا یک ، عدد اعشاری نیز اختصاص دهید، دقت کنید که برای دیدن نوشته باید در داخل رویدادی به مانند Update بازی باشد، به این معنا که دائما این دستور اجرا شود تا عمل نمایش رشته بر روی صفحه نمایش اتفاق بیفتد، این

دستور با تعداد اجرای $O(1)$ است، یعنی تنها یک بار این دستور اجرا می شود

```
gEnv->pAuxGeomRenderer->Draw2dLabel(۱۰۰, ۵۰, ۳,
ColorF(۱, ۰, ۰), false, "Ahmad Karami");
```

رسم یک کره سبز رنگ با شعاع یک متر و در فضای سه بعدی بازی با
 $x=۶۰$ و $y=۶۱$ و $z=۳۳$

```
gEnv->pAuxGeomRenderer-
>DrawSphere(Vec3(۶۰,۶۱,۳۳), ۱, ColorF(۰,۱,۰));
```

این دستور عمل حذف شی جاری در بازی را انجام می دهد ، اگر دقت کنید دستور `GeEntityId()` به `Id` شی فعلی `GetEntity` اشاره دارد و با استفاده از دستور `RemoveEntiy` از کتابخانه `pEntitySystem` با اشاره گر ریشه به تمامی دستورات انجین (`gEnv`) این عمل اتفاق می افتد

```
gEnv->pEntitySystem-
>RemoveEntity(GetEntityId());
```

تغییر جاذبه از مقدار پیش فرض عدد ۱۳- به عدد ۱-

```
gEnv->pConsole-
>ExecuteString("p_gravity_z -۱");
```

فعال سازی فیزیک در یک شی

```
GetEntity()->EnablePhysics(true);
```

غیرفعال کردن فیزیک در یک شی

```
GetEntity()->EnablePhysics(false);
```

مخفی کردن یک شی با کانال شفافیت (Alpha)

```
GetEntity()->SetOpacity(a);
```

مقدار a می تواند عددی اعشاری بین ۰ تا ۲۵۵ باشد

```
GetEntity()->SetViewDistRatio(d);
```

مقدار d می تواند عددی برحسب متر در بازه ۰ تا ۲۵۵ برای نمایش یا عدم نمایش شی باشد، اگر مسافت پلیر از شی بیشتر از d باشد، شی دیده نمی شود و اگر مسافت پلیر از شی کمتر از d باشد شی دیده می شود

```
GetEntity()->SetPos(Vec3(۰, ۱, ۲));
```

موقعیت را به مختصات $x=۰$, $y=۱$, $z=۲$ تغییر خواهد داد

```
GetEntity()->SetName("Grenade");
```

تعیین اسم شی جاری

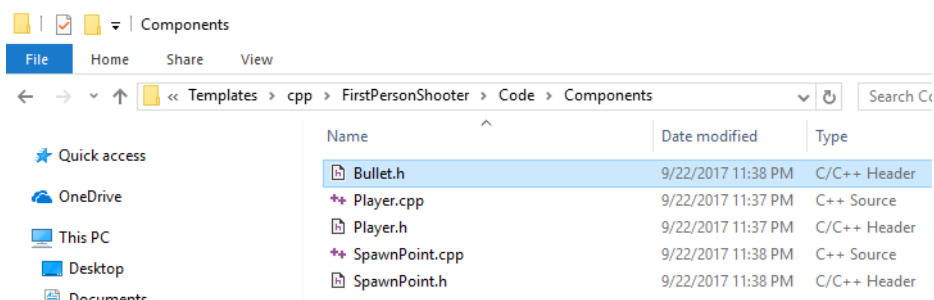
```
m_pEntity->SetScale(Vec3(۲, ۲, ۲));
```

تعیین اندازه (مقیاس) شی جاری

ساختارهای جدید (New Structures)

برای ساخت بازی ها با این موتور ،با استفاده از کلاس ها،ارث بری،چند ریختی با ساختارهای جدید برخورد می کنیم که در هیچ کتابی منتشر نشده است که سعی کرده ام بیشتر این مباحث را پوشش دهم،یکی از این ساختارها مباحث مربوط به ساخت Entity ها و Component ها است،برای ساخت هر Entity و استفاده در بازی (نه در زمان طراحی بازی یا نه در داخل ادیتور کرای انجین) نیاز به حداقل یک هدر فایل (*.h) است،مانند هدر فایل bullet.h در تمپلت FirstPersonShooter در تصویر زیر می بینید و در

فصل بعدی این هدر فایل بررسی می شود

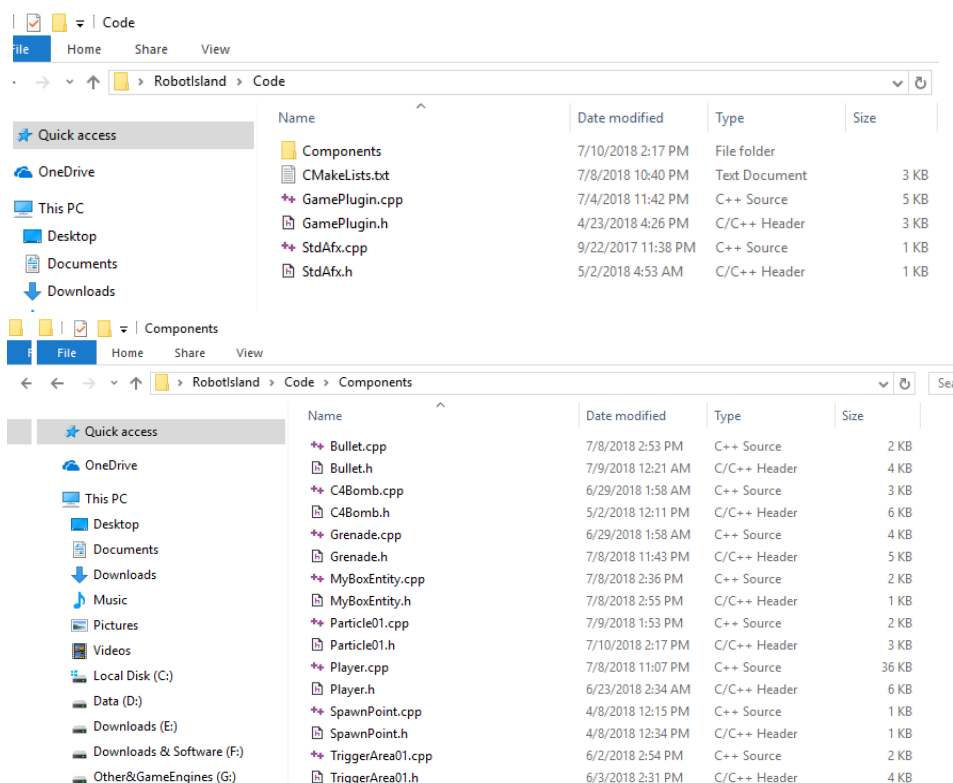


بر خلاف ساخت یک component یا entity مانند آنچه که در بالا به آن اشاره شد که فقط در بازی استفاده می شود و توسط فقط یک هدر فایل ساخته می شود و در طراحی مرحله نمی توان به آن دسترسی داشت یعنی در پنجره Create Object در بخش component ها لیست بندی نشده است(راه حل اول)، یک راه حل دیگر وجود دارد که بتوان به component یا entity داخل مرحله برای طراحی مرحله و در داخل ادیتور کرای انجین به آن دسترسی داشت یعنی در پنجره Create Object در بخش component ها لیست بندی شده است که برای ساخت و ثبت اینگونه Entity یا component در زمان طراحی مرحله (استفاده در داخل ادیتور

و داخل بازی) به یک هدر فایل (*.h) و یک سی پی پی فایل (*.cpp) نیازمند هستید (در این فصل و فصل بعدی مفصل به راه حل دوم پرداخته ام)، به مثال های زیر توجه کنید :

ایجاد یک کامپونت خالی در Schematyc و Prefab ایجاد شده در ادیتور (فقط برای شروع کافی است، این مثال را درک کنید)

ابتدا دو فایل Header و CPP ایجاد می کنید، می توانید برروی دیسکتاپ راست کلیک کرده و برروی گزینه New و سپس انتخاب notepad فایل متنی را ایجاد کنید، پسوند فایل متنی *.txt است و شما پسوند های آن دو فایل متنی *.txt را به *.cpp و *.h تغییر دهید و سپس به مسیر RobotIsland\Code\Components کپی کنید.



داخل فایل Header با نام MyBoxEntity.h کدهای زیر را تایپ کنید :

```
#pragma once
```

```
#include <CryEntitySystem/IEntityComponent.h>
```

```
#include <CryEntitySystem/IEntity.h>
```

```
class CAK۰۱ final : public IEntityComponent
{
```

```
public:
```

```
CAK۰۱() = default;
```

```
virtual ~CAK۰۱() {}
```

```
static void
```

```
ReflectType(Schematyc::CTypeDesc<CAK۰۱>& desc)
```

```
{
```

```
desc.SetGUID("{A۰CB۲۰E۹-۹۴BD-۴۷F۴-AB۴۱-  
E۰۱C۷A۲EDC۴F}"_cry_guid);
```

```
desc.SetEditorCategory("Ahmad.Karami");
```

```
desc.SetLabel("SpawnPoint.AhmadKarami۰۱");
```

```
desc.SetDescription("This spawn  
point can be used to spawn entities , Ahmad  
Karami ha ha ha ;-");
```

```
desc.SetComponentFlags({
    IEntityComponent::EFlags::Transform,
    IEntityComponent::EFlags::Socket,
    IEntityComponent::EFlags::Attach });
}
```

```
public:
void SpawnEntityAK...(IEntity*
otherEntity);
};
```

داخل فایل Cpp با نام MyBoxEntity.CPP کدهای زیر را تایپ کنید :

```
#include "StdAfx.h"
#include " MyBoxEntity.h"

#include <CrySchematyc/Reflection/TypeDesc.h>
#include <CrySchematyc/Utils/EnumFlags.h>
#include <CrySchematyc/Env/IEnvRegistry.h>
#include <CrySchematyc/Env/IEnvRegistrar.h>
#include
<CrySchematyc/Env/Elements/EnvComponent.h>
#include
<CrySchematyc/Env/Elements/EnvFunction.h>
#include
<CrySchematyc/Env/Elements/EnvSignal.h>
#include <CrySchematyc/ResourceTypes.h>
#include <CrySchematyc/MathTypes.h>
#include <CrySchematyc/Utils/SharedString.h>
```

```
static void
RegisterSpawnPointComponentAhmadKarami::\ (Schema
tyc::IEnvRegistrar& registrar)
{
Schematyc::CEnvRegistrationScope scope =
registrar.Scope(IEntity::GetEntityScopeGUID())
;
{
Schematyc::CEnvRegistrationScope
componentScope =
scope.Register(SCHEMATYC_MAKE_ENV_COMPONENT(CA
K::\));
// Functions
{
}
}
}
```

```
CRY_STATIC_AUTO_REGISTER_FUNCTION(&RegisterSpa
wnPointComponentAhmadKarami::\)
```

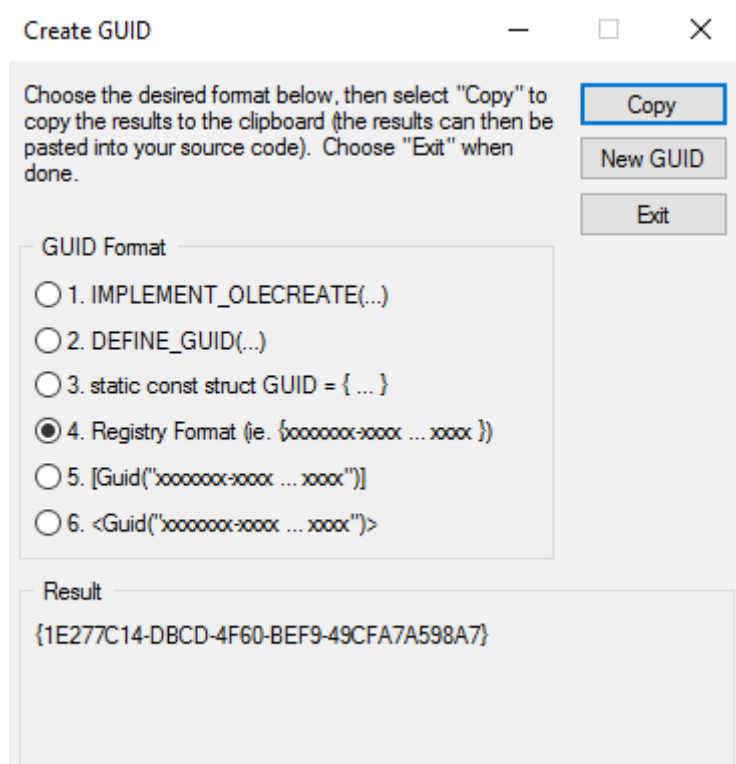
```
void CAK::\::SpawnEntityAK::\ (IEntity*
otherEntity)
{
otherEntity->SetWorldTM(m_pEntity-
>GetWorldTM());
}
```

می بینید که در فایل MyBoxEntity.h خط زیر وجود دارد که باید guid جدید به این فایل هدری داشته شود، بعداً این خط را توضیح خواهم داد، به

منوی Tools در ویژوال استودیو مراجعه کرده و گزینه Create GUID را انتخاب کنید

```
desc.SetGUID("{A۰CB۲۰E۹-۹۴BD-۴۷F۴-AB۴۱-  
E۰۱C۷A۲EDC۴F}"_cry_guid);
```

پنجره زیر ظاهر شده :

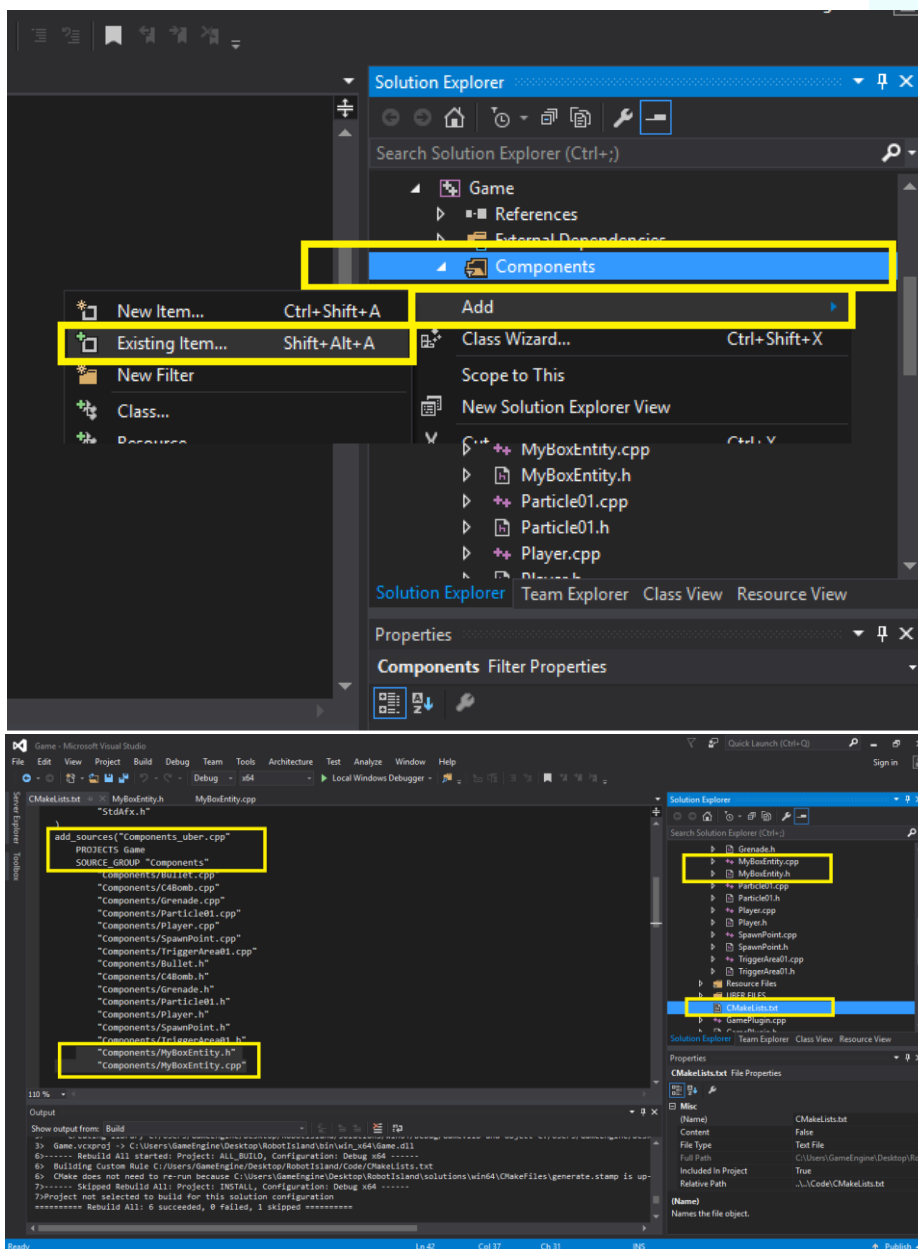


باید گزینه Registry Format را که گزینه چهارم است را انتخاب کنید و سپس برروی دکمه New GUID کلیک کنید و سپس دکمه Copy را انتخاب کنید تا guid جدیدی را کپی کرده و به خط زیر عوض کنید :

```
desc.SetGUID("{۷۱۵۰۸۸B۸-F۹B۲-۴۲۸۲-A۳A۹-  
۴۸۰۶E۹D۵۷۵۶A}"_cry_guid);
```

دقت کنید که هر **Entity** که می سازید شامل یک فایل **header** با پسوند **h** و یک فایل **cpp** با پسوند **cpp** است و هر **Entity** باید **guid** منحصر به فرد را داشته باشد، که هر بار می خواهید یک **Entity** جدید ایجاد کنید، باید یک **guid** جدید طبق روش بالا باید عمل کنید.

باید فایل های **MyBoxEntity.h** و **MyBoxEntity.cpp** را به فایل پروژه اضافه کنید و سپس اسم این دو فایل را مطابق شکل زیر به فایل **CMakeLists.txt** اضافه کنید، با توجه به کادرهای زرد و قرمز عمل کنید و در آخر به منوی **Build** مراجعه کرده و گزینه **Rebuild Solution** را انتخاب کنید



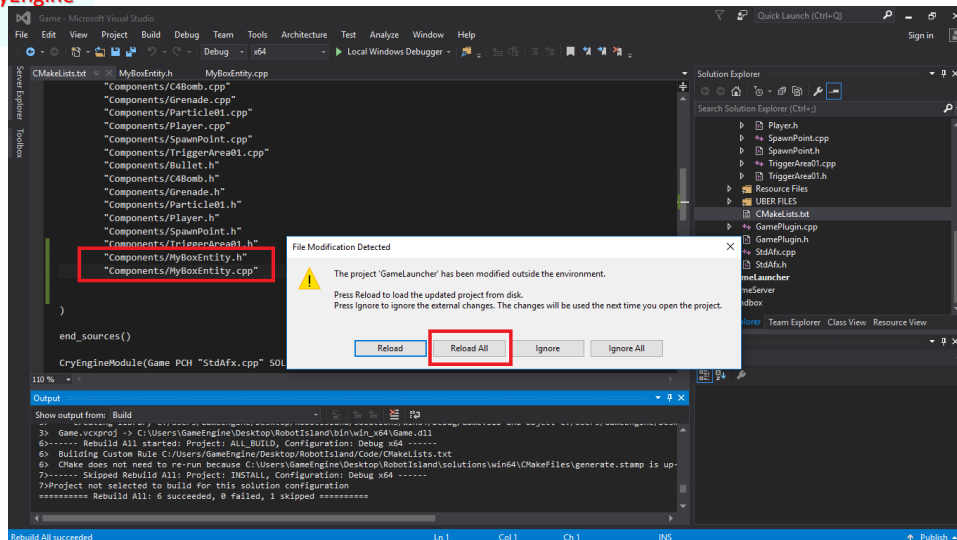
بعد از کامپایل کل پروژه پیغام هشدار در رابطه با CMakeLists.txt ظاهر می شود و شما باید دکمه Reload All را انتخاب کنید

فصل ۶

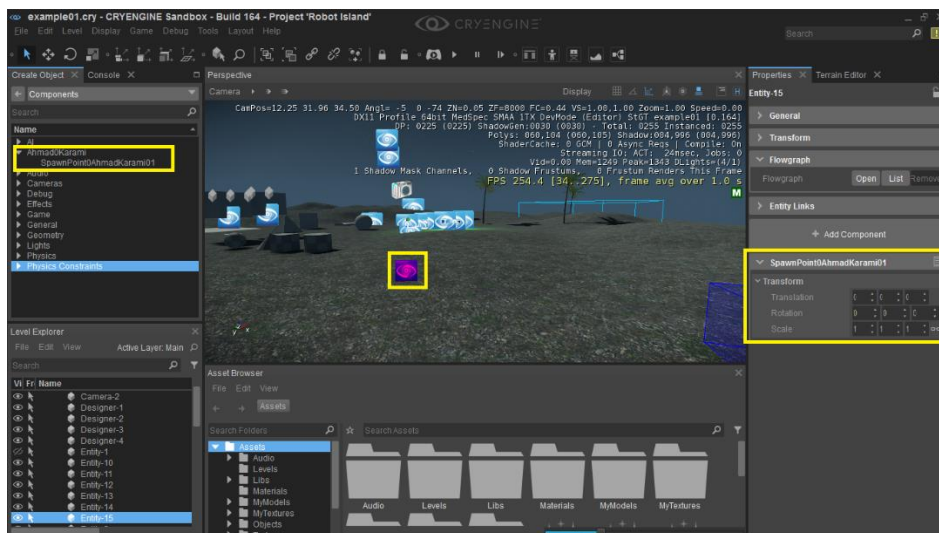
کد راکتوری C++ و

Api های

CryEngine



حالا پنجره کرای انجین را باز کنید و باید تصویر زیر را ببینید، به کادر های زرد رنگ نگاه کنید :



چه اتفاقی افتاد؟ اینجا فقط یک Entity خالی را مشاهده می کنید! درسته، این گام اول برای ساخت انواع Entity ها است، کافیه که تحلیل

کدهای مربوط به فایل MyBoxEntity.h و MyBoxEntity.cpp را درک کنید :

من ابتدا از فایل MyBoxEntity.h شروع می کنم و خطوط آن را توضیح میدهم :

#pragma once

برای جلوگیری از تعریف مجدد کلاس ها و برای جلوگیری از تعریف مجدد هدر فایل ها با پیش پردازنده include استفاده می شود، توصیه می شود برای ایجاد entity یا component در کرای انجین از این دستور استفاده کنید و فقط یکبار تعریف هدر فایل و فقط یکبار تعریف کلاس با این دستور انجام می شود

```
#include <CryEntitySystem/IEntityComponent.h>
#include <CryEntitySystem/IEntity.h>
```

دستوراتی که با # شروع می شود، دستورات پیش پردازنده گویند و هر کلمه از دستورات پیش پردازنده عملکرد خاص خود را دارد، یکی از این کلمات include است و وقتی که از این کلمه استفاده می کنیم باعث می شود کتابخانه هایی به پروژه مان متصل کنیم و از این کتابخانه ها استفاده کنیم، مثلاً برای استفاده از توابع Exp , Cosx , Sinx و غیره باید از کتابخانه Math.h استفاده کنیم این کتابخانه کلیه توابع و معادلات ریاضی را دارد، پس باید با استفاده از دستور #include Math.h از کتابخانه ریاضی استفاده کنیم، این دو خط include در ارتباط با استفاده از توابع و دستوراتی برای ساخت Component ها و Entity های مختلف است.


```
class CAK۰۱ final : public IEntityComponent
{
};
```

این خط در ارتباط با ساخت کلاسی با نام CAK۰۱ است (برای ساخت هر کلاس اسم اول آن با C بزرگ شروع می شود و حرف اول نام کلاس نیز بزرگ است) و با استفاده از مفهوم وراثت کلاس CAK۰۱ از واسطه (Interface) IEntityComponent ارث بری می کند و در نهایت آخرین کلاس در زمینه پیاده سازی و ارث بری خواهد بود، داخل کلاس نیز با استفاده از کلمات کلیدی زیر می توان توابع و متغیرها را از لحاظ دسترسی کنترل نمود :

public (سراسری) : هر متغیر یا تابعی که در این سطح تعریف شود، کلاس های دیگر می توانند به آنها دسترسی داشته باشند و توابع توسط کلاس های دیگر قابل دستکاری و قابل تغییر است

protected (حفاظت شده) : هر متغیر یا تابعی که در این سطح تعریف شود، کلاس هایی که ارث بری خواهند داشت می توانند به آنها دسترسی داشته باشند و البته داخل کلاس نیز متغیرها و توابع قابل دسترس خواهند بود

private (خصوصی) : هر متغیر یا تابعی که در این سطح تعریف شود، کلاس های دیگر نمی توانند به آنها دسترسی داشته باشند ، متغیرها و توابع تنها داخل کلاس قابل استفاده هستند

final (نهایی) : این آخرین کلاسی است که طبق **virtual / override** مشتق می شود و کلاسی دیگر نمی تواند از این کلاس که کلمه **final** استفاده شده ارث بری کند

```
public:
CAK۰۱() = default;
virtual ~CAK۰۱() {}
```

از آنجایی که نوع دسترسی به کلاس سراسری بوده و استفاده از مقدار پیش فرض در هر نوع داده ای برای کلاس و شی کاملاً امکان پذیر است، یک تخریبگر نیز به صورت مجازی برای کلاس تعریف شده است و بعد از تخریب کلاس، مثلاً وقتی که Entity حذف شد، بتواند عملی انجام دهد اما با توجه به اینکه در خط پایانی علامت {} وجود دارد یعنی بعد از تخریب کلاس (منظور شی ساخته شده)، عملی انجام نشود چون داخل {} هیچ دستوری وجود ندارد.

```
static void
ReflectType(Schematyc::CTypeDesc<CAK۰۱>& desc)
{
}
}
```

تابع سیستمی ReflectType به صورت ایستا (static) تعریف شده است و از روی نام آن مشخص است که عمل انعکاس را طبق داخل بلوک کد انجام می دهد، این انعکاس می تواند در Sandbox و در Schematyc اتفاق بیفتد، به این معنا که component و یا entity بر اساس کد ساخته شود و از آنجایی که باید کلاس را انتخاب کنید CTypeDesc نام کلاس برای ثبت در Sandbox و Schematyc کلاس CAK۰۱ است و اما داخل این تابع خطوط زیر وجود دارند :

```
desc.SetGUID("{A۰CB۲۰E۹-۹۴BD-۴۷F۴-AB۴۱-  
E۰۱C۷A۲EDC۴F}"_cry_guid);
```

در دفتر ثبت ویندوز (Registry Editor) هر شی (Entity) در بازی باید ثبت شود و هر شی در بازی باید کد منحصر به فردی داشته باشد که به این کد منحصر به فرد **guid** گفته می شود، این خط در واقع برای ثبت **entity** در کلاس ۰۱ CAK است.

```
desc.SetEditorCategory("Ahmad.Karami");
```

شما یک کاتالوگ با نام **Ahmad.Karami** برای Entity هایتان یا **component** یتان می سازید و در پنجره **Create Object** در بخش **Components** آن را می توانید مشاهده کنید

```
desc.SetLabel("SpawnPoint.AhmadKarami.۱");
```

اسم Entity یا Component شما در **Sandbox** و یا در **Schematyc** با نام **SpawnPoint.AhmadKarami.۱** خواهد بود

```
desc.SetDescription("This spawn  
point can be used to spawn entities , Ahmad  
Karami ha ha ha ;-)");
```

شما می توانید توضیحاتی را به Entity تان و یا Component تان اختصاص دهید، این فقط برای راهنمایی است، می توانید آن را به صورت یک رشته خالی رها کنید اما این کار توصیه نمی شود و معمولاً توضیحاتی برای **entity** یا **component** برای طراحان مرحله و بازی سازها نوشته می شود

```
desc.SetComponentFlags({
    IEntityComponent::Eflags::Transform,
    IEntityComponent::Eflags::Socket,
    IEntityComponent::Eflags::Attach });
```

اینجا پرچم هایی برای Entity یا Component است مثل سه خصیصه موقعیت، چرخش و مقیاس که با نام Transform رایج است، اینجا پرچم های دیگری مثل Attach و Socket نیز وجود دارند اما به جزئیات آن در عملکرد توسط کرایتک اشاره نشده است

```
public:
void SpawnEntityAK01(IEntity*
otherEntity);
```

تابعی غیر سیستمی تعریف شده است که نحوه تکثیر Entity یا قرارگیری Entity در فضای سه بعدی را اعلان می کند، پیاده سازی این تابع و دیگر توابع در فایل cpp انجام می شود و حالا نوبت به توضیح در رابطه با فایل MyBoxEntity.cpp است :

```
#include "StdAfx.h"
#include "MyBoxEntity.h"
```

فایل هدر ریشه با نام StdAfx.h در تمامی فایل های cpp وجود دارد، بعداً در رابطه با این فایل توضیح خواهم داد که عملکرد آن چیست و متغیرهای سراسری مشترک و توابع سراسری مشترک در این هدر فایل تعریف می شوند.

از آنجایی که فایل هدر `MyBoxEntity.h` را ایجاد و کدنویسی کردید، برای استفاده از کلیه متغیرها، توابع و ویژگی ها برای درست کردن `entity` و `component` از این فایل هدر، آن را باید در فایل `MyBoxEntity.cpp` نیز درج کنید و برای استفاده از کلیه متغیرها، توابع و ویژگی های هدر فایل سیستمی یا غیر سیستمی (توسط برنامه نویس نوشته شده) باید از دستور پیش پردازنده با کلمه `#include` استفاده نمایید، در ادامه هدر فایل های دیگری نیز هستند که در ادامه این دو خط آمده است :

```
#include <CrySchematyc/Reflection/TypeDesc.h>
#include <CrySchematyc/Utils/EnumFlags.h>
#include <CrySchematyc/Env/IEnvRegistry.h>
#include <CrySchematyc/Env/IEnvRegistrar.h>
#include
<CrySchematyc/Env/Elements/EnvComponent.h>
#include
<CrySchematyc/Env/Elements/EnvFunction.h>
#include
<CrySchematyc/Env/Elements/EnvSignal.h>
#include <CrySchematyc/ResourceTypes.h>
#include <CrySchematyc/MathTypes.h>
#include <CrySchematyc/Utils/SharedString.h>
```

بخشی از این هدر فایل ها هم شامل کتابخانه های مختلف برای ثبت `entity` و `component` در سیماتیک و سندباکس است، بخشی از این طیف از کتابخانه ها در هر هدر فایل و سی پی پی فایلی باید موجود باشد تا هنگام بازتولید `solution` توسط `visual studio` و `cmake` خطاهای کامپایلری در عدم دسترسی به منابع بازتولید شده رخ ندهد، زیرا که بعدا اگر میخواهید

این کدها را منتشر کنید، نباید خطاهای بازتولید شده از عدم دسترسی به هدر فایل ها رخ دهند.

```
static void
RegisterSpawnPointComponentAhmadKarami۰۱(Schema
tyc::IEnvRegistrar& registrar)
{

}
```

برای ثبت **component** در محیط سیماتیک و سندباکس به صورت ثبت کننده محیطی باید از این تابع ایستا استفاده نمود، این عمل در ارتباط با موازی کاری فایل هدر تعریف شده بالا با نام **MyBoxEntity.h** است، داخل این تابع خطوط زیر وجود دارند :

```
Schematyc::CEnvRegistrationScope scope =
registrar.Scope(IEntity::GetEntityScopeGUID())
;
{
Schematyc::CEnvRegistrationScope
componentScope =
scope.Register(SCHEMATYC_MAKE_ENV_COMPONENT(CA
K۰۱));
// Functions
{
}
}
```

اینجا محدوده ای با نام **scope** وجود دارد که تبدیلات زبان ماشین یا همان صفر و یک انجام می شود با استفاده از **guid** باید عمل ثبت کننده را انجام

دهیم تا در دفتر ویندوز یا همان ریجستری ویندوز انجام شود، این عمل در ادامه با ثبت `component` و `entity` در سندباکس و در نهایت برای محیط سیماتیک نیز انجام می شود و شما با استفاده از پیش پردازنده ماکرویی `#define` یک کامپوننت جدید در سیماتیک (`SCHEMATYC_MAKE_ENV_COMPONENT`) با نام کلاس `CAK۰۱` ایجاد می کنید و بدنه آن بدون تابع است (احتمالا این بخش در رابطه با ایجاد نودهای مختلف در سیماتیک است)

```
CRY_STATIC_AUTO_REGISTER_FUNCTION(&RegisterSpawnPointComponentAhmadKarami۰۱)
```

این تابع توسط ماکرو پیش پردازنده عمل ثبت `entity` و `component` را با موفقیت انجام می دهد (البته اگر کدها را درست تایپ کرده باشید)

اگر به یاد بیاورید یک تابع غیرسیستمی در هدر فایل `MyBoxEntity.h` اعلان شده بود و حالا باید آن را توسط خطوط زیر پیاده سازی کنید

```
void CAK۰۱::SpawnEntityAK۰۱(IEntity*
otherEntity)
{
    otherEntity->SetWorldTM(m_pEntity-
>GetWorldTM());
}
```

تابع `SpawnEntityAK۰۱` در کلاس `CAK۰۱` است و هر زمانی که `Entity` یا `Component` را به داخل صحنه بازی `Drag & Drop` (بکشیم و رها) کنیم و از آنجایی که توسط پارامتر

otherEntity از نوع اشاره گر ساختار **IEntity** موقعیت فعلی شی را محاسبه می کند و در نهایت موقعیت شی در فضای سه بعدی **GetWorldTM** گرفته می شود (توسط اشاره گر داخلی کلاس **m_pEntity**) و سپس در فضای سه بعدی قرارگیری خواهد شد اجازه دهید این **Entity** را بیشتر توسعه دهیم و مفاهیم بسیار مهمی از ساختارهای جدید را برای شما تشریح کنم اما قبل توسعه، دستوراتی وجود دارند که باید این دستورات را تجزیه و تحلیل کنیم و در فصل بعدی به این مهم خواهیم پرداخت

ایجاد Console Variable ها

یکی از مهارت های دیگری که شما باید به آن مسلط باشید، ساخت انواع **Console Variable** است، اما **Console Variable** چیست؟ **Console Variable** یا متغیرهای کنسولی همانطور که از روی اسم آن نیز پیداست، اسامی هستند که در کنسول کرای انجین در بخش سندباکس یا در بازی فراخوانی می شوند، مثلاً شما وقتی می خواهید زمان روز را تغییر دهید، به متغیر کنسولی **e_TimeOfDay** مراجعه کرده و مقابل این متغیر در کنسول عددی به آن اختصاص می دهید که زمان روز یا زمان شب است مثلاً عدد ۲۳ که ساعت زمانی شب در بازی است، در ابتدای این فصل من توضیحات خوبی برای استفاده از متغیرهای کنسولی ارائه دادم و حالا نوبت به آن رسیده که با استفاده از زبان شیرین و قدرتمند **C++** متغیرهای کنسولی را ایجاد کنیم، به مثال زیر دقت کنید، در صفحات قبلی توضیح دادم که کلیه کدهای فایل ***.cpp** و هدر فایل ها ***.h** باید در داخل پوشه **Component** در مسیر زیر باشند

CRYENGINE_۵,۴\Templates\cpp\FirstPersonShooter\
Code\Components

یا در مسیر زیر باشند

RobotIsland\Code\Components

همچنین نحوه اضافه کردن کلیه کدهای فایل *.cpp و هدر فایل ها *.h به ویژوال استودیو و اضافه کردن به cmakeLists.txt و build کردن آن را آموختید، از این به بعد این مراحل توضیح داده شده را انجام دهید و از ذکر مجدد آن پرهیز میکنم

ابتدا فایل MyCVars.h را توضیح می دهیم

```
#pragma once
class CAhmadKaramiCVars
{
```

یک کلاس را با نام CAhmadKaramiCVars تعریف می کنم که وظیفه ی آن ایجاد یک متغیر کنسولی است

با توجه به بخش public یک سازنده با نام کلاس CAhmadKaramiCVars ایجاد می کنیم، حتی اگر سازنده ها نیز هیچ عملی را انجام ندهند شما باید سازنده را تعریف کنید

```
public:
CAhmadKaramiCVars() {}
```

و سپس یک تخریب کننده با نام کلاس CAhmadKaramiCVars ایجاد می کنیم، قانون تعریف سازنده حتی که هیچ عملی را انجام ندهد برای تخریب کننده نیز کاملاً صادق است، اگرچه در تخریب کننده تابع UnregisterCVars تعریف شده است و عمل پاک کردن متغیر

کنسولی ثبت شده را انجام می دهد، این مسئله باعث می شود که مصرف بهینه حافظه و مدیریت حافظه برای ساخت بازی در نظر گرفته شود

```
~CAhmadKaramiCVars() { UnregisterCVars(); }
```

حالا در بخش **public** باید دو تابع تعریف شود :

RegisterCVars : این تابع عمل ثبت متغیر کنسولی را در بازی انجام می دهد

UnregisterCVars : این تابع عمل پاک کردن متغیر کنسولی را در بازی انجام می دهد

```
void RegisterCVars();
void UnregisterCVars();
};
```

حالا نوبت به فایل **MyCVars۰۱.cpp** می رسد

```
#include "MyCVars۰۱.h"
```

از آنجایی که یک هدر فایل با کلاسی از متغیر کنسولی تعریف کرده ام باید از این کلاس متغیر کنسولی استفاده کنم و هدر فایل آن را با نام **MyCVars۰۱.h** تعریف می کنم

```
#include <CrySystem/ISystem.h>
#include <CrySystem/IConsole.h>
```

برای ساخت متغیرهای کنسولی و استفاده از این متغیرها باید از دو هدر فایل `IConsole.h` و `ISystem.h` استفاده شود

در داخل تابع `RegisterCVars` عمل ثبت یک متغیر کنسولی انجام می شود اما توابع سیستمی مختلفی برای ثبت انواع متغیرهای کنسولی وجود دارد مانند :

`RegisterString` : یک متغیر کنسولی بر اساس آرگومان رشته ای ایجاد می شود که محتوای آن رشته خواهد بود

`RegisterFloat` : یک متغیر کنسولی بر اساس آرگومان رشته ای ایجاد می شود که محتوای آن عدد اعشاری خواهد بود

`RegisterInt` : یک متغیر کنسولی بر اساس آرگومان رشته ای ایجاد می شود که محتوای آن عدد صحیح ۳۲ بیتی خواهد بود

`RegisterInt64` : یک متغیر کنسولی بر اساس آرگومان رشته ای ایجاد می شود که محتوای آن عدد صحیح ۶۴ بیتی خواهد بود

در اینجا من از تابع سیستمی `RegisterString` در ایجاد متغیر کنسولی استفاده می کنم

```
void CAhmadKaramiCVars::RegisterCVars()
{
```

```
    ConsoleRegistrationHelper::RegisterString("Mes
sageAhmadKarami","I Like CryEngine C++
Programmer",VF_RESTRICTEDMODE,"this is a test
for send float value...");
```

با توجه به تابع سیستمی `RegisterString` یک متغیر کنسولی براساس آرگومان های زیر ثبت می شود:

`MessageAhmadKarami` : نام متغیر کنسولی

```

I Like CryEngine C++ Programmer : محتوای متغیر کنسولی
VF_RESTRICTEDMODE : محدودیت برای پرچم متغیر تعریف شده
this is a test for send string value... : شرحی
برای متغیر کنسولی ایجاد شده است که برای برنامه نویسان مفید خواهد بود
}

```

در داخل تابع UnregisterCVars عمل پاک کردن یک متغیر کنسولی انجام می شود

```

void CAhmadKaramiCVars::UnregisterCVars()
{

IConsole* pConsole=gEnv->pConsole;
pConsole->UnregisterVariable("MessageAhmadKarami",true)
;

```

با توجه به تابع سیستمی UnregisterVariable متغیر کنسولی با نام MessageAhmadKarami پاک می شود

```

}

```

برای استفاده و به کار بردن متغیر کنسولی تعریف شده باید هدر فایل `MyCVars.۰.۱.h` را در `GamePlugin.h` تعریف کنیم

```
#include "Components/MyCVars.۰.۱.h"
```

```
CAhmadKaramiCVars* m;
```

باید یک متغیر کلاسی اشاره گر با نام `m` را بر اساس نوع داده کلاسی `CAhmadKaramiCVars` را تعریف کنیم

در نهایت فایل `GamePlugin.cpp` با توجه به دستور `switch` در `case` مربوط به `ESYSTEM_EVENT_GAME_POST_INIT` بلوک `if` زیر را تعریف می کنیم

```
if(!m)
{
    m=new CAhmadKaramiCVars();
}
```

با توجه به متغیر `m` یک کلاس با نام `CAhmadKaramiCVars` را ایجاد می کنیم تا شی متغیر کنسول ایجاد شود

```
m->RegisterCVars();
```


فصل هفتم

ساخت Entity های جدید و

Component های جدید

Time و deltaTime چیست؟

زمان بعد چهارم در کرای انجین است، شما باید توانایی کنترل زمان را در بازی داشته باشید و هر Entity که ساخته می شود و زمان در آن نقش داشته باشد باید بتوانید نحوه استفاده از زمان را در آن بدانید و فریم به فریم اجرای زمان را مدیریت کنید، اما زمان جریانی از اجرای فریم ها است و از برش های بسیار کوچک تشکیل شده است که با نام delta از آن یاد می شود، شما قبلا در حساب دیفرانسیل و انتگرال با مفهوم delta آشنا شدید و این مفهوم هم در اینجا صادق است و در اینجا مجموعه ای از برش های زمانی باهم جمع می شوند و تشکیل ثانیه ها را خواهد داد، واحد زمان ثانیه است و در مثال زیر برای دریافت deltaTime از دستور زیر استفاده می شود، یک متغیر اشاره گری با نام pCtx از نوع داده ساختار (struct) SEntityUpdateContext تعریف شده است، شما می توانید اسم دیگری به این متغیر دهید مثلا deltaTime اما در صفحات اینترنت و فروم های آموزشی شرکت کرایتک اسم pCtx برگزیده شده است، اجازه دهید همین اسم برای متغیر باشد، در اینجا event از نوع پارامتری و از نوع داده ساختار (struct) SEntityEvent است و در اینجا کنترل زمان بر اساس پارامتر [۰] nParam که شامل deltaTime بوده و مقدار دهی به ساختار SEntityUpdateContext را انجام می دهد

```
SEntityUpdateContext* pCtx =
(SEntityUpdateContext*)event.nParam[۰];
```

از آنجایی که متغیر pCtx از نوع اشاره گر تعریف شده و عمل cast بر روی متغیر نیز انجام شده است، با استفاده از دستور زیر به برش زمانی یا همان deltaTime دسترسی خواهید داشت

اما این دستور چه کاربردی دارد؟ از آنجایی که اجرای یک بازی شامل واحد زمان است و زمان نقش مهمی در پیکره بازی را برعهده دارد، قطعا اشیاء بازی بیشترشان به زمان احتیاج دارند، مثلا یک جعبه وقتی عمل چرخش را انجام می دهد، عمل چرخش با زمان گره خورده است، پس برش های زمانی یا همان **deltaTime** باید داخل یک تابع که عمل چرخش جعبه را انجام می دهد باید استفاده شود ، دستور زیر در داخل **header** فایل تعریف شده است، مثل دستور زیر :

```
OnUpdate(pCtx->fFrameTime);
```

در اینجا یک تابع با نام **OnUpdate** وجود دارد که برش های زمانی را به صورت آرگومان **pCtx->fFrameTime** به آن تابع ارسال می شود، فراموش نکنید که زمان مفهومی انتگرالی است و نه مفهوم سیگما در بازی ، در ادامه ما چرا ، باید تابع تعریف شده بالا را در داخل کلاسی با نام **MyBox** در **cpp** فایل تعریف کنید و برای عمل چرخش دستورات زیر را با استفاده از برش زمانی در نظر بگیرید :

```
void MyBox::OnUpdate(float deltaTime)
{
    GetEntity()->SetRotation(GetEntity()-
    >GetRotation()*Quat::CreateRotationZ(deltaTime
    *۲));
}
```

همانطور که می بینید، تابع **OnUpdate** مقدار پارامتری **deltaTime** با نوع داده **float** یا همان اعشاری را دریافت می کند و با توجه به اینکه اشاره گر **GetEntity** به شی بازی فعلی اشاره دارد و با استفاده از تابع

SetRotation عمل چرخش شی جاری **Set** یا همان مقدار دهی می کند و با توجه به پارامتر **deltaTime** و گرفتن زاویه چرخش شی فعلی با ضرب زاویه محور **Z** ایجاد می شود، مقدار زاویه فعلی با **GetRotation** در شی فعلی گرفته شده و در زاویه ایجاد شده طبق محور **Z** به تابع **SetRotation** ارسال می شود، اگر دقت کنید می بینید که هر بار که زاویه عوض می شود، زمان در ایجاد این چرخش نقش کلیدی را ایفا می کند، ابتدا شی فعلی زاویه فعلی محورهای **Z** و **Y** و **X** را بدست می آورد و سپس زاویه محور **Z** طبق یک برش زمانی با سرعت ۲ برابر ضرب می شود تا عمل تولید زاویه طبق محور **Z** انجام شود.

رویداد (رخداد) چیست و چگونه رویدادهای مختلف را ایجاد کنیم؟

به منشاء انجام عملیات خاص که باعث اجرای مجموعه ای از دستورات می شود، مثلاً وقتی که بازی اجرا می شود، وقتی که بازی به پایان می رسد، وقتی بازی در حال اجرا است، وقتی پلیمر وارد محدوده ای می شود، وقتی پلیمر از محدوده ای خارج می شود و غیره

رویدادها باعث اجرای مجموعه ای از دستورات می شوند، اما نحوه در دست گرفتن این رویدادها و نحوه اجرای این رویدادها نیز بسیار مهم است، مثلاً رویدادی که وقتی بازی اجرا می شود، یک بار اتفاق می افتد یا رویدادی که از بازی خارج می شوید یکبار اتفاق می افتد، وقتی که بازی در حال اجرا است، این رویداد می تواند به تعداد بسیار زیاد اجرا شود، وقتی پلیمر وارد محدوده ای شد می تواند رویداد یکبار یا دوبار یا چندین بار اجرا شود و غیره، تعریف رویدادها با توابع مکملی همراه است، به این معنا که از توابع **GetEventMask** و **ProcessEvent** استفاده کرده، زیرا این دو تابع به شدت به هم وابسته هستند، پس توابعی که به شدت به هم وابسته باشند، به نام توابع مکمل یعنی تکمیل کننده همدیگر نام برده می شود، اینجا

دو تابع مکمل `ProcessEvent` و `GetEventMask` را در زیر می بینید که تعریف شده اند.

در کلاس `CPlayerComponent` در یک فایل `cpp` یک رویداد سیستمی با اسم تابع `GetEventMask` به صورت `const` (بدون تغییر و مقدار ثابت) وجود دارد و تنها دو مقدار از `enum` تعریف شده با اسم `EEntityEvent` را برمیگرداند و توسط ماکرو تعریف شده `BIT64` دو مقدار زیر توسط کلمه کلید `return` برگشت داده می شود، اگر توجه کنید مقدار برگشتی تابع نیز به صورت `uint64` است به این معنی که عدد صحیح طولانی (`log int`) است :

`ENTITY_EVENT_START_GAME` : هنگامی که بازی شروع می شود، تابع `GetEventMask` این مقدار را به `switch` تابع رویدادی `ProcessEvent` ارسال می کند و یکبار این عمل اتفاق می افتد
`ENTITY_EVENT_UPDATE` : هنگامی که بازی در حال اجرا است، تابع `GetEventMask` این مقدار را به `switch` تابع رویدادی `ProcessEvent` ارسال می کند و این عمل تا زمان پایان بازی انجام می شود

تابع رویدادی `ProcessEvent` همانطور که از نامش پیداست پردازش رویدادها را انجام می دهد و مقادیر مختلف از رویدادها را دریافت می کند و این مقادیر با پارامتر متغیری ساختار `SEntityEvent` دریافت شده و با استفاده از `event` که آدرس دهی را برای مقادیر پارامترها با رویدادهای مختلف انجام می دهد، این پارامتر متغیری ساختاری به یک `switch` تحویل داده می شود و سپس با استفاده از مقادیری که می تواند با `switch` برابری کند، `case` مورد نظر اجرا می شود مثلا :

هنگامی که بازی شروع می شود (با مقدار پارامتری `ENTITY_EVENT_START_GAME`) تابع غیر سیستمی (توسط برنامه نویس نوشته شده) `Revive()` اجرا می شود و هنگامی که بازی در حال اجرا است توابع غیر سیستمی زیر اجرا می شوند:

`UpdateMovementRequest`: این تابع فیزیک حرکت پلیمر را محاسبه و اجرا می کند

`UpdateLookDirectionRequest`: این تابع جهت حرکت پلیمر را محاسبه و اجرا می کند

`UpdateAnimation`: این تابع انیمیشن های مختلف مربوط به پلیمر را نمایش می دهد

`UpdateCamera`: این تابع به روزرسانی چرخش دوربین و پردازش دوربین را انجام می دهد

`OnUpdate`: این تابع نیز دیگر عملیات مربوط به کنترل پلیمر را برعهده دارد

نکته ای که نباید فراموش کنید این است که `deltaTime` های بازی در حال اجرا داخل توابع غیرسیستمی ارسال میشود و این توابع از این `deltaTime` ها (`pCtx->fFrameTime`) استفاده می کنند، مطمئناً زمان در حرکت پلیمر، فیزیک پلیمر، دوربین پلیمر و نمایش انیمیشن های پلیمر نقش مهمی دارد، البته بیشتر `Entity` ها با زمان کار می کنند و نحوه پردازش و کنترل فریم به فریم در زمان بازی بسیار مهم و حیاتی است.

```
uint64 CPlayerComponent::GetEventMask() const
{
    return BIT64(ENTITY_EVENT_START_GAME) |
        BIT64(ENTITY_EVENT_UPDATE);
}
```

```
CPlayerComponent::ProcessEvent(SEntityEvent&
event)
{
    switch (event.event)
    {
        case ENTITY_EVENT_START_GAME:
        {
            // Revive the entity when gameplay
            starts
            Revive();
        }
        break;
        case ENTITY_EVENT_UPDATE:
        {
            SEntityUpdateContext* pCtx =
            (SEntityUpdateContext*)event.nParam[.];

            // Start by updating the movement
            request we want to send to the character
            controller
            // This results in the physical
            representation of the character moving
            UpdateMovementRequest(pCtx-
            >fFrameTime);

            // Process mouse input to update
            look orientation.
            UpdateLookDirectionRequest(pCtx-
            >fFrameTime);

            // Update the animation state of the
            character
```

```
UpdateAnimation(pCtx->fFrameTime);
```

```
// Update the camera component offset
```

```
UpdateCamera(pCtx->fFrameTime);
```

```
OnUpdate(pCtx->fFrameTime);
```

```
}
```

```
break;
```

```
}
```

```
}
```

در اینجا مثال دیگری را نیز تشریح می کنم، تابع سیستمی `GetEventMask` حامل پیغامی بازگشتی با نام `ENTITY_EVENT_COLLISION` است و به صورت بازنویسی شده مجدد با مقدار ثابت اعلان شده است، همچنین تابع سیستمی دیگر که با تابع قبلی مکمل است، عمل پردازش رویداد با نام `ProcessEvent` را برعهده دارد، با استفاده از دستور `if` این بار بررسی می شود که عمل برخورد (`Collision`) اتفاق افتاده است یا نه؟ عمل برخورد به این معنا است وقتی که شی سخت و سفتی به شی سخت و سفت دیگری برخورد می کند، رویداد `ENTITY_EVENT_COLLISION` اتفاق می افتد، مثلاً برخورد یک توپ به زمین، برخورد گلوله به دشمن، برخورد راکت به تانک و غیره، در این مثال اگر عمل برخورد اتفاق بیفتد یعنی `entity` جاری به یک شی سخت و سفت دیگر برخورد کرده است، `entity` جاری با استفاده از تابع `RemoveEntity` در کتابخانه `pEntitySystem` در ریشه دسترسی به دستورات کل کرای انجین با استفاده از `Id` که `entity` جاری یعنی `GetEntityId` را گرفته و `entity` جاری حذف می شود، مثلاً اگر این کد را به توپ تنیس دهیم و اگر توپ تنیس به زمین

برخورد کند و از آنجایی که این کد به توپ تنیس داده شده است، توپ تنیس حذف خواهد شد

```
virtual uint64 GetEventMask() const override {
return BIT64(ENTITY_EVENT_COLLISION); }
virtual void ProcessEvent(SEntityEvent&
event) override
{
// Handle the OnCollision event, in
order to have the entity removed on collision
if (event.event ==
ENTITY_EVENT_COLLISION)
{

gEnv->pEntitySystem-
>RemoveEntity(GetEntityId());

}

}
```

در ادامه نوبت به توسعه Entity سفارشی تان می رسد و بیشتر آن را توسعه دهید، ایجاد یک کامبونت گلوله به صورت پریفب و سیماتیک را در کد زیر مشاهده می کنید، ابتدا یک هدر فایل را ایجاد و سپس یک سی پی پی فایل را ایجاد خواهید کرد و این دو فایل را با نام های Bullet.h و Bullet.cpp به داخل مسیری پروژه RobotIsland\Code\Components کپی-پیست کنید و از آنجایی که در فصل قبلی توضیحات خوبی برای کدها را ارائه دادم و این خطوط در

هدر فایل ها و سی پی پی فایل ها بیشتر مواقع تکرار خواهد شد، از تکرار توضیحات مجدد چشم پوشی می کنم و در ادامه صفحات نیز فقط به ذکر کدهای جدید بسنده خواهیم کرد.

یک فایل هدر را با نام **Bullet.h** ایجاد کنید و کدهای زیر را در آن تایپ کنید :

#pragma once

```

////////////////////////////////////
////////////////////////////////////
// Physicalized bullet shot from weaponry,
// expires on collision with another object
////////////////////////////////////
////////////////////////////////////
class CbulletComponent final : public
IEntityComponent
{
public:
    virtual ~CbulletComponent() {}

    // IEntityComponent
    virtual void Initialize() override
    {
        // Set the model
        const int geometrySlot = 0;
        m_pEntity->LoadGeometry(geometrySlot,
        "Objects/Default/primitive_sphere.cgf");

        // Load the custom bullet material.
        // This material has the 'mat_bullet'
        surface type applied, which is set up to play
    }
};

```


sounds on collision with 'mat_default' objects in Libs/MaterialEffects

```
auto *pBulletMaterial = gEnv-
>p3Dengine->GetMaterialManager()-
>LoadMaterial("Materials/bullet");
```

```
m_pEntity-
>SetMaterial(pBulletMaterial);
```

```
// Now create the physical
representation of the entity
SEntityPhysicalizeParams physParams;
physParams.type = PE_RIGID;
physParams.mass = ۲۰۰۰۰.f;
```

```
//m_pEntity->SetScale(Vec۳(۲, ۲, ۲));
m_pEntity->Physicalize(physParams);
```

```
// Make sure that bullets are always
rendered regardless of distance
```

```
// Ratio is ۰ - ۲۵۵, ۲۵۵ being ۱۰۰%
visibility
```

```
GetEntity()->SetViewDistRatio(۲۵۵);
```

```
// Apply an impulse so that the
bullet flies forward
```

```
if (auto *pPhysics = GetEntity()-
>GetPhysics())
{
    pe_action_impulse impulseAction;
```

```

        const float initialVelocity =
        ۱۰۰۰.f;

        // Set the actual impulse, in
        this cause the value of the initial velocity
        Cvar in bullet's forward direction
        impulseAction.impulse =
        GetEntity()->GetWorldRotation().GetColumn\() *
        initialVelocity;

        // Send to the physical entity
        pPhysics-
        >Action(&impulseAction);

    }

    // Reflect type to set a unique identifier
    for this component
    static void
    ReflectType(Schematyc::CtypeDesc<CbulletCompon
    ent>& desc)
    {
        desc.SetGUID("{B۵۳A۹A۵F-F۲۷A-۴۲CB-۸۲C۷-
        B۱E۳۷۹C۴۱A۲A}"_cry_guid);
        desc.SetEditorCategory("Game");
        desc.SetLabel("Bullet");
        desc.SetDescription("This Bullet can
        be used to spawn entities");
    }

```

```

        desc.SetComponentFlags({
            IEntityComponent::Eflags::Transform,
            IEntityComponent::Eflags::Socket,
            IEntityComponent::Eflags::Attach });
    }

    virtual uint64 GetEventMask() const
    override { return
        BIT64(ENTITY_EVENT_COLLISION); }

    virtual void ProcessEvent(SentityEvent&
        event) override
    {
        // Handle the OnCollision event, in
        // order to have the entity removed on collision
        if (event.event ==
            ENTITY_EVENT_COLLISION)
        {
            gEnv->pEntitySystem-
            >RemoveEntity(GetEntityId());
        }
    }
    // ~IEntityComponent

public:
    void SpawnEntityBullet(Ientity*
        otherEntity);
};

```

برای آغاز منطق هر entity یا component در بازی لازم است که از توابع سیستمی مختلف استفاده شود، درک این توابع بسیار ساده است و

چگونگی پیاده سازی این توابع را خواهید آموخت، اولین تابع سیستمی با نام تابع آغازگر یا با نام تابع مقداردهنده است که هر زمان که ساخت یک entity و یا حتی component توسط کرای انجین انجام می شود، این تابع اجرا خواهد شد، مثلاً یک entity را به داخل بازی & drag drop می کنید، این تابع اجرا خواهد شد و کلیه دستورات داخل این تابع نیز به طبع اجرا خواهد شد، اسم این تابع Initialize است و دستورات زیر را با توجه به هدر فایل Bullet.h اجرا خواهد کرد :

```
virtual void Initialize() override
```

```
{
```

این تابع با توجه به مجازی بودن عملکرد سیستمی اش بارنویسی و بازطراحی می شود

```
const int geometrySlot = ۰;
```

یک متغیر با نام geometrySlot تعریف می شود و مقدارش غیرقابل تغییر و ثابت است و مقدار صفر برای آن اختصاص داده می شود.

```
m_pEntity-
```

```
>LoadGeometry(geometrySlot,
```

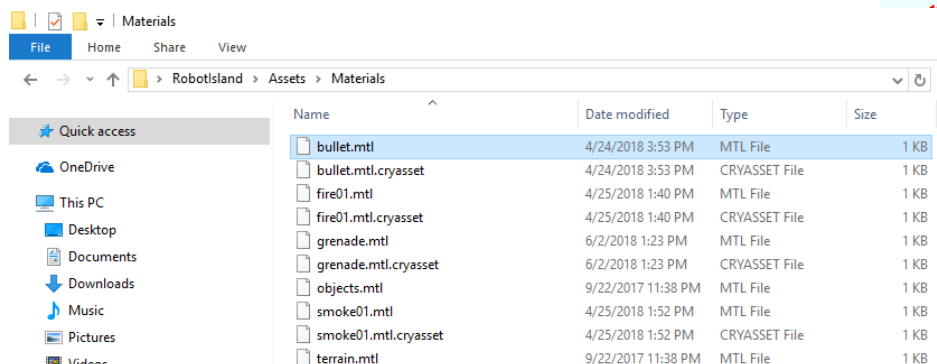
```
“Objects/Default/primitive_sphere.cgf”);
```

اشاره گر داخلی با نام m_pEntity، برای entity جاری یا component جاری یک دستور بارگذاری (لود) را صادر می کند که باعث می شود یک مدل سه بعدی (هندسی) با نام primitive_sphere.cgf در slot پنجمه Assets

Browser در مسیر **Objects/Default** نمایش داده شود، در واقع entity یا Component یک مدل سه بعدی را نمایش می دهد

```
auto *pBulletMaterial = gEnv-
>p3Dengine->GetMaterialManager()-
>LoadMaterial("Materials/bullet");
```

مدل سه بعدی تان را لود کردید و حالا نوبت به آن رسیده که متریالی که قبلاً در **Material Editor** ایجاد کردید به شی تان اختصاص دهید، این دستور باعث می شود که با توجه به پنجره **Assets Browser** در مسیر **Materials** یا در پنجره **Windows Explorer** با توجه به تصویر پایین متریال **bullet.mtl** بر روی مدل سه بعدی لود شود، برای لود متریال به وسیله تابع **LoadMaterial** نباید متریال با پسوند ***.mtl** را بنویسید و تنها کافیست اسم متریال نوشته شود مثل آنچه که در کد بالا وجود دارد، همانطور که می بینید یک متغیر اشاره گر با نام **pBulletMaterial** تعریف شده است و متریال **bullet** در داخل این متغیر آدرس دهی می شود تا عمل لود انجام شود، کلیه دستورات در کرای انجین به این کلمه **gEnv** ختم می شود و با توجه به کتابخانه های اشاره گری مختلف و عمل پاس شدن از مسیرهای کتابخانه ای و توابع مختلف باید به تابع **LoadMaterial** برسید.



```
m_pEntity->SetMaterial(pBulletMaterial);
```

با این خط با توجه به اشاره گر داخلی در کلاس فعلی، تابع **SetMaterial** عمل نمایش دادن متریال **bullet** را بر روی مدل سه بعدی انجام می دهد.

```
// Now create the physical
representation of the entity
SEntityPhysicalizeParams physParams;
```

تعریف یک متغیر ساختاری (struct) از نوع ساختمان داده **SEntityPhysicalizeParams** که در هدر فایل **IEntity.h** وجود دارد که مقادیر مربوط به متغیر ساختاری **physParams** را باید پر کنید

```
physParams.type = PE_RIGID;
```

نوع فیزیک به کار رفته از رفتار **Rigidbody** تبعیت می کند و از آنجایی که مدل سه بعدی لود شده در کرای انجین شامل **proxy** است یعنی بدنه

سفت و سخت به آن اضافه شده و قوانین فیزیک برروی مدل سه بعدی انجام می شود

```
physParams.mass = ۲۰۰۰۰۰.f;
```

جرم شی یا مدل سه بعدی ۲۰ کیلوگرم یا همان ۲۰۰۰۰ گرم است

```
//m_pEntity->SetScale(Vec۳(۱, ۲, ۳));
```

مقیاس شی را می توانید تغییر دهید، این خط به حالت توضیح درآورده شده است به این معنا که خط مربوطه کامپایل نخواهد شد اما اگر خط از حالت توضیح خارج شود، اندازه شی (مقیاس شی جاری) به صورت محورهای $x=1$ $z=3$ و $y=2$ خواهد بود، می توانید اندازه این شی را مربع در نظر بگیرید مانند $z=2$ و $y=2$ و $x=2$ باشد.

```
M_pEntity->Physicalize(physParams);
```

شی جاری باید رفتار فیزیک را در خود حمل کند و با تابع **Physicalize** این عمل انجام می شود

```
GetEntity()->SetViewDistRatio(۲۵۵);
```

تابع **SetViewDistRatio** شی را تا فاصله ۲۵۵ متر از پلیمر را نشان خواهد داد

```
if (auto *pPhysics = GetEntity()->GetPhysics())
```

```
{
```

این خط بررسی می کند که آیا تخصیص حافظه برای یک متغیر اشاره گر با نام **pPhysics** انجام شده است و اگر انجام شده است آیا شی دارای

معادلات فیزیک است و `rigidbody` و `proxy` به آن اختصاص داده شده است، اگر اینگونه باشد، بلوک `if` اجرا می شود

```
pe_action_impulse impulseAction;
```

حالا نوبت اختصاص شتاب (تکانه یا ضربه) به شی فرا رسیده است و برای اختصاص تکانه یا ضربه به شی باید یک متغیر از نوع ساختمان داده ساختاری `pe_action_impulse` در هدر فایل `physinterface.h` با وراثت ساختاری از `pe_action` تعریف کنید و نام آن را `impulseAction` بگذارید

```
const float initialVelocity =  
۱۰۰۰.f;
```

تعریف یک متغیر دیگر از نوع غیرقابل تغییر با نام `initialVelocity` را انجام دهید و عدد ۱۰۰۰ نیوتن به آن اختصاص دهید، قرار است که این نیرو از تکانه به شی وارد شود و لازم به ذکر است که نیوتن واحد نیروی وارده بر جسم است، قوانین فیزیک کلاسیک نیوتن به این شرح بود :

قانون اول : هر جسمی (در اینجا `entity`) در هر حالتی که باشد تمایل دارد به همان حالت بماند، مثلاً وقتی یک توپ بر روی زمین است و ساکن است، همانطور ساکن می ماند، به این قانون لختی نیز گفته می شود.

قانون دوم : اگر به یک `entity` نیروهایی وارد شود، شتابی میگیرد که با برآیند نیروهای وارده بر `entity` نسبت مستقیم دارد و با آن هم جهت است و با جرم `entity` نسبت وارونه دارد :

$$a = \frac{f}{m}$$

در اینجا f نیرو است و m جرم entity و a شتاب است

قانون سوم : به هر entity که نیرو وارد کنیم ،آن entity نیز به همان اندازه به واردکننده نیرو،نیرو وارد خواهد کرد،مانند اینکه شما به یک دیوار نیرو وارد می کنید و دیوار نیز به همان اندازه و در خلاف جهت نیرو شما نیرو وارد خواهد کرد.

```
impulseAction.impulse =
GetEntity()->GetWorldRotation().GetColumn\()*
initialVelocity;
```

با توجه به اینکه متغیر ساختاری impulseAction باید impulse را داشته باشد با استفاده از گرفتن خصیصه چرخش از نوع سراسری world و این چرخش طبق محور y با استفاده از GetColumn\() انجام شود و باید به متغیر نیوتنی initialVelocity ضرب شود تا تکانه در متغیر ساختاری impulseAction با فیلد impulse ثبت شود

در واقع عمل تکانه یا زدن ضربه طبق محور y یعنی به طرف عمق (به طرف جلو) مقدار دهی می شود و دستور GetColumn\() محور x (سمت راست) است، GetColumn\() محور y (به سمت عمق و رو به جلو) است و GetColumn\() محور z (سمت بالا) است

```
pPhysics-
>Action(&impulseAction);
```

حالا با استفاده از متغیر pPhysics از نوع فیزیک،با تابع Action ،تکانه انجام و اجرا می شود و شی جاری به طرف جلو پرتاب می شود.

```
}
}
```

خطوط دیگری هستند که تکرار فصل قبلی است که از توضیح مجدد آن صرف نظر کردم

```
if (event.event ==
ENTITY_EVENT_COLLISION)
{
```

اینجا بررسی می کند که اگر گلوله (مدل سه بعدی هندسی که فیزیک نیرو بر اساس تکانه است) به سطح جسمی برخورد کند (Collision) چه کاری انجام شود

```
gEnv->pEntitySystem-
>RemoveEntity(GetEntityId());
```

گلوله براساس تابع RemoveEntity با توجه به Id که به صورت فعلی دریافت کرده توسط تابع GetEntityId حذف می شود و نیروی گلوله (برروی اشیاء که فیزیک دارند) اثر خواهد کرد و اشیاء تکان خواهند خورد و اشیاء که فیزیک ندارند، گلوله هیچ تاثیری برروی آنها نخواهد گذاشت، این فقط یک مثال پایه برای درک کدها در صفحات بعدی است.

```
}
```

یک فایل کد را با نام Bullet.cpp را ایجاد کنید و کد زیر را در آن تایپ کنید :

```
#include "StdAfx.h"
#include "Bullet.h"
```

```
#include <CrySchematyc/Env/IEnvRegistrar.h>
```

```
#include
```

```
<CrySchematyc/Env/Elements/EnvComponent.h>
```

```
static void
```

```
RegisterBulletComponent(Schematyc::IEnvRegistrar& registrar)
```

```
{
    Schematyc::CEnvRegistrationScope scope =
    registrar.Scope(IEntity::GetEntityScopeGUID())
```

```
;
```

```
{
    Schematyc::CEnvRegistrationScope
    componentScope =
    scope.Register(SCHEMATYC_MAKE_ENV_COMPONENT(CBulletComponent));
    // Functions
```

```
{
```

```
{
```

```
}
```

```
}
```

```
}
```

```
CRY_STATIC_AUTO_REGISTER_FUNCTION(&RegisterBulletComponent)
```

```
void
```

```
CBulletComponent::SpawnEntityBullet(IEntity* otherEntity)
```

```
{
```

```
otherEntity->SetWorldTM(m_pEntity->GetWorldTM());
```

```
}
```

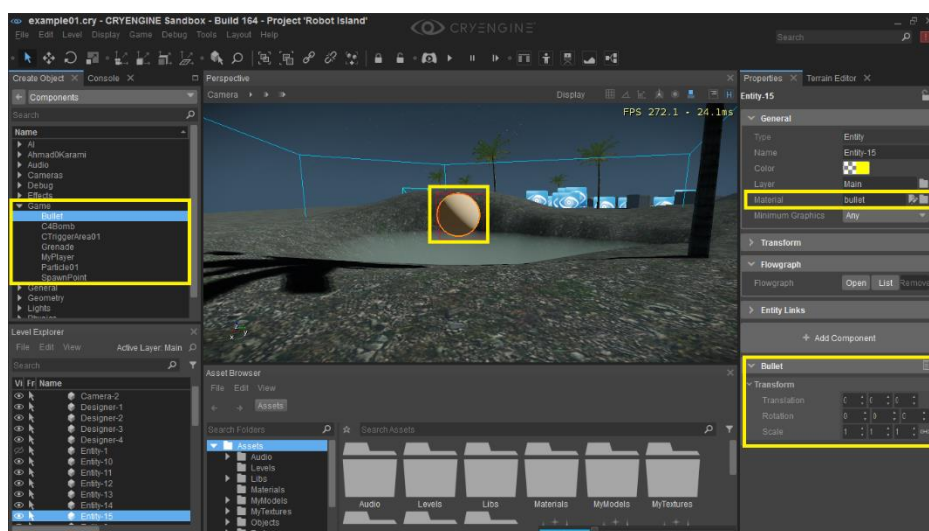
تعدادی از خطوط کدها در فصل قبلی توضیح داده شده است و من از توضیحات مجدد این کدها صرف نظر می کنم مانند خط کد هدر فایل `"StdAfx.h"` `#include` که البته توضیحات مفصل آن را در فصل بعدی ارائه خواهد شد

`#include "Bullet.h"`

از آنجایی که در فصل قبلی هدر فایل ها را براساس فایل های سی پی پی را توضیح دادم و کدهای آن را نیز تایپ کردید، برای استفاده از این هدر فایل باید از پیش پردازنده با کلمه کلیدی `include` استفاده کنید و البته بقیه کدها نیز مشابه توضیحات در فصل قبلی است، نگران نباشید مثال های دیگری هم هستند که باید یاد بگیرید و از این مثال ها استفاده کنید و درک بسیار بهتری از ساخت `entity` ها و `component` ها را بدست خواهید آورد

فراموش نباید شود که این دو فایل `Bullet.h` و `Bullet.cpp` در حتما در `CMakeLists.txt` باید ثبت کنید، نحوه ثبت کدهای `*.h` و `*.cpp` را در فصل قبل توضیح داده ام و به یاد آورید که برای ساخت هر `entity` یا `component` و نمایش آن در سندباکس مانند آنچه که در تصویر زیر می بینید حتما یک هدر فایل و یک سی پی پی فایل را در `CMakeLists.txt` ثبت نمایید و اگر برای ساخت هر `entity` یا `component` فقط یک هدر فایل استفاده کنید، نمی توانید آن را در طراحی مرحله و در داخل سندباکس استفاده کنید و تنها در داخل بازی قابل استفاده

است و داخل ادیتور کرای انجین در پنجره **Create Object** بخش **Components** قابل دسترسی و قابل دیدن نیست مانند هدر فایل **Bullet.h** در پروژه تمپلت **FirstPersonShooter** که فقط در بازی قابل استفاده است و در ادیتور کرای انجین در پنجره **Create Object** در بخش **Components** ها قابل نمایش و قابل دسترسی نیست، و در این فصل به بررسی هدر فایل **Bullet.h** در پروژه تمپلت **FirstPersonShooter** خواهیم پرداخت



حالا مر رسیم به یک مثال از آتشی که جاذبه دارد و بعد از ۳ ثانیه از بین می رود و اسم این Entity را Particle۰۱ گذاشته ام و یک مثال پایه دیگری است تا بتوانید با مفهوم **deltaTime** بیشتر آشنا شده و کنترل زمان با استفاده از یک متغیر به نام **sum** و یک تابع غیرسیستمی با نام **OnUpdate** را بهتر درک کنید.

ابتدا هدر فایل را ایجاد و سپس سی پی پی فایل را ایجاد خواهید کرد و این دو فایل را با نام های Particle۰۱.h و Particle۰۱.cpp در مسیر پروژه RobotIsland\Code\Components کپی-پیست کنید و از آنجایی که در صفحات قبلی خطوطی از کدها را توضیح دادم و این خطوط در هدر فایل ها و سی پی پی فایل ها تکرار خواهد شد، از تکرار توضیحات مجدد چشم پوشی می کنم و در ادامه صفحات نیز فقط به ذکر کدهای جدید بسنده خواهیم کرد، لازم به ذکر مجدد است که از این دو فایل به عنوان یک Entity جدید یا component جدید می توانید استفاده کنید.

یک فایل کد را با نام Particle۰۱.h را ایجاد کنید و کد زیر را در آن تایپ کنید :

```
#pragma once
#include <CryEntitySystem/IEntityComponent.h>
#include <CryEntitySystem/IEntity.h>
#include <CryParticleSystem\IParticlesPfx.h>
#include <CryParticleSystem\IParticles.h>
////////////////////////////////////
//////////
// Spawn point
////////////////////////////////////
//////////
class CParticle۰۱ final : public
IEntityComponent
{
public:
CParticle۰۱() = default;
virtual ~CParticle۰۱() {}
```

```

virtual void Initialize() override
{

    const int geometrySlot = .;
    m_pEntity->LoadGeometry(geometrySlot,
    "Objects/Default/primitive_sphere.cgf");

    auto *pGrenadeMaterial = gEnv->p3DEngine-
    >GetMaterialManager()-
    >LoadMaterial("Materials/bullet");

    m_pEntity->SetMaterial(pGrenadeMaterial);

    // Now create the physical representation of
    the entity
    SEntityPhysicalizeParams physParams;
    physParams.type = PE_RIGID;
    physParams.mass = ۱۵ ; //۲۰۰۰۰۰.f;
    //  m_pEntity->SetScale(Vec3(۰.۱, ۰.۱, ۰.۱));
    m_pEntity->Physicalize(physParams);

    IParticleEffect* pEffect = gEnv-
    >pParticleManager->FindEffect("fire.pfx");
    GetEntity()->LoadParticleEmitter(., pEffect);

}

```

```

// Reflect type to set a unique identifier for
this component
// and provide additional information to
expose it in the sandbox
static void
ReflectType(Schematyc::CTypeDesc<CParticle>&
desc)
{
desc.SetGUID("{A۰CB۲۰E۹-۹۴BD-۴۷F۴-AB۴۱-
E۰۱CYA۲EDC۴F}"_cry_guid);
desc.SetEditorCategory("Game");
desc.SetLabel("Particle");
desc.SetDescription("This Particle can be
used to spawn entities , Ahmad Karami ha ha ha
;-)");
desc.SetComponentFlags({
IEntityComponent::EFlags::Transform,
IEntityComponent::EFlags::Socket,
IEntityComponent::EFlags::Attach });
}

virtual void ProcessEvent(SEntityEvent& event)
override {
if (event.event == ENTITY_EVENT_UPDATE) {

SEntityUpdateContext*
pcx=(SEntityUpdateContext*)event.nParam[۰];

OnUpdate(pcx->fFrameTime);

}
}

```



```

if (event.event == ENTITY_EVENT_COLLISION) {

    // You type Code
    // gEnv->pEntitySystem-
    >RemoveEntity(GetEntityId());

}

}

virtual uint64 GetEventMask() const override {
    // Listen to the enter and leave events, in
    // order to receive callbacks above when entities
    // enter our trigger box
    return BIT64(ENTITY_EVENT_UPDATE) |
    BIT64(ENTITY_EVENT_COLLISION);
}

public:
void SpawnEntityParticle(const IEntity*
otherEntity);
void OnUpdate(float deltaTime);
float sum=0.f;

};

```

حالا به بررسی هدر فایل پارتیکل می پردازم :

```
#include <CryParticleSystem\IParticlesPfx.h>
#include <CryParticleSystem\IParticles.h>
```

این دو خط به تمام ویژگی های قدیمی و جدید از توابع پارتیکل سیستم دسترسی پیدا می کند.

در تابع سیستمی `ProcessEvent` دو نوع رویداد با دستور `if` بررسی می شود، اولین رویداد به صورت `Update` و دومی به صورت `Collision` است، در رویداد `Update` تابعی با نام `OnUpdate` تعریف شده است و برش های زمانی با دستور `pcx->fFrameTime` برای تابع ارسال می شود و در داخل تابع چه اتفاقی می افتد در فایل `Particle۰۱.cpp` بررسی می کنم، در رویداد `Collision` نیز هیچ اتفاقی نمی افتد و تنها این رویداد اعلان شده است و تابعی برای آن تعریف نشده است و تنها یک توضیح به رنگ سبز رنگ آمده است که کدهایتان را تایپ کنید (`// You type Code`) و این بستگی به شما دارد که هنگام برخورد `entity` پارتیکل چه اتفاقی برای آن بیفتد.

در تابع سیستمی `GetEventMask` نیز دو مقدار برای تابع سیستمی `ProcessEvent` بر میگرداند که می تواند شامل رویداد `Update` یا رویداد `Collision` باشد و این دو مقدار ثابتی هستند که در `entity` پارتیکل می توان از آن دو رویداد استفاده کرد، البته می توان آن دو رویداد را به سه رویداد یا چهار رویداد و بیشتر نیز توسعه داد.

```
void SpawnEntityParticle۰۱(IEntity*
otherEntity);
```

تابع `SpawnEntityParticle` ۰۱ نیز به مانند توابع دیگر برای تخصیص موقعیت `x,y,z` برای `entity` پارتيكل به حساب می آید.

```
void OnUpdate(float deltaTime);
```

پارامتر مقداری برش های زمانی (`deltaTime`) به صورت عدد اعشاری (`float`) به تابع غیر سیستمی `OnUpdate` ارسال می شود، در اینجا کلمه `public` باعث می شود که این تابع یا هر آنچه که در محدوده `public` باشد خارج از کلاس در شی مورد دسترسی قرار گیرد، بهتر از برای جلوگیری از دسترسی برنامه نویس یا اشیاء دیگر به این کلاس از کلمه کلیدی `private` استفاده میشد، اما در اینجا فقط مثالی برای نحوه استفاده و بیان کلمات `public` و `private` آورده ام.

```
float sum=۰.f;
```

یک متغیر نیز برای جمع برش های زمانی (`deltaTime`) با نوع داده عددی اعشاری تعریف کردم.

یک فایل کد را با نام `Particle۰۱.cpp` را ایجاد کنید و کد زیر را در آن تایپ کنید :

```
#include "StdAfx.h"
```

```
#include "Particle۰۱.h"
```

```
#include <CrySchematyc/Reflection/TypeDesc.h>
```

```
#include <CrySchematyc/Utils/EnumFlags.h>
```

```
#include <CrySchematyc/Env/IEnvRegistry.h>
```

```
#include <CrySchematyc/Env/IEnvRegistrar.h>
```

```
#include
```

```
<CrySchematyc/Env/Elements/EnvComponent.h>
```

```
#include
```

```
<CrySchematyc/Env/Elements/EnvFunction.h>
```

```

#include
<CrySchematyc/Env/Elements/EnvSignal.h>
#include <CrySchematyc/ResourceTypes.h>
#include <CrySchematyc/MathTypes.h>
#include <CrySchematyc/Utils/SharedString.h>

static void
RegisterCParticle( (Schematyc::IEnvRegistrar&
registrar)
{
    Schematyc::CEnvRegistrationScope scope =
registrar.Scope( IEntity::GetEntityScopeGUID() )
;
    {
        Schematyc::CEnvRegistrationScope
componentScope =
scope.Register( SCHEMATYC_MAKE_ENV_COMPONENT( CP
article ) );
        // Functions
        {
        }
    }
}

CRY_STATIC_AUTO_REGISTER_FUNCTION(&RegisterCPa
rticle)

void
CParticle::SpawnEntityParticle( IEntity*
otherEntity)
{

```

```

otherEntity->SetWorldTM(m_pEntity-
>GetWorldTM());

}

```

```

void CParticle::OnUpdate(float deltaTime)
{
    sum+=deltaTime;

    if(sum>=۳)
    gEnv->pEntitySystem-
    >RemoveEntity(GetEntityId());
}

```

قبلا توضیحات مفصلی در رابطه با عملکرد و ارتباط بین هدر فایل ها و سی پی پی فایل ها در فصل قبل بیان نموده ام و حتما به فصل قبل مراجعه نموده و مطالب را مجددا مطالعه کنید و در آن بخش های مختلف هدر فایل ها و سی پی پی فایل ها بررسی نموده ام و در اینجا تنها به بررسی کدهای پیاده سازی OnUpdate زیر می پردازم :

```

void CParticle::OnUpdate(float deltaTime)
{

```

این تابع در کلاس CParticle۰۱ تعریف شده است و شامل یک مقدار پارامتری برش زمانی با نام `deltaTime` از نوع داده عددی اعشاری وجود دارد

```
sum+=deltaTime;
```

با استفاده از متغیر `sum` برش های زمانی را به صورت پیوسته و مجدد با هم جمع می کند و این برش ها شامل ثانیه ها خواهد شد.

```
if(sum>=۳)
```

اگر جمع برش های زمانی برابر یا بیشتر از عدد ۳ شد خط بعدی اجرا می شود، در غیراینصورت تا زمانی که شرط برقرار نباشد خط بعدی اجرا نخواهد شد.

```
gEnv->pEntitySystem-  
>RemoveEntity(GetEntityId());
```

با استفاده از این خط و با برقراری شرط بالا شی جاری حذف خواهد شد یعنی متغیر `sum` با شدن ۳ ثانیه و بزرگتر از ۳ ثانیه پارتیکل سیستم یا همان شی فعلی حذف خواهد شد.

```
}
```

لازم به ذکر است که پارتیکلی که کدنویسی کردم شامل جاذبه است و می تواند سقوط کند، در واقع من کدهای `Bullet.h` و `Bullet.cpp` را به همراه چندین خط کد جدید که در صفحات قبلی نیز توضیح داده ام، باهم ترکیب کرده ام و در کدهای `Particle۰۱.h` و `Particle۰۱.cpp` قرار دادم و نتیجه آن شد که آتشی (`fire.pfx`) که جاذبه دارد و بعد از ۳ ثانیه خاموش می شود، شما می توانید کد دیگری را بنویسید که به جای پارتیکل آتش به پارتیکل انفجاری که جاذبه دارد تغییر

دهید، این کار به سادگی آب خوردن است، شما دو راه دارید یا از ویژگی های Properties در C++ استفاده کنید (در این فصل به بررسی آن می پردازم) یا یک کلاس دیگر پارتیکلی با نام Particle۰۲.h و Particle۰۲.cpp ایجاد کنید و لود شدن پارتیکل را به انفجار تغییر دهید

```
IParticleEffect* p۰۱effect = gEnv-
>pParticleManager->FindEffect("fire.pfx");
```

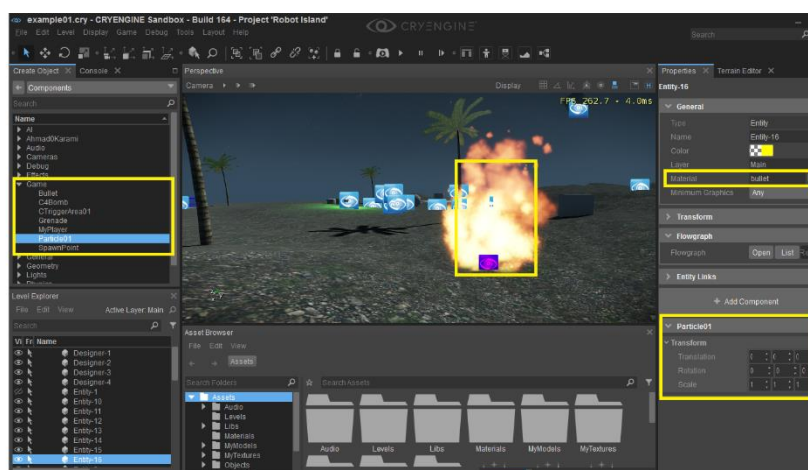
در صورتی که دوست دارید زمانی که آتش بر روی سطحی برخورد کرد و از بین برود خط توضیح // مربوط به رویداد Collision را در هدر فایل Particle۰۱.h غیر فعال کنید

```
// gEnv->pEntitySystem-
>RemoveEntity(GetEntityId()));
```

شما می توانید خلاقیت های دیگری را به خرج دهید تا بازی تان جذاب تر شود.

فراموش نباید شود که این دو فایل (Particle۰۱.h و Particle۰۱.cpp) در حتما در CMakeLists.txt باید ثبت کنید، نحوه ثبت کدهای *.cpp و *.h را در فصل قبلی توضیح داده ام و همچنین به یاد آورید که GUID جدیدی به هر entity یا component باید اختصاص دهید تا در سندباکس بتوانید به آن در

پنجره Create Object در بخش Components دسترسی داشته باشید



حالا بیا یک entity را بسازیم که یک نارنجک کامل باشد، منفجر شود و هر چیزی که در اطراف آن دارای فیزیک باشد، از بین رود، ابتدا هدر فایل را ایجاد و سپس سی پی پی فایل را ایجاد کنید و این دو فایل را با نام های Grenade.h و Grenade.cpp در مسیر ————— زیر پروژه RobotIsland\Code\Components کیست کنید.

هدر فایل نارنجک با نام Grenade.h را با کدهای زیر تایپ کنید

#pragma once

```
////////////////////////////////////
////////////////////////////////////
```

// C++ Code by Ahmad Karami ۲۰۱۸

```
////////////////////////////////////
////////////////////////////////////
```

```
class CGrenadeComponent final : public  
IEntityComponent
```



```

public:
virtual ~CGrenadeComponent() {}

virtual void Initialize() override
{
    const int geometrySlot = .;
    m_pEntity->LoadGeometry(geometrySlot,
        "MyModels/Grenade/grenade.۱.cgf");

    auto *pGrenadeMaterial = gEnv->pRDEngine-
        >GetMaterialManager()-
        >LoadMaterial("MyModels/Grenade/grenade");

    m_pEntity->SetMaterial(pGrenadeMaterial);

    SEntityPhysicalizeParams physParams;
    physParams.type = PE_RIGID;
    physParams.mass = ۲۰۰۰۰۰.f;
    //m_pEntity->SetScale(Vec۳(۲, ۲, ۲));
    m_pEntity->Physicalize(physParams);
    GetEntity()->SetViewDistRatio(۲۵۵);

    GetEntity()->SetName("Grenade");

    if (auto *pPhysics = GetEntity()-
        >GetPhysics())
    {
        pe_action_impulse impulseAction;

```

```

const float initialVelocity = ۱۰.۵f;

impulseAction.impulse =ahmadval *
initialVelocity;

pPhysics->Action(&impulseAction);

}

}

static void
ReflectType(Schematyc::CTypeDesc<CGrenadeCompo
nent>& desc)
{
desc.SetGUID("{B۵۸۶۰۰۷۳-۷DD۴-۴۸۲۱-۸A۹۹-
BY۰EB۶F۳D۶DB}"_cry_guid);
desc.SetEditorCategory("Game");
desc.SetLabel("Grenade");
desc.SetDescription("This Grenade can be used
to spawn entities");
desc.SetComponentFlags({
IEntityComponent::EFlags::Transform,
IEntityComponent::EFlags::Socket,
IEntityComponent::EFlags::Attach });
}

```

```

virtual uint64 GetEventMask() const override {
    return BIT64(ENTITY_EVENT_COLLISION) |
        BIT64(ENTITY_EVENT_UPDATE); }
virtual void ProcessEvent(SEntityEvent& event)
override
{
    if (event.event == ENTITY_EVENT_UPDATE)
    {

        SEntityUpdateContext* pCtx =
            (SEntityUpdateContext*)event.nParam[.];

        OnUpdate(pCtx->fFrameTime);

    }

    if (event.event == ENTITY_EVENT_COLLISION)
    {
        // Type your codes
        // an example : when you throw your greandes
        to forward , after collision on surfaces spawn
        a particle system like a bit dust 😊

        repeatCollisionCount++;

        if(repeatCollisionCount==۱)
        {
            SEntitySpawnParams spawnParamsDustParticle;
            spawnParamsDustParticle.pClass = gEnv-
                >pEntitySystem->GetClassRegistry()-
                >GetDefaultClass();

```

```
spawnParamsDustParticle.vPosition =
GetEntity()->GetPos();
```

```
const float ParticleScale۴ = ۰.۲۵f;
spawnParamsDustParticle.vScale =
Vec۳(ParticleScale۴);
```

```
IEntity* pEntityDust = gEnv->pEntitySystem-
>SpawnEntity(spawnParamsDustParticle);
```

```
pEntityDust-
>CreateComponentClass<CParticle۰۱>();
}
}
}
// ~IEntityComponent
```

```
public:
void SpawnEntityGrenade(IEntity* otherEntity);
void OnUpdate(float deltaTime);
float sum=۰.۰f;
bool Once=true;
int repeatCollisionCount=۰;

};
```

توضیحات مربوط به این خطوط کدها شبیه مثال های دیگر است اما تفاوت

های مهم و بسیار جدیدی نیز در آن می توان یافت، مثل خط زیر :

```
GetEntity()->SetName("Grenade");
```

این خط نام نارنجک (Entity جاری) را به Grenade تغییر نام می دهد

```
impulseAction.impulse =ahmadval *
initialVelocity;
```

جهت پرتاب نارنجک براساس این خط انجام می شود،متغیر استاتیک ahmadval که از نوع vec3 (بردار) است در داخل هدر فایل StdAfx.h به صورت خط زیر تعریف کرده ام :

```
static Vec3 ahmadval;
```

متغیر initialVelocity نیز مقدار نیرویی ثابت بر حسب نیوتن با ضرب جهت در متغیر ahmadval به نارنجک وارد می کند.(واحد نیرو یا واحد تکانه یا واحد ضربه بر حسب نیوتن است)

```
OnUpdate(pCtx->fFrameTime);
```

تابع غیرسیستمی OnUpdate برش های زمانی را به داخل تابع ارسال می کند.

من می توانستم یک تابع غیر سیستمی با یک نام بسازم و کدهای زیر را در آن درج نمایم اما این کار را برعهده شما می گذارم،حالا کدهای زیر را برای رویداد collision ببینید :

```
repeatCollisionCount++;
```

یک واحد به متغیر سراسری و صحیح repeatCollisionCount اضافه می کند،متغیر repeatCollisionCount مقدار پیش فرض صفر را دارد و حالا با یک واحد به آن افزوده می شود و برابر یک میگردد

```
if(repeatCollisionCount==۱)
{
```

این شرط بررسی می کند و تضمین می کند که تنها یکبار عملیات زیر اجرا شود و در صورتی که متغیر `repeatCollisionCount` برابر یک باشد عملیات زیر انجام شود

```
SEntitySpawnParams spawnParamsDustParticle;
```

یک متغیر با نام `spawnParamsDustParticle` از نوع داده ساختاری `SEntitySpawnParams` تعریف کرده ام.

```
spawnParamsDustParticle.pClass = gEnv-
>pEntitySystem->GetClassRegistry()-
>GetDefaultClass();
```

این خط، فیلد مربوطه با نام `pClass` را مقدار دهی میکند و مقدارش کلاس پیش فرضی است که قرار است پارتیکلی از آن کلاس پیش فرض یعنی `entity` جاری (نارنجک) تکثیر (`spawn`) شود و کلاس نارنجک به فیلد `pClass` اختصاص یابد

```
spawnParamsDustParticle.vPosition =
GetEntity()->GetPos();
```

این خط، فیلد مربوطه با نام `vPosition` (موقعیت) را مقدار دهی میکند و مقدارش موقعیت فعلی نارنجک (`entity` جاری) است

```
const float MyParticleScale = ۰.۲۵f;
```

این متغیر بر روی اندازه پارتیکل تاثیری نمیگذارد اما بر روی شعاع فیزیک پارتیکل تاثیر خواهد گذاشت

```
spawnParamsDustParticle.vScale =  
Vec3(MyParticleScale);
```

مقیاس پارتیکل تغییر می کند و با برداری سه بعدی طبق محورهای مختصات x, y, z به صورت اندازه مربعی (یا شعاعی) خواهد شد

```
IEntity* pEntityDust = gEnv->pEntitySystem->  
>SpawnEntity(spawnParamsDustParticle);
```

متغیری از نوع اشاره گر با ساختار `IEntity` با نام `pEntityDust` تعریف کرده ام و عمل تکثیر پارتیکل را براساس متغیر `spawnParamsDustParticle` آدرس دهی می کند، در نهایت باید آدرس کلاسی که عمل تکثیر پارتیکل را انجام می دهد را به متغیر `pEntityDust` اختصاص داد

```
pEntityDust->  
>CreateComponentClass<CParticle01>();
```

آدرس کلاسی `CParticle01` که باید عمل تکثیر پارتیکل را انجام می دهد به متغیر `pEntityDust` اختصاص داده شده و حالا پارتیکل تکثیر خواهد شد، به این معنی که هر گاه نارنجک پرتاب شود و نارنجک بر روی سطحی برخورد کند کلاس `CParticle01` تکثیر خواهد شد

```
}
```

بدیهی است که شما باید به جای کلاس `CParticle01` از کلاسی استفاده کنید که عمل تکثیر گرد و خاک را در هنگام برخورد نارنجک به سطوح یا به زمین اجرا شود، این کلاس دقیقا مانند کلاس `CParticle01`

است اما به جای لود و نمایش فایل پارتیکل `fire.pfx` پارتیکلی با نام مثلا `dust.pfx` باید جایگزین شود، شما مانند آنچه که برای پارتیکل `CParticle۰۱` ایجاد کردید برای کلاس مثلا `Dust۰۱` ایجاد کنید که شامل دو فایل هدر و سی پی پی با نام های `Dust۰۱.h` و `Dust۰۱.cpp` باشد و خط زیر را اینگونه تعریف کنید :

```
pEntityDust->CreateComponentClass<CDust۰۱>();
```

```
bool Once=true;
```

این متغیر برای بررسی وضعیت انفجار و فقط برای یکبار کدها را اجرا خواهد کرد.

```
int repeatCollisionCount=۰;
```

این متغیر که از نوع صحیح است، هنگامی که نارنجک پرت می شود، این تضمین را به وجود می آورد که فقط یکبار گردو خاکی به صورت پارتیکل در هنگام برخورد نارنجک با سطوح ایجاد شود

```
////////////////////////////////////  
////////
```

```
// C++ Code by Ahmad Karami ۲۰۱۸
```

```
////////////////////////////////////  
////////
```

```
#include "StdAfx.h"
```

```
#include "Grenade.h"
```

```
#include "Particle۰۱.h"
```

```
#include <CrySchematyc/Env/IEnvRegistrar.h>
```

```
#include
```

```
<CrySchematyc/Env/Elements/EnvComponent.h>
```



```
#include <CryParticleSystem\IParticlesPfxr.h>
#include <CryParticleSystem\IParticles.h>
#include <CryEntitySystem\IEntity.h>
#include <CryEntitySystem\IEntityComponent.h>
#include <CryEntitySystem\IEntityBasicTypes.h>
#include <CryEntitySystem\IEntitySerialize.h>
```

```
static void
RegisterGrenadeComponent(Schematyc::IEnvRegistrar& registrar)
{
    Schematyc::CEnvRegistrationScope scope =
    registrar.Scope(IEntity::GetEntityScopeGUID())
    ;
    {
        Schematyc::CEnvRegistrationScope
        componentScope =
        scope.Register(SCHEMATYC_MAKE_ENV_COMPONENT(CGrenadeComponent));
        // Functions
        {
        }
    }
}
```

```
CRY_STATIC_AUTO_REGISTER_FUNCTION(&RegisterGrenadeComponent)
```

```
void
CGrenadeComponent::SpawnEntityGrenade(IEntity*
otherEntity)
{
```

```

otherEntity->SetWorldTM(m_pEntity-
>GetWorldTM());
}

void CGrenadeComponent::OnUpdate(float
deltaTime)
{
    sum+=deltaTime;
    if(sum>=۴.f && Once)
    {

        IEntityIt *it = gEnv->pEntitySystem-
        >GetEntityIterator();
        it->MoveFirst();

        while (!it->IsEnd())
        {
            IEntity *pEntity = it->Next();

            float x = abs(pEntity->GetPos().x -
            GetEntity()->GetPos().x)*abs(pEntity-
            >GetPos().x - GetEntity()->GetPos().x);
            float y = abs(pEntity->GetPos().y -
            GetEntity()->GetPos().y)*abs(pEntity-
            >GetPos().y - GetEntity()->GetPos().y);
            float z = abs(pEntity->GetPos().z -
            GetEntity()->GetPos().z)*abs(pEntity-
            >GetPos().z - GetEntity()->GetPos().z);

            float d = sqrt(x + y + z);
            string s= pEntity->GetName();
            CryLog("pEntity->GetName() :"+s);

```

```

CryLog(ToString(strcmpi(pEntity->GetName(),
"Player")));
if (strcmpi(pEntity->GetName(), "Player") != ۰ &&
strcmpi(pEntity->GetName(), "Grenade") != ۰ &&
d <= ۵.f )
{
    SEntitySpawnParams spawnParams۴;
    spawnParams۴.pClass = gEnv->pEntitySystem-
>GetClassRegistry()->GetDefaultClass();
    spawnParams۴.vPosition = pEntity->GetPos();
    const float ParticleScale۴ = ۰.۲۵f;
    spawnParams۴.vScale = Vec۳(ParticleScale۴);

    IEntity* pEntity۴ = gEnv->pEntitySystem-
>SpawnEntity(spawnParams۴);

    pEntity۴->CreateComponentClass<CParticle۰۱>();

    gEnv->pEntitySystem->RemoveEntity(pEntity-
>GetId());

}
if (strcmpi(pEntity->GetName(), "Player") == ۰
&& d <= ۵.f)
    pEntity->SetPos(Vec۳(۶۵.f, ۵۵.f, ۳۳.f));

}

if(it->IsEnd())

```

```
gEnv->pEntitySystem->RemoveEntity(GetEntity()->GetId());
```

```
Once = false;
sum = 0.f;
PublicPlayMySound("bullet_impact");
 IEntity* pParticleEntity=nullptr;
if (IParticleEffect *pEffect = gEnv-
>pParticleManager->FindEffect("fire.pfx",
"Particles"))
{
if (!pParticleEntity)
{
SEntitySpawnParams spawn;
spawn.pClass = gEnv->pEntitySystem-
>GetClassRegistry()->GetDefaultClass();
pParticleEntity = gEnv->pEntitySystem-
>SpawnEntity(spawn);
}
if (pParticleEntity)
{
pParticleEntity->SetPos(GetEntity()-
>GetPos());
pParticleEntity->FreeSlot(.);
pParticleEntity->LoadParticleEmitter(. ,
pEffect);
}
}
}
```

حالا به توضیحات کدهای فایل Greande.cpp می پردازم :

```
#include "StdAfx.h"
```

با استفاده از دستور پیش پردازنده include ، به تابع، توابع و متغیرهای تعریف شده در هدر فایل StdAfx.h می توان دسترسی داشت

```
#include "Grenade.h"
```

با استفاده از دستور پیش پردازنده include ، به تابع، توابع و متغیرهای تعریف شده در هدر فایل Grenade.h می توان دسترسی داشت

```
#include "Particle۰۱.h"
```

با استفاده از دستور پیش پردازنده include ، به تابع، توابع و متغیرهای تعریف شده در هدر فایل Particle۰۱.h می توان دسترسی داشت

بقیه هدر فایل ها نیز از این عملکرد تبعیت می کنند و هر هدر فایل متغیرها و توابع خود را دارد

```
void CGrenadeComponent::OnUpdate(float  
deltaTime)  
{
```

برش های زمانی در تابع OnUpdate در کلاس CGrenadeComponent دریافت می شود

```
sum+=deltaTime;
```

با استفاده از متغیر sum برش های زمانی deltaTime جمع می شوند

```
if(sum>=۴.f && Once)
{
```

اگر جمع برش زمانی (متغیر sum) مساوی ۴ یا بزرگتر از ۴ ثانیه شد (در دنیای واقعی نیز نارنجک بعد از ۴ ثانیه منفجر می شود) و البته یکبار (Once) دستورات زیر بر اساس لیست entity های موجود در مرحله اجرا شود، با استفاده از متغیر Once این تضمین ایجاد می شود که نارنجک فقط یکبار منفجر شود و تنها یکبار لیست entity های موجود در مرحله فعلی (مپ) را دریافت کند.

```
IEntityIt *it = gEnv->pEntitySystem-
>GetEntityIterator();
```

یک متغیر اشاره گر تعریف کردم که از نوع struct در هدر فایل IEntitySystem.h است.

با استفاده از تابع GetEntityIterator در کتابخانه pEntitySystem در ریشه دستورات کل کرای انجین gEnv می توان به کل entity های موجود در مرحله (مپ) دست یافت

```
it->MoveFirst();
```

این خط به اولین entity مرحله اشاره دارد و آدرس اشاره گر متغیر it را به اولین entity داخل مرحله اختصاص می دهد

```
while (!it->IsEnd())
{
```

با استفاده از دستور `while` بررسی می کند که آیا متغیر `it` به پایان لیست رسیده است یا نه؟ در صورتی که به پایان لیست رسیده باشد، دستورات زیر اجرا نمی شوند و در صورتی که به پایان لیست نرسیده باشد دستورات زیر اجرا خواهد شد.

```
IEntity *pEntity = it->Next();
```

متغیر اشاره گر `pEntity` را تعریف کرده ام و این متغیر نیز مانند متغیر `it` از نوع `struct` در هدر فایل `IEntitySystem.h` است و آدرس متغیر `it` بعدی را در خود حمل می کند و در واقع به `entity` بعدی در مرحله فعلی اشاره می کند که آدرس آن نیز در داخل متغیر `it` است.

```
float x = abs(pEntity->GetPos().x -  
GetEntity()->GetPos().x)*abs(pEntity->  
GetPos().x - GetEntity()->GetPos().x);
```

این خط کد طبق فرمول زیر محاسبه می کند، اختلاف برداری یا اختلاف نقطه ای بر حسب محور `x` با نقطه `entity` بعدی در لیست `entity` های مرحله با نقطه `entity` نارنجک چقدر است، داخل متغیر `x` گذاشته می شود

$$x = (|px - gx|) * (|px - gx|)$$

```
float y = abs(pEntity->GetPos().y -  
GetEntity()->GetPos().y)*abs(pEntity->  
GetPos().y - GetEntity()->GetPos().y);
```

این خط کد طبق فرمول زیر محاسبه می کند، اختلاف برداری یا اختلاف نقطه ای بر حسب محور y با نقطه `entity` بعدی در لیست `entity` های مرحله با نقطه `entity` نارنجک چقدر است، داخل متغیر y گذاشته می شود

$$y = (|py - gy|) * (|py - gy|)$$

```
float z = abs(pEntity->GetPos().z -
GetEntity()->GetPos().z)*abs(pEntity-
>GetPos().z - GetEntity()->GetPos().z);
```

این خط کد طبق فرمول زیر محاسبه می کند، اختلاف برداری یا اختلاف نقطه ای بر حسب محور z با نقطه `entity` بعدی در لیست `entity` های مرحله با نقطه `entity` نارنجک چقدر است، داخل متغیر z گذاشته می شود

$$z = (|pz - gz|) * (|pz - gz|)$$

```
float d = sqrt(x + y + z);
```

این فرمول فاصله بین دو مختصات $p(x,y,z)$ و $g(x,y,z)$ است که نقطه p منظور مختصاتی است که `pEntity` در آن وجود دارد (یکی از `entity` هایی که در لیست `entity` های مرحله موجود است) و نقطه g منظور مختصاتی است که `GetEntity` در آن موجود است، نقطه g به نقطه نارنجک اشاره دارد.

$$d = \sqrt{x + y + z}$$


```
string s= pEntity->GetName();
```

یک متغیر از نوع `string` تعریف کرده ام که اسم `entity` بعدی از لیست `entity` ها را در خود ذخیره می کند

```
CryLog("pEntity->GetName() :"+s);
```

با این خط کد، نام `entity` در لیست `entity` های موجود در مرحله در پنجره `output` یا `console` نمایش داده می شود (صرفاً جهت اطلاع برای برنامه نویس)

```
CryLog(ToString(strcmpi(pEntity->GetName(),  
"Player")));
```

این خط کد نیز در پنجره `output` یا `console` پیغامی را نمایش می دهد که آیا `entity` موجود در لیست پلیمر است یا نه؟ (صرفاً جهت اطلاع برای برنامه نویس)

```
if (strcmpi(pEntity->GetName(),"Player")!=0 &&  
strcmpi(pEntity->GetName(), "Grenade") != 0 &&  
d <= ۵.f )  
{
```

این شرط بررسی می کند که از لیست `entity` های موجود در مرحله :

۱- اگر `entity` جاری مخالف `Player` باشد

۲- اگر `entity` جاری مخالف `Grenade` باشد

۳- اگر فاصله entity نارنجک از entity جاری کمتر یا مساوی ۵ باشد مراحل زیر اجرا شود :

```
SEntitySpawnParams spawnParams۴;
spawnParams۴.pClass = gEnv->pEntitySystem-
>GetClassRegistry()->GetDefaultClass();
spawnParams۴.vPosition = pEntity->GetPos();
const float ParticleScale۴ = ۰,۲۵f;
spawnParams۴.vScale = Vec۳(ParticleScale۴);
IEntity* pEntity۴ = gEnv->pEntitySystem-
>SpawnEntity(spawnParams۴);
pEntity۴->CreateComponentClass<CParticle۰۱>();
```

۷ خطی که به ؛ ختم می شود را در چند صفحه قبل توضیح دادم که عمل تکثیر یک کلاس با نام CParticle۰۱ را انجام می دهد

```
gEnv->pEntitySystem->RemoveEntity(pEntity-
>GetId());
```

این خط به ما می گوید که entity که الان حذف می شود entity هایی است که می توان آنها را حذف کرد و جزء entity هایی هستند که دارای فیزیک بوده و عمل حذف شدن امکان پذیر است، این خط به ما می گوید که entity که حذف می شود نه نارنجک است و نه پلیر است، به خط زیر دقت کنید

```
}
```

```
if (strcmpi(pEntity->GetName(), "Player") == 0 && d <= ۵.f)
```

این شرط بررسی می کند که فاصله نارنجک از entity جاری در لیست entity های مرحله کمتر یا مساوی ۵ باشد و entity جاری برابر Player بود، همان entity جاری در لیست entity های مرحله (به موقعیت فضای سه بعدی $x=۶۵$ ، $y=۵۵$ ، $z=۳۳$ منتقل شود، یعنی اگر نارنجک در فاصله کمتر یا مساوی ۵ متر با پلیمر بود، پلیمر به مختصات $x=۶۵$ ، $y=۵۵$ ، $z=۳۳$ منتقل شود

```
pEntity->SetPos(Vec3(۶۵.f, ۵۵.f, ۳۳.f));
}
```

```
if(it->IsEnd())
```

این شرط بررسی میکند که اگر اشاره گر `it` به پایان لیست entity های مرحله رسید باید entity نارنجک نیز حذف شود

```
gEnv->pEntitySystem->RemoveEntity(GetEntity()->GetId());
```

این خط باعث می شود که آخر entity یعنی نارنجک نیز در مرحله حذف شود (چون نارنجک بعد از منفجر شدن متلاشی می شود)

```
Once = false;
sum = ۰.f;
```

متغیر های `Once` و `sum` نیز قبلا عملکردشان توضیح داده شده است

```
PublicPlayMySound("bullet_impact");
```

این تابع در هدر فایل StdAfx.h تعریف کرده ام و برای پخش کردن آهنگ، موسیقی یا صوت استفاده می شود، اسم تریگر صوتی Audio Controls Editor در پنجره bullet_impact باید موجود باشد و در فصل های قبلی در رابطه با وارد کردن فایل های mp3 ، wav ، ogg و به کرای انجین و این پنجره در این کتاب توضیحات مفیدی بیان کرده ام

```
static void PublicPlayMySound(string
TriggerName)
{
```

این تابع از نوع استاتیک (static) است، به این معنا که مستقل از entity عمل می کند، یعنی لازم نیست یک شی (entity) از کلاس (هدر فایل و سی پی پی فایل) ایجاد کنید و سپس از طریق شی به تابع دسترسی داشته باشید، شما در هر جای پروژه می توانید مستقیماً تابع PublicPlayMySound را فراخوانی کنید، این تابع پارامتر مقداری از نوع string را میگیرد و اسم پارامتر باید نام یک تریگر صوتی (TriggerName) باشد

```
CryAudio::ControlId const MyTriggerId =
CryAudio::StringToId(TriggerName);
```

با توجه به فضای نام CryAudio و استفاده از ControlId متغیر ثابتی را با نام MyTriggerId تعریف کرده ام که رشته یا نام یک تریگر صوتی را به یک Id تریگری تبدیل می کند.

```
gEnv->pAudioSystem->LoadTrigger(MyTriggerId);
```

عمل تریگر شدن برای لود متغیر انجام می شود

```
gEnv->pAudioSystem-  
>ExecuteTrigger(MyTriggerId);
```

بعد از لود شدن تریگر، صدای صوت، موسیقی یا موزیک پخش خواهد شد و در بازی شنیده خواهد شد

```
}
```

```
gEnv->pAudioSystem-  
>UnloadTrigger(MyTriggerId);
```

برای عمل خارج کردن و خالی کردن تریگر صوتی از حافظه از این دستور استفاده می شود

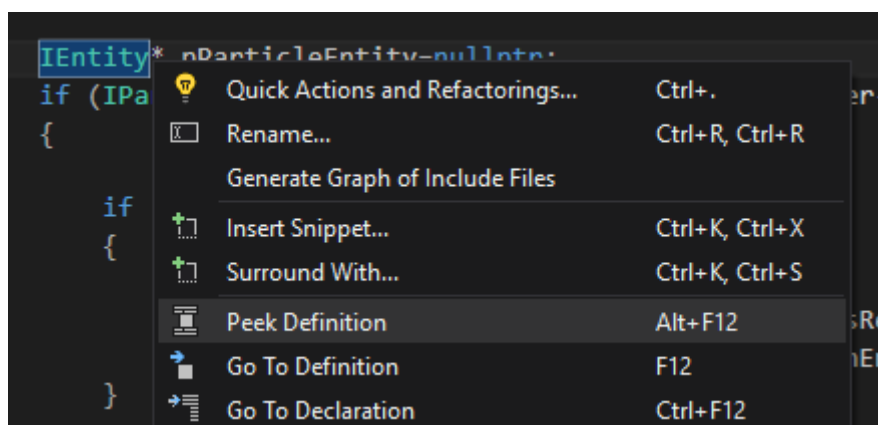
```
gEnv->pAudioSystem->StopTrigger(MyTriggerId);
```

لازم به ذکر است که برای توقف تریگر صوتی و عدم پخش صوت، موسیقی یا موزیک نیز از این دستور استفاده می شود

```
IEntity* pParticleEntity=nullptr;
```

یک متغیر با نام pParticleEntity طبق نوع ساختمان داده struct در هدر فایل IEntity.h را تعریف کرده ام

برای دیدن جزئیات هر کلمه کلیدی و واژه ای که می خواهید بدانید چگونه در کتابخانه های CryEngine یا در کتابخانه های C++ تعریف شده است، کافیت بر روی کلمه کلیک کنید تا کادر آبی کلمه انتخابی را احاطه کند و سپس رو همان کلمه راست کلیک کرده و گزینه Peek Definition را انتخاب کنید یا بر روی کلمه کلیک کنید تا کادر آبی کلمه انتخابی را احاطه کند و دکمه های ترکیبی Alt + F12 بر روی صفحه کلید را فشار دهید



در اینجا من کلمه IEntity را انتخاب کرده ام و حالا طبق یکی از روش های بالا عمل میکنم و می بینید که کادر دیگری باز می شود و هدر فایل IEntity.h را برای من نشان خواهد داد و نحوه چگونگی تعریف و پیاده سازی کلمه انتخابی IEntity را در هدر فایل IEntity.h نیز در تصویر زیر می بینید :

```

IEntity* pParticleEntity=NULLPTR;

//!! Interface to entity object.
struct IEntity
{
#ifdef SWIG
    //!! This is a GUID of the Schematyc environment scope where all entity components must be registered
    static CryGUID GetEntityScopeGUID()
    {
        static CryGUID guid = "be845278-0dd2-409f-b8be-97895607c256"_cry_guid;
        return guid;
    }
#endif
};

if (IParticleEffect *pEffect = gEnv->pParticleManager->FindEffect("fire.pfx", "Particles"))

```

```

if (IParticleEffect *pEffect = gEnv-
>pParticleManager->FindEffect("fire.pfx",
"Particles"))
{

```

در این خط کد متغیر اشاره گری را تعریف کرده ام با نام `pEffect` که یک پارتيکل سیستم با نام `fire.pfx` را پیدا می کند و طبق پیدا شدن این پارتيکل بلوک `if` اجرا خواهد شد و در صورتی که این پارتيکل سیستم پیدا نشد دستورات زیر در بلوک `if` را اجرا نخواهد کرد

```

if (!pParticleEntity)
{

```

طبق شرطی که در این خط تعریف شده و متغیر `pParticleEntity` تهی است، بلوک `if` درونی اجرا خواهد شد

```
SEntitySpawnParams spawn;
```

این متغیر با نوع داده ای که دارد عمل تکثیر را برعهده میگیرد

```
spawn.pClass = gEnv->pEntitySystem-
>GetClassRegistry()->GetDefaultClass();
```

فیلد `pClass` مقدار پیش فرض کلاسی را باید دریافت کند تا عمل تکثیر از کلاسی از پارتیکل سیستم را انجام دهد

```
pParticleEntity = gEnv->pEntitySystem-
>SpawnEntity(spawn);
```

این خط عمل تکثیر کلاس را به متغیر `pParticleEntity` انجام می دهد و باید عمل تکثیری را براساس آدرس این متغیر انجام دهد

```
}
```

```
if (pParticleEntity)
{
```

از آنجایی که متغیر `pParticleEntity` حالا دارای کلاس پیش فرضی از عمل تکثیر را در خود حمل می کند و خالی نیست، بلوک `if` اجرا می شود

```
pParticleEntity->SetPos(GetEntity()-
>GetPos());
```

موقعیت کلاس تولید شده و تکثیر شده از متغیر `pParticleEntity` در فضای سه بعدی `x,y,z` به موقعیت `entity` جاری (`GetEntity()->GetPos()`) تغییر خواهد کرد و به موقعیت جدید خواهد رفت (`SetPos`)

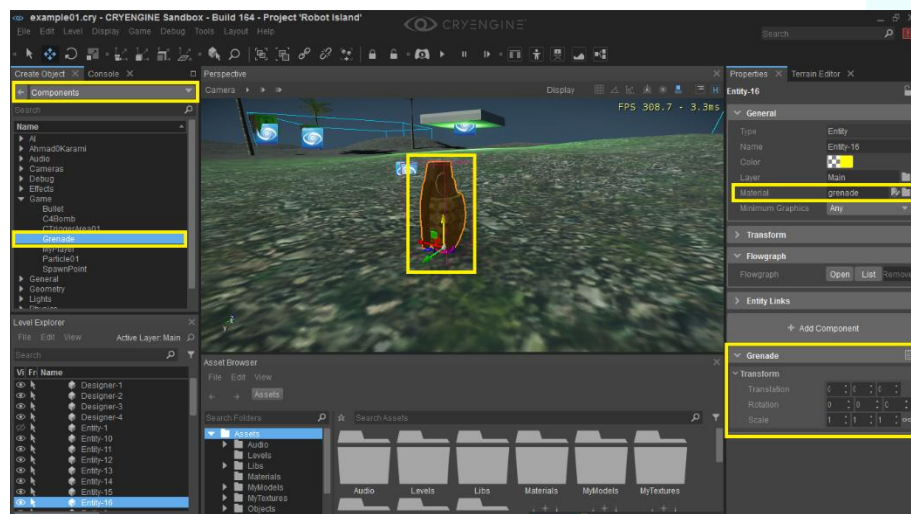

```
pParticleEntity->LoadParticleEmitter(.,
pEffect);
```

کلاس تکثیر شده آماده دریافت لود مقداری است که در اینجا این مقدار به صورت پارتيكل آدرس دهی می شود و آدرس فایل پارتيكل در متغير pEffect موجود است و در slot۰ پارتيكل سیستم در کلاس تکثیر شده و تبدیل شده به entity جاری متغير pParticleEntity لود خواهد شد

```
}
}

}
```

فراموش نباید شود که این دو فایل (Grenade.h و Grenade.cpp) در حتما در CMakeLists.txt باید ثبت کنید، نحوه ثبت کدهای *.cpp و *.h را در فصل قبلی توضیح داده ام



کلاسی که به ما کمک می کند تا پلیر را در یک فضای سه بعدی قرار دهیم و نقطه شروع بازی را در مرحله (مپ) تعیین کنیم کلاس **SpawnPoint** است و در مسیر زیر قرار دارد :

CRYENGINE_۵,۴\Templates\cpp

هدر فایل **SpawnPoint.h** شامل کدهای زیر است، اگر دقت کنید این هدر فایل معادل همان هدر فایل **MyBoxEntity.h** است و تنها برخی کلمات است که تغییر یافته اند و عملکرد فایل **SpawnPoint.h** و فایل **MyBoxEntity** کاملاً یکسان است و در فصل قبلی هدر فایل **MyBoxEntity.h** رو توضیح دادم، پس از توضیحات مجدد این هدر فایل و هدر فایل **SpawnPoint.h** صرف نظر میکنم، هدر فایل **SpawnPoint.h** برای تعیین نقطه آغاز بازی استفاده می شود و پلیر در آن نقطه بر اساس کلاس **SpawnPoint** شروع به بازی می کند و کلاس **SpawnPoint** در کلاس پلیر استفاده می شود و در فصل بعدی آن را توضیح خواهم داد

```
#pragma once
```

```
////////////////////////////////////
////////////////////////////////////
// Spawn point
////////////////////////////////////
////////////////////////////////////
class CSpawnPointComponent final : public
IEntityComponent
{
public:
CSpawnPointComponent() = default;
virtual ~CSpawnPointComponent() {}

// Reflect type to set a unique identifier for
this component
// and provide additional information to
expose it in the sandbox
static void
ReflectType(Schematyc::CTypeDesc<CSpawnPointCo
mponent>& desc)
{
desc.SetGUID("{۴۱۳۱۶۱۳۲-۸A۱E-۴۰۷۳-B۰CD-
A۲۴۲FD۲D۲E۹۰}"_cry_guid);
desc.SetEditorCategory("Game");
desc.SetLabel("SpawnPoint");
desc.SetDescription("This spawn point can be
used to spawn entities");
```

```

desc.SetComponentFlags({
    IEntityComponent::EFlags::Transform,
    IEntityComponent::EFlags::Socket,
    IEntityComponent::EFlags::Attach });
}

public:
void SpawnEntity(IEntity* otherEntity);
};

```

فایل SpawnPoint.cpp نیز همانند فایل MyBoxEntity.cpp است با همان ساختار فقط بعضی از کلمات است که تغییر یافته اند اما عملکردشان کاملاً یکسان است و در اینجا من از توضیحات مجدد فایل SpawnPoint.cpp صرف نظر میکنم

```

#include "StdAfx.h"
#include "SpawnPoint.h"

#include <CrySchematyc/Env/IEnvRegistrar.h>
#include
<CrySchematyc/Env/Elements/EnvComponent.h>

static void
RegisterSpawnPointComponent(Schematyc::IEnvReg
istrar& registrar)
{
    Schematyc::CEnvRegistrationScope scope =
    registrar.Scope(IEntity::GetEntityScopeGUID())
;
{

```

```

Schematyc::CEnvRegistrationScope
componentScope =
scope.Register(SCHEMATYC_MAKE_ENV_COMPONENTEN
T(CSpawnPointComponent));
// Functions
{
}
}
}

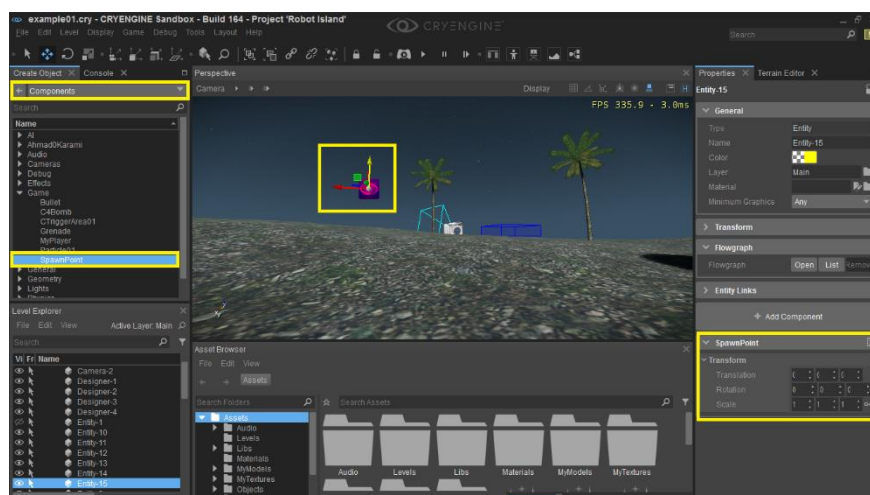
CRY_STATIC_AUTO_REGISTER_FUNCTION(&Register
SpawnPointComponent)

void
CSpawnPointComponent::SpawnEntity(IEntity*
otherEntity)
{
otherEntity->SetWorldTM(m_pEntity-
>GetWorldTM());
}

```

موجودیت (entity) SpawnPoint در پلیمر موقعیت نقاط x , y , z را تعیین می کند که پلیمر در هر مکانی باشد عمل آغاز شدن بازی در مکانی که entity مربوط به کلاس SpawnPoint است، بازی در آن مکان شروع می شود

فراموش نباید شود که این دو فایل (SpawnPoint.h و SpawnPoint.cpp) در حتما در CMakeLists.txt باید ثبت کنید، نحوه ثبت کدهای *.cpp و *.h را در فصل قبلی توضیح داده ام



اگر جزء کسانی هستید که عاشق بازی های جنگی و نبردهای دشوار چریکی هستید، باید به شما بگویم که حتما باید بدانید که به عنوان برنامه نویس چگونه بمب های انفجاری با کنترل از راه دور یا بمب های تایمری را ایجاد کنید، این بمب ها با نام C۴ شناخته می شوند و در اینجا سورس کد مربوط به بمب C۴ را می بینید که جزء بمب های C۴ تایمری (پروتوتایپ یا نمونه اولیه) است، اگر دقت کنید بخش زیادی از این کدها مربوط به کدهای پیاده سازی شده در مدل بمب ها از نوع نارنجک است، من در اینجا از توضیحات مجدد صرف نظر میکنم اما عملکرد بمب های C۴ تایمری این است که بعد از چند ثانیه منفجر می شوند و شعاع مشخصی مثلا تا ۱۰ متر را منهدم می کند، ابتدا هدر فایل را ایجاد و سپس سی پی پی فایل را ایجاد کنید و این دو فایل را با نام های C۴Bomb.h و C۴Bomb.cpp در مسیر پروژه RobotIsland\Code\Components کی-پیست کنید.

ابتدا به توضیح کدهای C۴Bomb.h می پردازم

#pragma once

```

////////////////////////////////////
////////////////////////////////
// Physicalized bullet shot from weaponry,
// expires on collision with another object
////////////////////////////////
////////////////////////////////
class CCfBombComponent final : public
IEntityComponent
{
public:
virtual ~CCfBombComponent() {}

// IEntityComponent
virtual void Initialize() override
{
// Set the model
const int geometrySlot = .;
m_pEntity->
>LoadGeometry(geometrySlot, "MyModels/CfBomb/CfB
omb.cgf");

auto *pGrenadeMaterial = gEnv->pEngine->
>GetMaterialManager()-
>LoadMaterial("MyModels/CfBomb/cfbomb");

m_pEntity->SetMaterial(pGrenadeMaterial);

GetEntity()->SetViewDistRatio(۲۵۵);

```

```

GetEntity()->SetRotation(Quat(Vec3(۳۶۰ *
hit.n.x, ۳۶۰ * hit.n.y, ۳۶۰*hit.n.z)));

}

static void
ReflectType(Schematyc::CtypeDesc<CCfBombCompon
ent>& desc)
{
desc.SetGUID("{۰F۳۳F۲۵C-۴۳B۸-۴B۲C-۹۲A۱-
۵۶۶۵A۲۸۶B۸۹D}"_cry_guid);
desc.SetEditorCategory("Game");
desc.SetLabel("CfBomb");
desc.SetDescription("This CfBomb can be used
to spawn entities");
desc.SetComponentFlags({
IdentityComponent::Eflags::Transform,
IdentityComponent::Eflags::Socket,
IdentityComponent::Eflags::Attach });
}

virtual uint64 GetEventMask() const override {
return BIT64(ENTITY_EVENT_COLLISION) |
BIT64(ENTITY_EVENT_UPDATE); }
virtual void ProcessEvent(SentityEvent& event)
override
{
if (event.event == ENTITY_EVENT_UPDATE)
{

```



```

SentityUpdateContext* pCtx =
(SentityUpdateContext*)event.nParam[.];

OnUpdate(pCtx->fFrameTime);
}
}
// ~IentityComponent

```

```

public:
void SpawnEntityC۴(Ientity* otherEntity);
void OnUpdate(float deltaTime);
float sum=۰.f;
bool Once=true;
};

```

سعی کنید از مدل های سه بعدی C۴ کروی استفاده کنید تا کد زیر بر روی مدل سه بعدی کروی اعمال شود، بیشتر C۴ ها به صورت مدل های سه بعدی مربعی یا مستطیلی است :

```

GetEntity()->SetRotation(Quat(Vec۳(۳۶۰ *
hit.n.x, ۳۶۰ * hit.n.y, ۳۶۰*hit.n.z)));

```

این خط کد براساس بردار نرمال (hit.n) با توجه به نقطه برخورد Raycast عمل چرخش را بر روی entity جاری یا همان مدل سه بعدی C۴ صورت میگیرد و تابع SetRotation بر اساس مدل چرخش Quat را محاسبه می کند و تبدیلات ضرب برداری انجام می شود و در نهایت عمل چرخش براساس بردار نرمال ثبت می شود، من روی این بخش هنوز کار نکردم اما نمونه اولیه C۴ خوب به نظر می رسد، نکته ای که مهم

است، C۴ تان دارای جاذبه نیست و همچنین عمل Attach کردن به مدل هدف برای تخریب شدن را انجام نمی دهد، در واقع عمل تکثیر C۴ توسط پلیر انجام می شود و تنها یک عمل Instantiate یا تکثیر اتفاق می افتد، مانند آنچه که در entity نارنجک وجود دارد، یک تابع غیر سیستمی با نام OnUpdate نیز در entity C۴ وجود دارد که deltaTime را به داخل تابع ارسال می کند.

کدهایی زیر در فایل C۴Bomb.cpp در entity نارنجک کاملاً یکسان است و از تکرار توضیحات مجدد آن پرهیز میکنم، عملکرد و شباهت این کدها و این entity ها بسیار شبیه هم هستند و می دانم حالا وحشت گذشته از C++ CryEngine را ندارید.

نکته ای که نباید فراموش کنید این است که اگر هدر فایل *.h تعریف می کنید، سعی کنید حتماً *.cpp فایل آن را نیز تعریف نماید زیرا که فایل های *.cpp به شما کمک می کنند تا component ها یا entity ها در پنجره Schematyc و پنجره CreateObject برای شما قابل دسترسی باشند، این کار به شما کمک می کند که پریفب های مختلف را طراحی کنید و بهتر و سریعتر بازی هایتان را بسازید، دقت کنید از آنجایی که شما هدر فایل C۴Bomb.h را ساخته اید و حالا قصد استفاده از متغیرها و توابع هدر فایل C۴Bomb.h در C۴.Bomb.cpp را دارید باید با استفاده از دستور پیش پردازنده include آن را تعریف کنید، مثل آنچه که در مثال زیر و در فایل C۴Bomb.cpp می بینید :

```
#include "StdAfx.h"
#include "C۴Bomb.h"
#include "Particle۰۱.h"
#include <CrySchematyc/Env/IEnvRegistrar.h>
```

```

#include
<CrySchematyc/Env/Elements/EnvComponent.h>
#include <CryParticleSystem\IParticlesPfxr.h>
#include <CryParticleSystem\IParticles.h>

#include <CryEntitySystem\IEntitySystem.h>

static void
RegisterCfBombComponent(Schematyc::IEnvRegistrar& registrar)
{
    Schematyc::CEnvRegistrationScope scope =
    registrar.Scope(IEntity::GetEntityScopeGUID())
    ;
    {
        Schematyc::CEnvRegistrationScope
        componentScope =
        scope.Register(SCHEMATYC_MAKE_ENV_COMPONENT(CC
        fBombComponent));
        // Functions
        {
        }
        }
    }

CRY_STATIC_AUTO_REGISTER_FUNCTION(&RegisterCfBombComponent)

void CCfBombComponent::SpawnEntityCf(IEntity*
otherEntity)
{

```

```
otherEntity->SetWorldTM(m_pEntity-  
>GetWorldTM());  
}
```

```
void CCBombComponent::OnUpdate(float  
deltaTime)  
{
```

```
sum+=deltaTime;  
if(sum>=5.f && Once)  
{
```

```
 IEntityIt *it = gEnv->pEntitySystem-  
>GetEntityIterator();  
it->MoveFirst();
```

```
while (!it->IsEnd())  
{  
 IEntity *pEntity = it->Next();
```

```
float x = abs(pEntity->GetPos().x -  
GetEntity()->GetPos().x)*abs(pEntity-  
>GetPos().x - GetEntity()->GetPos().x);  
float y = abs(pEntity->GetPos().y -  
GetEntity()->GetPos().y)*abs(pEntity-  
>GetPos().y - GetEntity()->GetPos().y);  
float z = abs(pEntity->GetPos().z -  
GetEntity()->GetPos().z)*abs(pEntity-  
>GetPos().z - GetEntity()->GetPos().z);
```

```
float d = sqrt(x + y + z);
```

```

string s= pEntity->GetName();
CryLog("pEntity->GetName() :"+s);
CryLog(ToString(strcmpi(pEntity->GetName(),
"Player")));
if (strcmpi(pEntity->GetName(),"Player")!=0 &&
d <= ۵.f )
{
    SEntitySpawnParams spawnParams;
    spawnParams.pClass = gEnv->pEntitySystem-
>GetClassRegistry()->GetDefaultClass();
    spawnParams.vPosition = pEntity->GetPos();

    const float ParticleScale = ۰.۲۵f;
    spawnParams.vScale = Vec3(ParticleScale);

    IEntity* pEntity = gEnv->pEntitySystem-
>SpawnEntity(spawnParams);

    pEntity->CreateComponentClass<CParticle>();

    gEnv->pEntitySystem->RemoveEntity(pEntity-
>GetId());
}

if (strcmpi(pEntity->GetName(), "Player") == 0
&& d <= ۵.f)
    pEntity->SetPos(Vec3(۶۵.f, ۵۵.f, ۳۳.f));
}

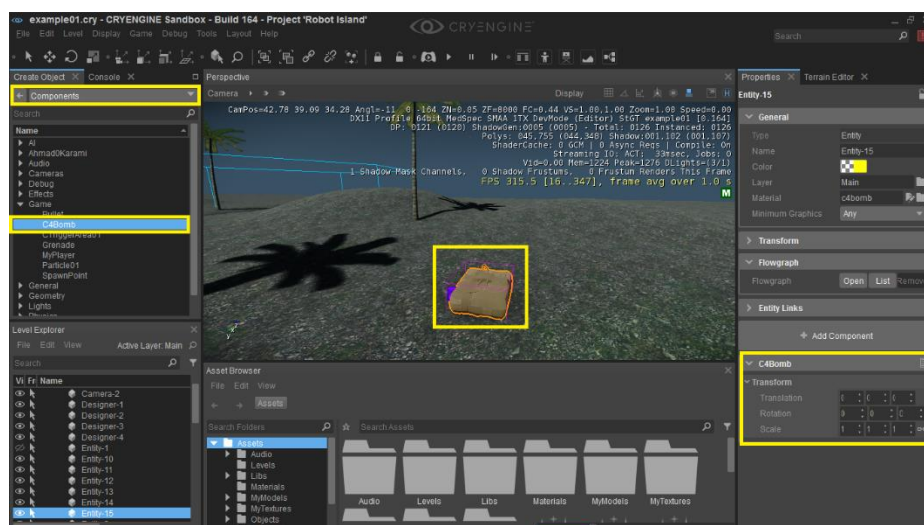
```

```

Once = false;
sum = 0.f;
PublicPlayMySound("bullet_impact");
IEntity* pParticleEntity=NULLptr;
if (IParticleEffect *pEffect = gEnv-
>pParticleManager->FindEffect("fire.pfx",
"Particles"))
{
if (!pParticleEntity)
{
SEntitySpawnParams spawn;
spawn.pClass = gEnv->pEntitySystem-
>GetClassRegistry()->GetDefaultClass();
pParticleEntity = gEnv->pEntitySystem-
>SpawnEntity(spawn);
}
if (pParticleEntity)
{
pParticleEntity->SetPos(GetEntity()-
>GetPos());
pParticleEntity->FreeSlot(.);
pParticleEntity->LoadParticleEmitter(.,
pEffect);
}
}
}
}

```

فراموش نباید شود که این دو فایل (`CfBomb.h` و `CfBomb.cpp`) در حتما در `CMakeLists.txt` باید ثبت کنید، نحوه ثبت کدهای `*.h` و `*.cpp` را در فصل قبلی توضیح داده ام



حالا بیایم مثال متفاوتی را پیاده کنیم و نقش `TriggerEnter` و `TriggerExit` را در بازی برای شما توضیح دهیم، این یکی از مهمترین بخش ها در ساخت بازی های ویدئویی است، تریگر به معنی راه انداختن کاری است، مثلا وقتی که به دری نزدیک می شویم باید در را بتوان باز یا بسته کرد، یا به کلید لامپی نزدیک می شویم باید بتوان لامپ را خاموش و روشن کرد و در این مثال جعبه مهمات را پیاده می کنیم که وقتی پلیمر به جعبه مهمات نزدیک شود، خشاب اسلحه ها در پلیمر پر شود یا مقداری به آن اضافه شود، البته با استفاده از سیستم پرداخت درون برنامه `crycash` می توان در

بازی به پلیر این اختیار را داد که خشاب ها را بتواند خریداری کند، در این مثال به سیستم crycash نمی پردازم.

ابتدا هدر فایل را ایجاد و سپس سی پی پی فایل را ایجاد خواهید کرد و این دو فایل را با نام های TriggerArea۰۱.h و TriggerArea۰۱.cpp در مسیر پروژه RobotIsland\Code\Components کپی-پیست کنید، حالا داخل هدر فایل TriggerArea۰۱.h کد زیر را تایپ کنید، بخش های دیگر این کدها تکراری هستند اما به توابع سیستمی Initialize ، GetEventMask ، ProcessEvent نگاهی دقیق خواهیم انداخت زیرا که مباحث جدیدی را در خود دارند و حتما برای شما نیز جذاب خواهند بود و ابتدا به بررسی TriggerArea۰۱.h می پردازم

```
#pragma once
#include <CryEntitySystem/IEntityComponent.h>
#include <CryEntitySystem/IEntity.h>

////////////////////////////////////
////////////////////////////////////
// Ammo Trigger Box!
////////////////////////////////////
////////////////////////////////////

class CtriggerArea۰۱ final : public
IEntityComponent
{
public:
CtriggerArea۰۱() = default;
virtual ~CtriggerArea۰۱() {}
```



```

virtual void Initialize() override
{
    // Create a new IEntityTriggerComponent
    instance, responsible for registering our
    entity in the proximity grid
    IEntityTriggerComponent* pTriggerComponent =
    m_pEntity-
    >CreateComponent<IEntityTriggerComponent>();
    // Listen to area events in a ۲m^۳ box around
    the entity
    const Vec3 triggerBoxSize = Vec3(۲, ۲, ۲);
    // Create an axis aligned bounding box,
    ensuring that we listen to events around the
    entity translation
    const AABB triggerBounds = AABB(triggerBoxSize
    * -.۵f, triggerBoxSize * .۵f);
    // Now set the trigger bounds on the trigger
    component
    pTriggerComponent-
    >SetTriggerBounds(triggerBounds);

    const int geometrySlot = .;
    GetEntity()->LoadGeometry(geometrySlot,
    "Objects/Default/primitive_cube_small.cgf");
    auto *pGrenadeMaterial = gEnv->pEngine-
    >GetMaterialManager()-
    >LoadMaterial("Materials/grenade");
    GetEntity()->SetMaterial(pGrenadeMaterial);
}

```

```

static void
ReflectType(Schematyc::CtypeDesc<CtriggerArea>
>& desc)
{
desc.SetGUID("{0D۴E۲۵۹C-CBYC-۴F۰۷-۹۴D۵-
۹AFF۴E۷۱۸۶F۸}"_cry_guid);
desc.SetEditorCategory("Game");
desc.SetLabel("CtriggerArea");
desc.SetDescription("This CtriggerArea can be
used to Trigger entities , Ahmad Karami ha ha
ha 😊");
desc.SetComponentFlags({
IdentityComponent::Eflags::Transform,
IdentityComponent::Eflags::Socket,
IdentityComponent::Eflags::Attach });
}

virtual void ProcessEvent(SentityEvent& event)
override {

if (event.event == ENTITY_EVENT_UPDATE)
{
SentityUpdateContext* pcx =
(SentityUpdateContext*)event.nParam[.];

OnUpdate(pcx->fFrameTime);

}

if (event.event == ENTITY_EVENT_ENTERAREA) {

```

```
// Get the entity identifier of the entity
that just entered our shape const
```

```
EntityId enteredEntityId =
static_cast<EntityId>(event.nParam[.]);
```

```
if (gEnv->pEntitySystem-
>FindEntityByName("Player")->GetId() ==
enteredEntityId)
{
HKeyAmmo += ۵۰;
```

```
gEnv->pEntitySystem-
>RemoveEntity(GetEntityId());
}
```

```
}
else if (event.event ==
ENTITY_EVENT_LEAVEAREA) {
// Get the entity identifier of the entity
that just left our shape const
EntityId leftEntityId =
static_cast<EntityId>(event.nParam[.]);
```

```
if (gEnv->pEntitySystem-
>FindEntityByName("Player")->GetId() ==
leftEntityId )
{

}
}
```

}

```
virtual uint64 GetEventMask() const override {
    // Listen to the enter and leave events, in
    // order to receive callbacks above when entities
    // enter our trigger box
    return BIT64(ENTITY_EVENT_ENTERAREA) |
        BIT64(ENTITY_EVENT_LEAVEAREA) |
        BIT64(ENTITY_EVENT_UPDATE);
}
```

```
public:
void SpawnEntityTriggerArea·\ (Ientity*
otherEntity);
void OnUpdate(float deltaTime);
float sum=۰;
int r;
};
```

این تابع به صورت مجازی و قابل توسعه شده است، کامپونت IEntityTrigger فضایی سه بعدی را تخصیص می دهد که هرگاه به آن فضای سه بعدی نزدیک شویم و به آن وارد شویم عملیاتی اجرا شود و هرگاه از آن خارج شویم عملیاتی دیگر اجرا شود، در واقع این همان تعریف از واژه تریگر است

```
virtual void Initialize() override
{
```

```

IEntityTriggerComponent* pTriggerComponent =
m_pEntity->CreateComponent<IEntityTriggerComponent>();

```

متغیری اشاره گر با نام pTriggerComponent از نوع داده با ساختمان داده IEntityTriggerComponent است، این خط یک کامپونت از نوع تریگر را می سازد

```
const Vec3 triggerBoxSize = Vec3(۲, ۲, ۲);
```

این خط کد متغیری با نام triggerBoxSize با فضای ۸ متر مربع/مکعب به صورت ثابت و بدون تغییر می سازد، از آنجایی که از متغیر ثابت و بدون تغییر باقی خواهد ماند (const)، فضای سه بعدی ۸ متر مربع/مکعب نیز ثابت خواهد بود

```
const AABB triggerBounds = AABB(triggerBoxSize
* -.۵f, triggerBoxSize * .۵f);
```

این خط کد فضای سه بعدی را با ساختار و ساختمان داده AABB با متغیر بدون تغییر و ثابت با نام triggerBounds تعریف می کند و فضای سه بعدی min و فضای سه بعدی max خواهد بود، دامنه این متغیر یک مکعب مربع است که به ابعاد و محدوده (-۱, -۱, -۱) تا min تا max (۱, ۱, ۱) خواهد بود، یعنی فضا عملاً نصف شده است

```

pTriggerComponent->SetTriggerBounds(triggerBounds);

```

این خط کامبونت تریگر را با محدوده و فضای تخصیص داده شده با متغیر `triggerBounds` مقداردهی خواهد کرد

```
const int geometrySlot = ۰;
```

این خط متغیر با نام `geometrySlot` را صفر می کند و مقدار متغیر تغییر نخواهد کرد

```
GetEntity()->LoadGeometry(geometrySlot,  
"Objects/Default/primitive_cube_small.cgf");
```

این خط یک مدل سه بعدی مکعب را لود می کند و آن را در `entity` جاری قرار خواهد داد

```
auto *pGrenadeMaterial = gEnv->p3DEngine->  
>GetMaterialManager()-  
>LoadMaterial("Materials/grenade");
```

این خط متریال از قبل طراحی شده را لود میکند، این متریال مربوط به یک متریال نارنجک است و متریال لود شده در داخل متغیر اشاره گر `pGrenadeMaterial` قرار خواهد گرفت

```
GetEntity()->SetMaterial(pGrenadeMaterial);
```

این خط لود کردن متریال بر روی مدل سه بعدی مکعب را اجرا خواهد کرد و مدل سه بعدی مکعب به جعبه نارنجک تغییر خواهد یافت

```
}
```

```
virtual void ProcessEvent(SEntityEvent& event)
override {
```

```
if (event.event == ENTITY_EVENT_UPDATE)
{
SEntityUpdateContext* pcx =
(SEntityUpdateContext*)event.nParam[.];
```

```
OnUpdate(pcx->fFrameTime);
```

در اینجا می بینید که تابع غیرسیستمی و نوشته شده توسط برنامه نویس با نام `OnUpdate` وجود دارد که برش های زمانی را به داخل تابع ارسال می کند

```
}
```

```
if (event.event == ENTITY_EVENT_ENTERAREA) {
```

این خط کد با توجه به بلوک `if` عمل رویداد سیستم را بررسی می کند که آیا `entity` وجود دارد که به داخل فضا و محدوده تریگر فرو رفته باشد یا به محدوده تریگر وارد شده باشد، اگر این چنین است، خطوط بعدی اجرا خواهد شد

```
EntityId enteredEntityId =
static_cast<EntityId>(event.nParam[.]);
```

در این خط، یک متغیر از نوع ساختمان داده تغییر یافته `typedef` با نام `enteredEntityId` تعریف کرده ام و `Id` مربوط به `entity` که وارد تریگر می شود را برمیگرداند، البته این متغیر یعنی

enteredEntityId باید با Id پلیر یکی باشد تا عملیات افزایش خشاب اسلحه یا اسلحه ها را انجام دهد، اگر بلوک **if** زیر تعریف نشود، هر entity که وارد تریگر شود، افزایش خشاب انجام می شود، مثلاً اگر یک نارنجک به طرف تریگر (همان جعبه مهمات) پرتاب شود و نارنجک به تریگر (جعبه مهمات) وارد شود، افزایش خشاب انجام می شود، من از این عمل جلوگیری کرده ام

```
if (gEnv->pEntitySystem-
>FindEntityByName("Player")->GetId() ==
enteredEntityId)
{
```

با این بلوک **if** زمانی که پلیر وارد تریگر شود، این تضمین ایجاد می شود که Id پلیر با entity Id که وارد تریگر شود مقایسه برابری را انجام دهد، از آنجایی که Id entity که وارد تریگر شده همان Id پلیر است، بلوک **if** اجرا خواهد شد

```
HKeyAmmo += ۵۰;
```

این خط کد ۵۰ واحد به تعداد قبلی نارنجک اضافه می کند، متغیر HKeyAmmo در داخل هدرفایل StdAfx.h تعریف شده است و به صورت استاتیک تعریف کرده ام

```
gEnv->pEntitySystem-
>RemoveEntity(GetEntityId());
```

تابع **GetEntityId** به تریگر یا همان entity جاری اشاره دارد و عمل حذف تریگر توسط تابع **RemoveEntity** در کتابخانه

pEntitySystem در ریشه دستورات gEnv کرای انجین انجام می شود

```

}
}
else if (event.event ==
ENTITY_EVENT_LEAVEAREA) {

EntityId leftEntityId =
static_cast<EntityId>(event.nParam[0]);

```

در این خط، یک متغیر از نوع ساختمان داده تغییر یافته typedef با نام leftEntityId تعریف کرده ام و Id مربوط به entity که از تریگر خارج می شود را برمیگرداند، اینجا شما می توانید کدهایتان را در هنگامی که پلیس یا هر entity که از تریگر خارج می شود را درج کنید، شما می توانید کنترل کنید که تنها دستورات مربوط به پلیس که از تریگر خارج می شود را توسط بلوک if زیر اجرا شود :

```

if (gEnv->pEntitySystem-
>FindEntityByName("Player")->GetId() ==
leftEntityId )
{

}

```

اگر می خواهید که هر entity که از تریگر خارج شود، دستورات تریگر اجرا شود، می توانید بلوک if بالا را حذف کنید 😊

```

}

```

```
}
```

```
virtual uint۶۴ GetEventMask() const override {
```

این تابع سیستمی مانند مکملش `ProcessEvent` در هر `entity` وجود دارد، این تابع به صورت ثابت و با برگرداندن سه ماکرو بیتی عمل مسک گذاری را برای تابع مکمل `ProcessEvent` انجام می دهد.

این تابع سه مقدار بیتی با رویدادهای مشخص زیر را بر میگرداند:

`ENTITY_EVENT_ENTERAREA` : با این رویداد عمل تریگر شدن بر اساس همه `entity` ها انجام می شود، این رویداد بررسی می کند که آیا `entity` وجود دارد که وارد تریگر شده است یا نه؟ هنگامی که هر `entity` وارد تریگر شود، این رویداد فعال خواهد شد و `Id` آن `entity` که به تریگر وارد شده است را برمیگرداند.

`ENTITY_EVENT_LEAVEAREA` : با این رویداد عمل تریگر شدن بر اساس همه `entity` ها انجام می شود، این رویداد بررسی می کند که آیا `entity` وجود دارد که از تریگر خارج شده است یا نه؟ هنگامی که هر `entity` از تریگر خارج شود، این رویداد فعال خواهد شد و `Id` آن `entity` که از تریگر خارج شده است را برمیگرداند.

`ENTITY_EVENT_UPDATE` : در این فصل این رویداد را به صورت کامل توضیح داده ام

```
return BIT۶۴(ENTITY_EVENT_ENTERAREA) |  
BIT۶۴(ENTITY_EVENT_LEAVEAREA) |  
BIT۶۴(ENTITY_EVENT_UPDATE);  
}
```

```
void OnUpdate(float deltaTime);
```

تابع غیر سیستمی که برش ها یا ذره های زمانی را دریافت می کند

```
float sum=0.;
```

یک متغیر از نوع داده اعشاری که مقدار صفر را می گیرد، کار این متغیر جمع ذره ها یا برش های زمانی است تا ثانیه ها را رقم بزند

```
int r;
```

یک متغیر از نوع داده صحیح که مقدار نگرفته است و داخل فایل TriggerArea۰۱.cpp مقدار را دریافت خواهد کرد، کار این متغیر تغییر جهت چرخش جعبه مهمات (نارنجک) است

حالا به بررسی TriggerArea۰۱.cpp می پردازیم، بخشی از کدهای این فایل نیز تکراری است و قبلا توضیح داده شده است و من به بررسی کدهایی جدید می پردازم :

```
#include "StdAfx.h"
```

```
#include "TriggerArea۰۱.h"
```

```
#include <CrySchematyc/Reflection/TypeDesc.h>
```

```
#include <CrySchematyc/Uutils/EnumFlags.h>
```

```
#include <CrySchematyc/Env/IEnvRegistry.h>
```

```
#include <CrySchematyc/Env/IEnvRegistrar.h>
```

```
#include
```

```
<CrySchematyc/Env/Elements/EnvComponent.h>
```

```

#include
<CrySchematyc/Env/Elements/EnvFunction.h>
#include
<CrySchematyc/Env/Elements/EnvSignal.h>
#include <CrySchematyc/ResourceTypes.h>
#include <CrySchematyc/MathTypes.h>
#include <CrySchematyc/Utils/SharedString.h>

static void
RegisterCTriggerArea·\ (Schematyc::IEnvRegistrar
& registrar)
{
    Schematyc::CEnvRegistrationScope scope =
    registrar.Scope(IEntity::GetEntityScopeGUID())
    ;
    {
        Schematyc::CEnvRegistrationScope
        componentScope =
        scope.Register(SCHEMATYC_MAKE_ENV_COMPONENT(CT
        riggerArea·\));
        // Functions
        {
        }
        }
    }

CRY_STATIC_AUTO_REGISTER_FUNCTION(&RegisterCTr
iggerArea·\)

void
CTriggerArea·\::SpawnEntityTriggerArea·\ (IEntit
y* otherEntity)

```

```

otherEntity->SetWorldTM(m_pEntity-
>GetWorldTM());

}

void CTriggerArea::OnUpdate(float deltaTime)
{

sum+=deltaTime;
if (sum >= ۳)
{
sum=۰;
r=rand() % ۲;
}

if(r==۰)
GetEntity()->SetRotation(GetEntity()-
>GetRotation()*Quat::CreateRotationZ(deltaTime
*۲));
else
GetEntity()->SetRotation(GetEntity()-
>GetRotation()*Quat::CreateRotationZ(-
deltaTime * ۲));

}

#include "StdAfx.h"

```

از آنجایی که متغیرهایی هستند که به عنوان متغیرهای اسلحه ها در هدر فایل StdAfx.h تعریف و اعلان شده اند و این متغیرهای اسلحه ها در فایل TriggerArea۰۱.cpp استفاده می شوند پس با استفاده از پیش پردازنده include هدر فایل StdAfx.h را در فایل TriggerArea۰۱.cpp

تعریف کرده ام

```
#include "TriggerArea۰۱.h"
```

از آنجایی که توابع سیستمی و غیرسیستمی و متغیرها و رویدادهای سیستمی در هدر فایل TriggerArea۰۱.h تعریف و اعلان شده است، این فایل به TriggerArea۰۱.h وابسته است، این موضوع در رابطه با همه entity ها که با ++C نوشته می شوند صادق است

```
void CTriggerArea۰۱::OnUpdate(float deltaTime)
```

```
{
    // برش های زمانی به داخل تابع غیر سیستمی OnUpdate در هدر فایل
    // TriggerArea۰۱.h ارسال می شود و در فایل TriggerArea۰۱.cpp
    // یعنی در اینجا به صورت آرگومان دریافت می شوند، پارامتر deltaTime
    // شامل برش های زمانی است و توسط متغیر sum جمع می شود، اگر جمع
    // برش ها یا ذره های زمانی بزرگتر از ۳ یا مساوی ۳ ثانیه شد، دوباره متغیر
    // sum صفر می شود و عمل تغییر جهت انجام می شود
```

```
sum+=deltaTime;
```

```
if (sum >= ۳)
```

```
{
```

```
sum=۰;
```

```
r=rand() % ۲;
```

با استفاده از متغیر r عمل تغییر جهت ذخیره می شود، داخل متغیر r چه می گذرد؟ تابع rand یک عدد تصادفی را بین صفر یا یک برمیگرداند و با

استفاده از % عمل باقی مانده تقسیم بر ۲ انجام می شود، این تغییر جهت در بلوک بعدی if کاربرد دارد.

```
}
```

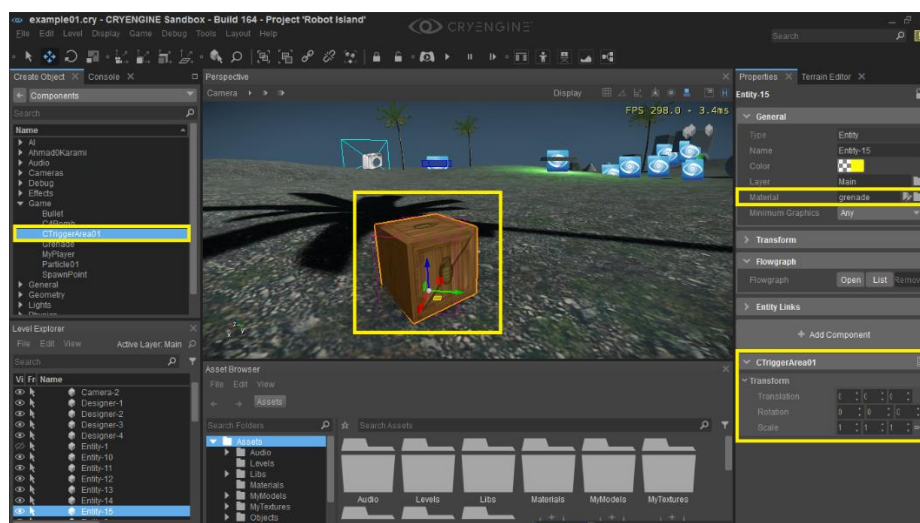
اینجا یک بلوک if-else وجود دارد که بررسی می کند که اگر r برابر صفر بود عمل چرخش جعبه نارنجک یا همان entity جاری به صورت پاد-ساعتگرد باشد و در غیر اینصورت (اگر r مخالف صفر باشد) عمل چرخش ساعتگرد شود

```
if(r==0)
GetEntity()->SetRotation(GetEntity()-
>GetRotation()*Quat::CreateRotationZ(deltaTime
*۲));
else
GetEntity()->SetRotation(GetEntity()-
>GetRotation()*Quat::CreateRotationZ(-
deltaTime * ۲));
```

در آغاز این فصل چرخش entity ها بر اساس محورهای مختصات x و y و z را به صورت کامل تشریح کرده ام و از توضیحات مجدد آن پرهیز میکنم، فقط لازم به ذکر است که عمل ساعتگرد با توجه به علامت منفی در آرگومان تابع CreateRotationZ به صورت ضرب -
 $\Delta t * ۲$ است و در صورتی که چرخش پادساعتگرد باشد، علامت منفی حذف خواهد شد، حالا نتیجه کار را به صورت تصویر ببینید

```
}
```

فراموش نباید شود که این دو فایل (`TriggerArea۰۱.h` و `TriggerArea۰۱.cpp`) را حتما در `CMakeLists.txt` باید ثبت کنید، نحوه ثبت کدهای `*.h` و `*.cpp` را در صفحات قبلی توضیح داده ام



این تریگر تنها برای دریافت نارنجک ها با مقدار ۵۰ عدد بود، شما می توانید با استفاده از هدر فایل `TriggerArea۰۱.h` و فایل `TriggerArea۰۱.cpp` تریگرهای دیگری را برای دریافت انواع مهمات کدنویسی کنید، این یک مثال پایه برای توسعه بازی تان است. نوبت به تحلیل هدر فایل `StdAfx.h` می رسد و چگونگی تعریف متغیرها و توابع استاتیکی که در دیگر کلاس ها قابل استفاده باشد، وقتی کلمه کلیدی `static` (استاتیک) به کار برده می شود، به معنی استفاده از متغیرها، کلاس ها یا توابع مشترک و قابل دسترس در همه کلاس هاست، مثلا مقدار سلامتی پلیمر با متغیر `HealthPlayer` می تواند به صورت استاتیک تعریف شود تا

همه دشمنان به این متغیر دسترسی داشته باشند و میزان سلامتی یا مقدار سلامتی پلیمر را کاهش دهند، در اینجا مثال های خوبی برای استفاده از متغیرها و توابع در هدر فایل StdAfx.h آمده است تا شما درک بهتری از تعریف متغیرهای سراسری اشتراکی با کلمه کلید static داشته باشید، این هدر فایل توسط CMake/Visual Studio ساخته می شوند و بخشی از کدها تولید شده اند و بخش دیگری نیز توسط برنامه نویس نوشته خواهند شد :

```
// Copyright ۲۰۰۱-۲۰۱۶ Crytek GmbH / Crytek
Group. All rights reserved.
```

```
#pragma once
```

```
#include <CryCore/Project/CryModuleDefs.h>
#define eCryModule eCryM_Game
#define GAME_API DLL_EXPORT
```

```
#include <CryCore/Platform/platform.h>
#include <CrySystem/ISystem.h>
#include <Cry3DEngine/I3DEngine.h>
#include <CryNetwork/ISerialize.h>
```

چند خط بالا به صورت پیش فرض ایجاد شده است و شامل تعاریف (define) و برپاسازی API های پایه در کرای انجین (include) و ارتباطات بین سندباکس و ویژوال استودیو (DLL_EXPORT) را به همراه دارد.

```
static Vec3 ahmadval;
```

متغیر استاتیک برداری سه بعدی با نام `ahmadval` تعریف کردم، این متغیر وظیفه ای مهم دارد و جهت پلیر را در خود ذخیره می کند

```
static ray_hit hit;
```

متغیر استاتیک اشعه های برخوردی (`Raycast`) با نام `hit` تعریف کرده ام، این متغیر کاربردهای متفاوتی دارد، مثلاً برد اسلحه، نقطه شلیک اسلحه به هدف، شناسایی یک اسلحه بر روی زمین، شناسایی مهمات و غیره.

```
static int FKeyAmmo=۱۰;
```

متغیر استاتیک از نوع عدد صحیح با نام `FKeyAmmo` که مقدار مهماتی را در خود ذخیره می کند (وقتی دکمه `F` صفحه کلید زده شد از این متغیر استفاده می شود - مانند اسلحه `shotgun` عمل می کند و در اینجا انفجاری نقطه زن است)

```
static int GKeyAmmo=۵;
```

متغیر استاتیک از نوع عدد صحیح با نام `GKeyAmmo` که مقدار مهماتی را در خود ذخیره می کند (وقتی دکمه `G` صفحه کلید زده شد از این متغیر استفاده می شود - بمب `C۴` جاسازی می شود)

```
static int HKeyAmmo=۷;
```

متغیر استاتیک از نوع عدد صحیح با نام `HKeyAmmo` که مقدار مهماتی را در خود ذخیره می کند (وقتی دکمه `H` صفحه کلید زده شد از این متغیر استفاده می شود - نارنجک پرتاب می شود)

```
static int JKeyAmmo=۱۰۰;
```

متغیر استاتیک از نوع عدد صحیح با نام JKeyAmmo که مقدار مهماتی را در خود ذخیره می کند (وقتی دکمه J صفحه کلید زده شد از این متغیر استفاده می شود - اسلحه آتش زا و آتش افکن و مسلسل را به کار میگیرد)

```
static void PublicPlayMySound(string
TriggerName)
{
    CryAudio::ControlId const MyTriggerId =
    CryAudio::StringToId(TriggerName);
    gEnv->pAudioSystem-
>LoadTrigger(MyTriggerId);
    gEnv->pAudioSystem-
>ExecuteTrigger(MyTriggerId);
    //gEnv->pAudioSystem-
>StopTrigger(MyTriggerId);
    //gEnv->pAudioSystem-
>UnloadTrigger(MyTriggerId);
}
```

در این فصل به توضیح این تابع پرداخته ام و شما می توانید توابع دیگری را پیاده کنید و مانند متغیرها و این تابع در تمامی کلاس ها استفاده کنید فایل StdAfx.cpp تنها شامل این سه خط کد می شوند که دو خط توضیح و یک خط پایانی می گوید که از هدر فایل StdAfx.h استفاده می شود.

```
// CryEngine Source File
// Copyright ©, Crytek, ۱۹۹۹-۲۰۱۶
```

```
#include "StdAfx.h"
```

قبل از اینکه بخواهم هدر فایل Player.h و Player.cpp را تشریح کنم (به فصل بعدی موکول می شود)، لازم می دانم که فایل CMakeLists.txt را توضیح دهم:

بیشتر بخش های این فایل بدون دستکاری و ویرایش خواهند ماند اما بخشی از این فایل وجود دارد که مستقیماً باید ویرایش شود و کدهایی که به صورت هدر فایل ها یا سی پی سی فایل ها است را باید اضافه یا حتی حذف کنید و گروه هایی از کامپونت ها یا مجموعه ای از کدهای هدر فایل ها و سی پی سی فایل ها با ایجاد پوشه های مختلف ایجاد کنید، این فایل به ویژوال استودیو و کرای انجین می گوید که چگونه عمل بیلد و کامپایل پروژه بازی انجام شود

```
cmake_minimum_required (VERSION ۳,۶,۲)
```

این خط کمترین ورژن مورد نیاز برای کامپایل پروژه را گوش زد می کند، خوشبختانه ما از نسخه ۳,۷ به بعد استفاده می کنیم.

```
set(CRYENGINE_DIR "C:/CE ۵,۴  
Final/CRYENGINE_۵,۴")
```

این خط به ویژوال استودیو می گوید که مسیر کرای انجین کجاست و سندباکس در کدام مسیر است.

```
set(TOOLS_CMAKE_DIR  
"${CRYENGINE_DIR}/Tools/CMake")
```

این خط مسیر ابزار CMake را به ویژوال استودیو می گوید.

```
set(PROJECT_BUILD ۱)
```

```
set(PROJECT_DIR
"C:/Users/GameEngine/Desktop/RobotIsland")
```

این خط پروژه ی بازی که بر روی آن کار می کنیم را به ویژوال استودیو می گوید

```
include("${TOOLS_CMAKE_DIR}/CommonOptions.cmake")
```

این خط به ویژوال استودیو می گوید که نحوه کامپایل پروژه چگونه انجام شود

```
add_subdirectory("${CRYENGINE_DIR}"
"${CMAKE_CURRENT_BINARY_DIR}/CRYENGINE")
```

این خط مسیری که فایل exe کرای انجین برای ارتباط با سندباکس را مشخص می کند

```
include("${TOOLS_CMAKE_DIR}/Configure.cmake")
```

این خط نحوه پیکربندی و ساختار پیکربندی برای کامپایل پروژه را برای ویژوال استودیو بیان می کند

```
start_sources()
```

شروع منابعی که با کدهای C++ توسط برنامه نویس و ویژوال استودیو تولید شده را آماده کامپایل می کند

```
sources_platform(ALL)
```

این خط سورس منابعی که باعث کامپایل پروژه در سیستم های سخت افزاری ۳۲ بیتی و ۶۴ بیتی را برای بازی فراهم می کند

```
add_sources("Code_uber.cpp"
    PROJECTS Game
    SOURCE_GROUP "Root"
        "GamePlugin.cpp"
        "StdAfx.cpp"
        "GamePlugin.h"
        "StdAfx.h"
    )
```

این هدر فایل ها و سی پی پی فایل ها توسط ویژوال استودیو ایجاد شده اند و پایه پروژه بازی را تشکیل می دهند.

```
add_sources("Components_uber.cpp"
    PROJECTS Game
    SOURCE_GROUP "Components"
        "Components/Bullet.cpp"
        "Components/CfBomb.cpp"
        "Components/Grenade.cpp"
        "Components/MyBoxEntity.cpp"
        "Components/Particle۰۱.cpp"
        "Components/Player.cpp"
        "Components/SpawnPoint.cpp"
        "Components/TriggerArea۰۱.cpp"
        "Components/Bullet.h"
        "Components/CfBomb.h"
        "Components/Grenade.h"
```

```
"Components/MyBoxEntity.h"
"Components/Particle.۱.h"
"Components/Player.h"
"Components/SpawnPoint.h"
"Components/TriggerArea.۱.h"
"Components/ItemComponent.h"
"Components/ItemComponent.cpp"
"Components/ItemProperties.h"
"Components/MyCVars.۱.h"
"Components/MyCVars.۱.cpp"
```

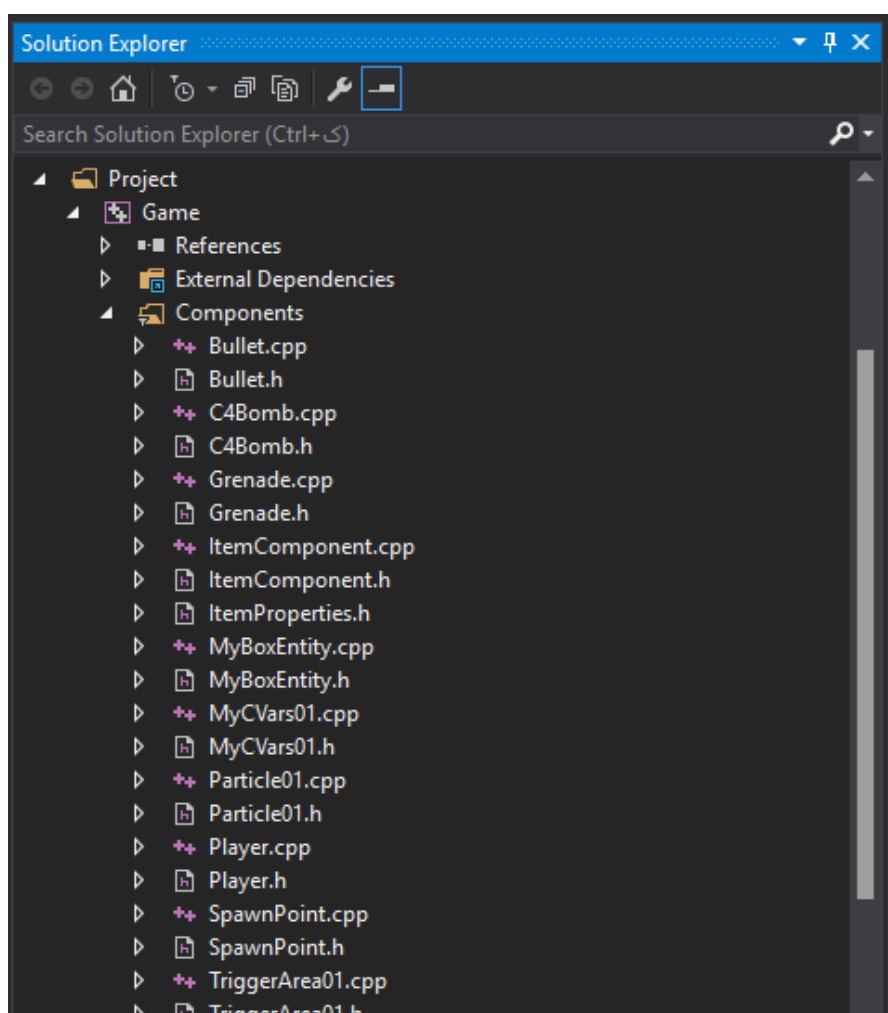
)

این هدر فایل ها و سی پی پی فایل ها توسط برنامه نویس ایجاد شده اند و به صورت دستی و مستقیما تایپ می شوند، وقتی که شما در حال ساخت entity ها یا component ها هستید و هدر فایل و سی پی پی فایل را کدنویسی کردید، حتما قبل از کامپایل شدن باید در اینجا به صورت دستی و مستقیما مثل آنچه که در بالا می بینید وارد می کنید و سپس عمل کامپایل را انجام می دهید در فصل قبلی توضیحات مفصل در این رابطه ارائه شد

end_sources()

این خط کد پایان منابع و سورس کدها را مشخص می کند، لازم به ذکر است اساس ساختار بندی و ساختار درختی در ویژوال استودیو به نحوه پوشه بندی منابع و سورس کدها در این فایل (CMakeLists.txt) بستگی دارد، مثلاً یک پوشه را با نام Player ایجاد کنیم و تمامی هدر فایل های پلیر و سی پی پی فایل پلیر را داخل آن پوشه قرار دهیم و برای entity ها یا component های دیگر نیز می توانیم پوشه های مختلف را ایجاد

کنیم، در تصویر زیر می بینید که پوشه ای با نام Components در ویژوال استودیو وجود دارد که تمامی هدر فایل ها و سی پی پی فایل ها در آن وجود دارد



```
CryEngineModule(Game PCH "StdAfx.cpp"  
SOLUTION_FOLDER "Project")
```


این خط ماژول های کرای انجین را در قالب پروژه به بازی الصاق می کند و اساس این ماژول ها در هدر فایل StdAfx.h است و این هدر فایل بوسیله فایل StdAfx.cpp قابل دسترس است

```
target_include_directories(${THIS_PROJECT}
PRIVATE
```

```
“${CRYENGINE_DIR}/Code/CryEngine/CryCommon
”
```

```
“${CRYENGINE_DIR}/Code/CryEngine/CryAction
”
```

```
“${CRYENGINE_DIR}/Code/CryEngine/CryS
chematyc/Core/Interface”
```

```
“${CRYENGINE_DIR}/Code/CryPlugins/Cry
DefaultEntities/Module”
)
```

این چند خط کد مسیر همه فایل های C++ کرای انجین را در پروژه در صورت نیاز در هر بخش تعریف و معرفی می نماید

Set StartUp project in Visual Studio

```
add_library(GameLauncher STATIC
"${CRYENGINE_DIR}/Code/CryEngine/CryCommon
/CryCore/Platform/platform.h")
set_target_properties(GameLauncher
PROPERTIES LINKER_LANGUAGE CXX)
if (WIN۳۲)
```

```
set_visual_studio_debugger_command(GameLaunche
```

```

r
"${CRYENGINE_DIR}/bin/win_x64/GameLauncher.exe"
"-project
"C:/Users/GameEngine/Desktop/RobotIsland/Game
.cryproject\"")
endif()

```

این چند خط کد نحوه اجرا و چگونگی اجرا و مسیر اجرا پروژه بازی را به صورت فایل باینری exe برای ویژوال استودیو بیان می کند

```

add_library(Sandbox STATIC
"${CRYENGINE_DIR}/Code/CryEngine/CryCommon/Cry
Core/Platform/platform.h")
set_target_properties(Sandbox PROPERTIES
LINKER_LANGUAGE CXX)
if (WIN32)
    set_visual_studio_debugger_command(Sandbox
"${CRYENGINE_DIR}/bin/win_x64/Sandbox.exe" "-
project
"C:/Users/GameEngine/Desktop/RobotIsland/Game
.cryproject\"")
endif()

```

این چند خط کد نحوه اجرا و چگونگی اجرا و مسیر اجرا پروژه بازی را به صورت فایل باینری exe در سندباکس برای ویژوال استودیو بیان می کند

```

add_library(GameServer STATIC
"${CRYENGINE_DIR}/Code/CryEngine/CryCommon/Cry
Core/Platform/platform.h")

```

```
set_target_properties(GameServer PROPERTIES
LINKER_LANGUAGE CXX)
if (WIN۳۲)
```

```
set_visual_studio_debugger_command(GameServer
"${CRYENGINE_DIR}/bin/win_x۶۴/Game_Server.exe"
"-project
"C:/Users/GameEngine/Desktop/RobotIsland/Game
.cryproject\"")
endif()
```

این چند خط کد نحوه اجرا و چگونگی اجرا و مسیر اجرا پروژه بازی را به صورت فایل باینری exe تحت شبکه برای ویژوال استودیو بیان می کند

```
set_solution_startup_target(GameLauncher)
```

```
if (WIN۳۲)
    set_visual_studio_debugger_command(
    ${THIS_PROJECT}
    "${CRYENGINE_DIR}/bin/win_x۶۴/GameLauncher.exe"
    "-project
    "C:/Users/GameEngine/Desktop/RobotIsland/Game
    .cryproject\"")
endif()
```

این چند خط کد نحوه اجرا و چگونگی اجرا و مسیر اجرا پروژه بازی را به صورت فایل باینری exe به صورت ۳۲ بیتی برای ویژوال استودیو بیان می کند

```
#BEGIN-CUSTOM
```

```
# Make any custom changes here, modifications
outside of the block will be discarded on
regeneration.
```

```
#END-CUSTOM
```

اینجا نیز در صورت داشتن تبحر و مهارت شما می توانید کدهایی را بر اساس نیازتان با توانایی ساخت پلاگین ها و دیگر ویژگی ها پیاده سازی کنید

در پروژه بازی دو نوع Uber وجود دارد که تفکیک کدهای پایه و کدهای برنامه نویس را انجام می دهد، لازم است که بدانید وقتی که entity ها یا component هایی را می سازید، باید فایل ها *.cpp را نه *.h را در فایل تفکیک کدهای برنامه نویس با نام Components_uber.cpp ثبت نماید، مانند آنچه که در کدهای زیر در فایل Components_uber.cpp نوشته شده است :

```
// Unity Build Uber File generated by CMake
// This file is auto generated, do not
manually edit it.
```

```
#include <StdAfx.h>
```

```
#include
```

```
<C:/Users/GameEngine/Desktop/RobotIsland/Code/
Components/Bullet.cpp>
```

```
#include
```

```
<C:/Users/GameEngine/Desktop/RobotIsland/Code/
Components/CfBomb.cpp>
```

```
#include
```

```
<C:/Users/GameEngine/Desktop/RobotIsland/Code/
Components/Grenade.cpp>
```

```
#include
```

```
<C:/Users/GameEngine/Desktop/RobotIsland/Code/Components/MyBoxEntity.cpp>
```

```
#include
```

```
<C:/Users/GameEngine/Desktop/RobotIsland/Code/Components/Particle\1.cpp>
```

```
#include
```

```
<C:/Users/GameEngine/Desktop/RobotIsland/Code/Components/Player.cpp>
```

```
#include
```

```
<C:/Users/GameEngine/Desktop/RobotIsland/Code/Components/SpawnPoint.cpp>
```

```
#include
```

```
<C:/Users/GameEngine/Desktop/RobotIsland/Code/Components/TriggerArea\1.cpp>
```

```
#include
```

```
<C:/Users/GameEngine/Desktop/RobotIsland/Code/Components/ItemComponent.cpp>
```

```
#include
```

```
<C:/Users/GameEngine/Desktop/RobotIsland/Code/Components/MyCVars\1.cpp>
```

فایل تفکیک کدهای پایه نیز با نام Code_uber.cpp وجود دارد و کدهای

زیر به صورت چند خط زیر تعریف شده است و تمام

```
// Unity Build Uber File generated by CMake
// This file is auto generated, do not
manually edit it.
```

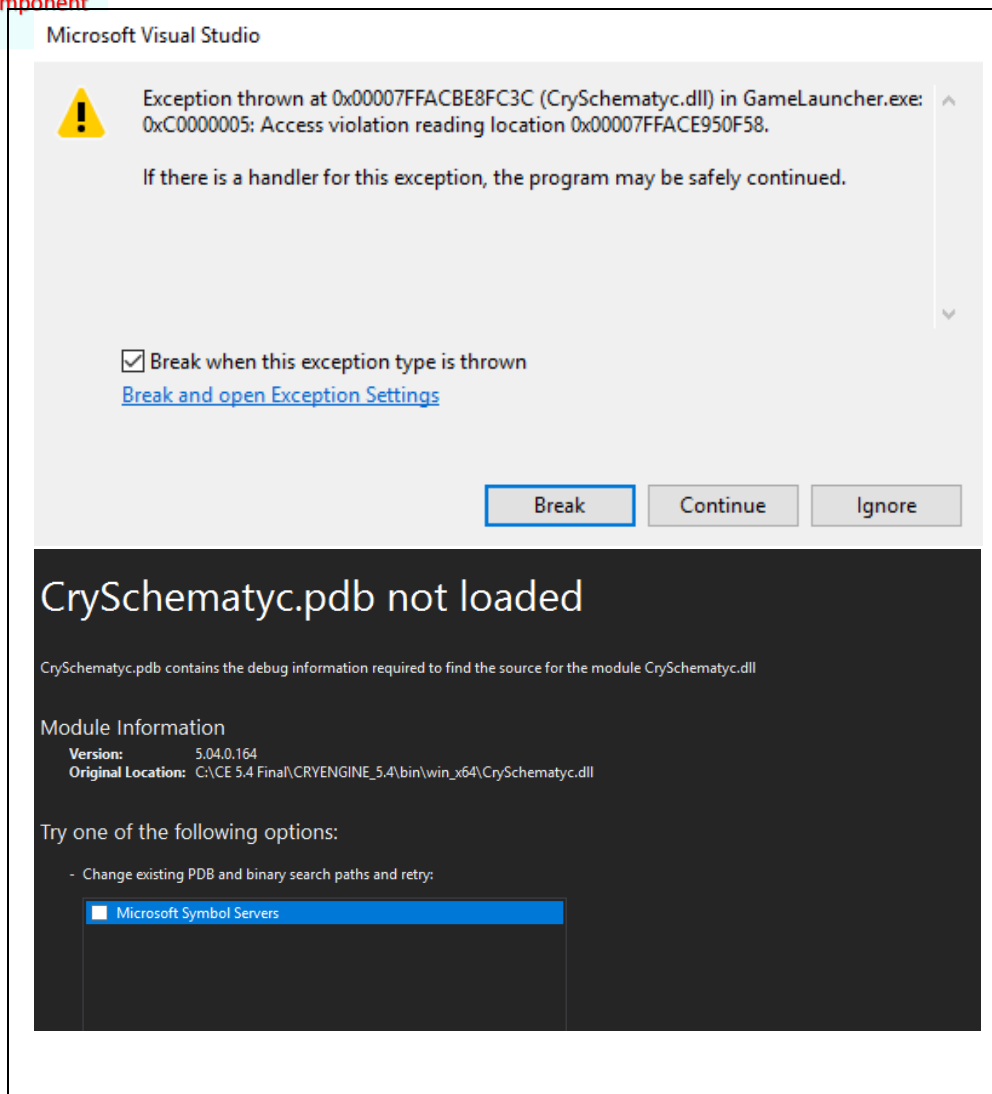
```
#include <StdAfx.h>
```

```
#include
```

```
<C:/Users/GameEngine/Desktop/RobotIsland/Code/  
GamePlugin.cpp>
```

نکته ای قابل تامل در سیماتیک کرای انجین ۵,۴، این است که زمانی در داخل سندباکس از کلاس و یا پریفب های سیماتیک استفاده می کنید و در **Visual Studio** نیز در حال کدزنی هستید و عمل بیلد یا همان کامپایل را انجام می دهید و بعد از اجرای بازی و بستن پنجره بازی، پیغام زیر ظاهر می شود که نشان می دهد شما از سیماتیک در مرحله استفاده کردید یا در پنجره **Assets Broswer** پروژه بازی فایل های کلاس سیماتیک وجود دارند که با حذف فایل های سیماتیک در **Assets Broswer** در محیط **Visual Studio** دیگر شاهد این خطا نخواهید بود





حالا به بررسی کدهای داخل هدر فایل GamePlugin.h می پردازم :

```
#pragma once
```

```
#include <CrySystem/ICryPlugin.h>
```

```
#include <CryGame/IGameFramework.h>
```

```
#include <CryEntitySystem/IEntityClass.h>
#include <CryNetwork/INetwork.h>
#include "Components/MyCVars.۱.h"
```

این چند خط از پیش پردازنده ها که با کلمه **include** شروع می شوند اساس و پایه موارد زیر را برای استفاده کردن از کدها را تشکیل می دهند :

۱- پلاگین های نوشته شده به دست کرایتک و پلاگین های نوشته شده بدست برنامه نویس

۲- کتابخانه ای عظیم از واسط ها، کلاس ها و کدهای مختلف برای بازی

۳- پیکربندی و ثبت کلاس ها نوشته شده به دست کرایتک و نوشته شده به دست برنامه نویس

۴- پیکربندی و شبکه سازی بازی برای ساخت بازی های تحت شبکه

۵- ایجاد متغیر کنسولی

```
class CPlayerComponent;
```

اینجا کلاس **Player** را که از دو هدر فایل **Player.h** و **Player.cpp** استفاده شده است را پیش تعریف می کند و از این کلاس در این هدر فایل و فایل **GamePlugin.cpp** استفاده می شود.

```
class CGamePlugin
: public ICryPlugin
, public ISystemEventListener
, public INetworkedClientListener
```

این مدخلی ورودی برای برپاسازی پنجره بازی را تشکیل می دهد و به صورت اتوماتیک یک کپی از کلاس **CGamePlugin** ایجاد کرده و در کتابخانه بازی بارگذاری می شود، وقتی که یک با استفاده از کلاس **CPlayerComponent** یک پلیمر غیر شبکه برپاسازی و ساخته می

شود، رویداد سیستمی `OnClientConnectionReceived` برای اولین بار اجرا میشود.

```
{
public:
CRYINTERFACE_SIMPLE(ICryPlugin)
CRYGENERATE_SINGLETONCLASS_GUID(CGamePlugin,
"Game_Blank", "{FC۹BD۸۸۴-۴۹DE-۴۴۹۴-۹D۶۴-
۱۹۱۷۳۴BBB۷E۳}"_cry_guid)
```

در این دو خط کد از هدر فایل `ClassWeaver.h` استخراج می شود و فضای خالی و سه بعدی را در پنجره بازی ایجاد می کند

```
virtual ~CGamePlugin();
```

تخریب کننده کلاس `CGamePlugin` در زمانی که بازی پایان یافت اتفاق می افتد

```
virtual const char* GetName() const override {
return "GamePlugin"; }
```

این خط واسطی برای دریافت نام پلاگینی است که در بازی استفاده می شود

```
virtual const char* GetCategory() const
override { return "Game"; }
```

این خط واسطی برای دریافت نام بازی است که در کاتالوگ بازی تعریف می شود

```
virtual bool
Initialize(SSystemGlobalEnvironment& env,
const SSystemInitParams& initParams) override;
```

مقادیر اولیه برای پنجره بازی را اختصاص می دهد

```
virtual void OnPluginUpdate(EPluginUpdateType
updateType) override {}
```

این رویداد هیچ عملی را انجام نمی دهد زیرا که هیچ کدی به آن اختصاص داده نشده است تا به روز رسانی در سطح پلاگین اتفاق بیفتد، اگرچه باید در سطح کلاس پدر یک تابع با نام OnPluginUpdate پیش نیاز برای عمل لود پلاگین های کرای انجین را انجام خواهد داد

```
virtual void OnSystemEvent(ESystemEvent event,
UINT_PTR wparam, UINT_PTR lparam) override;
```

اولین رویدادی که بعد از ساخت پنجره بازی اجرا می شود، رویداد OnSystemEvent است

```
virtual void
OnLocalClientDisconnected(EDisconnectionCause
cause, const char* description) override {}
```

رویداد OnLocalClientDisconnected هنگامی که قطع شبکه disconnect اتفاق می افتد اجرا می شود، این رویداد در ارتباط با Local Client است

```
virtual bool OnClientConnectionReceived(int
channelId, bool bIsReset) override;
```

رویداد `OnClientConnectionReceived` هنگامی اجرا می شود که وقتی یک `Client` جدید به بازی متصل می شود و در اینجا مقدار `false` به معنی عدم اجازه ارتباط با شبکه است

```
virtual bool OnClientReadyForGameplay(int  
channelId, bool bIsReset) override;
```

رویداد `OnClientReadyForGameplay` هنگامی اجرا می شود که وقتی یک `Client` جدید آماده بازی کردن است و در اینجا مقدار `false` به معنی عدم اجازه ارتباط با شبکه و بیرون کردن پلیمر است.

```
virtual void OnClientDisconnected(int  
channelId, EDisconnectionCause cause, const  
char* description, bool bKeepClient) override;
```

رویداد `OnClientDisconnected` هنگامی اجرا می شود که یک `Client` قطع می شود

```
virtual bool OnClientTimingOut(int channelId,  
EDisconnectionCause cause, const char*  
description) override { return true; }
```

رویداد `OnClientTimingOut` اجازه می دهد که یک `Client` عمل `time out` را در هنگامی که پکتی ارسال نمی شود را انجام دهد، هرگاه مقدار `true` بازگردانده شود یعنی اجازه داده می شود که عمل `disconnect` انجام شود در غیر اینصورت اگر مقدار `false` باشد `Client` در شبکه نگه داشته می شود

protected:

```
std::unordered_map<int, EntityId> m_players;
```

با استفاده از این متغیر یک کلید برای پلیر تعریف می شود که عمل دریافت را در رویداد OnClientConnectionReceived انجام می شود و کلید همان Id کانال دریافتی از شبکه است

```
CAhmadKaramiCVars* m;
```

یک متغیر کلاسی با نام m را بر اساس نوع داده کلاسی CAhmadKaramiCVars تعریف شده است که در ارتباط با ساخت متغیرهای کنسولی است و در فصل قبلی به نحوه ساخت متغیرهای کنسولی پرداختم

```
};
```

در ادامه به بررسی کدهای داخل هدر فایل GamePlugin.cpp می پردازم :

```
#include "StdAfx.h"
#include "GamePlugin.h"
```

هدر فایل GamePlugin.h در صفحات قبلی بررسی شد و حالا داخل این فایل از توابع و متغیر این هدر فایل استفاده می شود

```
#include "Components/Player.h"
```

اگر به یاد داشته باشید کلاس CPlayerComponent در هدر فایل Player.h GamePlugin.h تعریف شد، حالا نیاز است که هدر فایل

نیز در این فایل اضافه شود تا از کلاس CPlayerComponent استفاده کنیم

```
#include <CrySchematyc/Env/IEnvRegistry.h>
#include <CrySchematyc/Env/EnvPackage.h>
#include <CrySchematyc/Utils/SharedString.h>
```

این هدر فایل ها در ارتباط با ثبت پلاگین و ساختار کامپونت های شما در سیماتیک است

```
#include <IGameObjectSystem.h>
#include <IGameObject.h>
```

هنگامی که از این دو هدر فایل استفاده می کنید، هر گاه که یک شی یا اشیاء ی را ایجاد می کنید با استفاده از این دو هدر فایل رویدادی های مختلفی را برای آن تعریف می کنید تا بتوانید از این رویدادها در Entity ها یا Component ها استفاده کنید و منشاء اولیه entity ها و component ها در این دو خط است

```
#include <CryInput\IHardwareMouse.h>
```

این هدر فایل در ارتباط با نحوه استفاده از ماوس است

```
#include <CryCore/Platform/platform_impl.inl>
```

این خط کد Dll های مختلف را برای sandbox تولید می کند، این عمل فقط یکبار در هر dll ی که ایجاد می شود انجام می شود

```
CGamePlugin::~CGamePlugin()
{
```

```

gEnv->pGameFramework-
>RemoveNetworkedClientListener(*this);
gEnv->pSystem->GetISystemEventDispatcher()-
>RemoveListener(this);

if (gEnv->pSchematyc)
{
gEnv->pSchematyc-
>GetEnvRegistry().DeregisterPackage(CGamePlugin::GetCID());
}
}

```

تخریب کننده کلاس CGamePlugin هر چیزی را که در ارتباط با بازی است از بین می برد تا منابع مختلف حافظه را به RAM برگرداند، این مسئله برای محیط سیماتیک نیز صادق است

```

bool
CGamePlugin::Initialize(SSystemGlobalEnvironment& env, const SSystemInitParams& initParams)
{
    رویدادهای مختلف برای ثبت وقایع بازی در نهایت آماده به کار می شوند و
    مرحله نیز با تمامی پارامترها CVar و دیگر کدها باید لود شوند
    gEnv->pSystem->GetISystemEventDispatcher()-
    >RegisterListener(this, "CGamePlugin");
}

```

نهایتا ثبت کلیه رویدادها با این خط کد انجام می شود.

```

gEnv->pGameFramework-
>AddNetworkedClientListener(*this);

```

برای داشتن یک بازی تحت شبکه اجرای این خط این تضمین را ایجاد می کند.

```
return true;
```

مقدار بازگشتی این خط کد در هدر فایل GamePlugin.h توضیح داده شده است

```
}
```

```
void CGamePlugin::OnSystemEvent(ESystemEvent
event, UINT_PTR wparam, UINT_PTR lparam)
{
    این رویداد بعد از ایجاد پنجره بازی اجرا می شود
    switch (event)
    {
```

حالا باید تعیین کرد که اجرا بازی به چه شیوه ای انجام می شود و از دستور switch استفاده می کنیم

```
case ESYSTEM_EVENT_REGISTER_SCHEMATYC_ENV:
{

    auto staticAutoRegisterLambda =
    [](Schematyc::IEnvRegistrar& registrar)
    {
```

ثبت همه کامبونت ها که در داخل این پلاگین استفاده می شود و عبارت لاندای مجموعه ای برای سیماتیک محاسبه و اجرا می شود.

```

Detail::CStaticAutoRegistrar<Schematyc::IEnvRe
gistrar>::InvokeStaticCallbacks(registrar);
};

if (gEnv->pSchematyc)
{
gEnv->pSchematyc-
>GetEnvRegistry().RegisterPackage(
stl::make_unique<Schematyc::CEnvPackage>(
CGamePlugin::GetCID(),
"EntityComponents",
"Crytek GmbH",
"Components",
staticAutoRegisterLambda
)
);
}
}
break;

```

این کدها برای سیماتیک اجرا می شوند و در محیط ادیتور یا همان سندباکس انعکاس پیدا می کنند، به صورت خلاصه می توان بگویم هنگامی که شما می خواهید **component** یا **entity** را ایجاد کنید، قطعاً نیاز دارید که قبلاً ثبت عملیات رویدادها برای سیماتیک انجام شده باشد، اینجا نیز این رویدادها در اولین مرحله برای سیماتیک ثبت می شود و سپس در سطح پلاگین است که می توانید از سیماتیک نیز استفاده کنید، اگرچه شما نیاز به دانستن این بخش هم نداشتید زیرا که سیماتیک کرای انجین ۵,۵ به مراتب توسعه یافته تر از سیماتیک کرای انجین ۵,۴ است، منظور من در اینجا این است که احتمالاً بخشی از **API** ها در سیماتیک در سطح پلاگین نیز برای سیماتیک در کرای انجین ۵,۵ عوض شود.


```
case ESYSTEM_EVENT_GAME_POST_INIT:
```

```
{
```

در صفحات آغازین فصل قبل این بخش را به صورت مفصل توضیح دادم، منطق بازی بعد از لود مرحله و لود CVar ها شروع به اجرا می شود و شما باید تغییراتی که مدنظر دارید را در اینجا انجام دهید، مثلاً صفر کردن مقادیر متغیرها و...

```
if (!gEnv->IsEditor())
```

```
{
```

```
gEnv->pConsole->ExecuteString("map example.۱",
```

```
false, true);
```

```
gEnv->pConsole->ExecuteString("e_TimeOfDay
```

```
۱,۲۳");
```

```
gEnv->pConsole-
```

```
>ExecuteString("sys_DeactivateConsole ۱");
```

```
gEnv->pConsole->ExecuteString("r_DisplayInfo
```

```
۰.");
```

```
CryLog("gEnv->pConsole-
```

```
>ExecuteString(e_TimeOfDay ۱,۲۳) ; -)
```

```
GamePlugin");
```

```
if(!m)
```

```
{
```

```
m=new CAhmadKaramiCVars();
```

```
m->RegisterCVars();
```

```
}
```

بلوک `if` مربوطه برای ساخت متغیرکنسولی در کلاس `CAhmadKaramiCVars` در فصل قبل توضیح داده شده است

```
}
}
break;
}
}
```

```
bool
CGamePlugin::OnClientConnectionReceived(int
channelId, bool bIsReset)
{
```

یک پلیر و کامبونت مربوطه به پلیر توسط این رویداد ایجاد می شود و ارتباط بین `client` را دریافت می کند و مقدار `false` به معنی عدم اجازه ارتباط با شبکه و بیرون کردن پلیر است

```
SEntitySpawnParams spawnParams;
spawnParams.pClass = gEnv->pEntitySystem-
>GetClassRegistry()->GetDefaultClass();
spawnParams.sName = "Player";
spawnParams.nFlags |=
ENTITY_FLAG_NEVER_NETWORK_STATIC;

if (m_players.size() == 0 && !gEnv-
>IsDedicated())
{
spawnParams.id = LOCAL_PLAYER_ENTITY_ID;
spawnParams.nFlags |=
ENTITY_FLAG_LOCAL_PLAYER;
}
```

این چند خط کد به پلاگین می فهماند که پلیری که قرار است تکثیر شود باید یکسری مقدار اولیه و پارامتری دریافت کند تا عمل تکثیر بعدا انجام شود مثلا نام پلیر می تواند هر چیزی باشد اما اسم آن را همان "Player" می گذاریم.

اگر id برای پلیر لحاظ نشود همان flag مربوطه با عنوان ENTITY_FLAG_LOCAL_PLAYER در نظر گرفته می شود و id آن LOCAL_PLAYER_ENTITY_ID خواهد شد، یعنی بازی تحت شبکه نیست

```
if (IEntity* pPlayerEntity = gEnv-
>pEntitySystem->SpawnEntity(spawnParams))
{
```

با دستور if عمل تکثیر برای پلیر فراهم می شود

```
pPlayerEntity->GetNetEntity()-
>SetChannelId(channelId);
```

این خط برای پلیر یک id را از کانال گرفته و با پارامتر channelId در شبکه استفاده می کند

```
pPlayerEntity->GetNetEntity()-
>BindToNetwork();
```

برای پیوستن پلیر به شبکه از این خط کد استفاده می شود

```
CPlayerComponent* pPlayer = pPlayerEntity-
>GetOrCreateComponentClass<CPlayerComponent>()
;
```

با این خط کد حالا پلیر ایجاد می شود

```
m_players.emplace(std::make_pair(channelId,
pPlayerEntity->GetId()));
```

طبق کلید و id که برای کانال ایجاد کردیم حالا نوبت آن رسیده است که پلیر را به داخل مرحله ای که ساختیم وارد کنیم

```
}
```

```
return true;
```

مقدار بازگشتی این خط کد در هدر فایل GamePlugin.h توضیح داده شده است

```
}
```

```
bool CGamePlugin::OnClientReadyForGameplay(int
channelId, bool bIsReset)
```

```
{
```

با این رویداد می توان پلیرها را تحت شبکه احیا کنیم در زمانی که هر پلیر آماده بازی است

```
auto it = m_players.find(channelId);
```

متغیر it نوع id کانال را پیدا می کند

```
if (it != m_players.end())
```

اگر پلیرها در لیستی که وجود دارد تمام نشده اند دستور `if` بعدی اجرا خواهد شد

```
if (IEntity* pPlayerEntity = gEnv-
>pEntitySystem->GetEntity(it->second))
{
```

این خط کد بررسی می کند که پلیری که قرار است تحت شبکه ایجاد شود کدام یک خواهد بود و در کجای لیست می تواند باشد

```
if (CPlayerComponent* pPlayer = pPlayerEntity-
>GetComponent<CPlayerComponent>())
```

این خط کد کامپوننت مربوطه به کلاس `CPlayerComponent` را می گیرد و در داخل متغیر `pPlayer` می ریزد

```
{
pPlayer->Revive();
```

تابع `Revive` در لیست اشاره گر با دسترسی `pPlayer` اجرا می شود

```
}
}
}
```

```
return true;
```

مقدار بازگشتی این خط کد در هدر فایل `GamePlugin.h` توضیح داده شده است

```
}
```

```
void CGamePlugin::OnClientDisconnected(int
channelId, EDisconnectionCause cause, const
char* description, bool bKeepClient)
{
```

وقتی Client یا همان Player از شبکه خارج می شود یا ناگهان در ارتباط با شبکه اتصال قطع می شود، عمل حذف entity یا همان حذف پلیر از شبکه و البته از روی مرحله انجام می شود

```
auto it = m_players.find(channelId);
```

متغیر it نوع id کانال را پیدا می کند

```
if (it != m_players.end())
{
```

اگر پلیرها در لیستی که وجود دارد تمام نشده اند دستور if بعدی اجرا خواهد شد

```
gEnv->pEntitySystem->RemoveEntity(it->second);
```

این خط کد بررسی می کند که پلیری که قرار است تحت شبکه ایجاد شود کدام یک خواهد بود و در کجای لیست می تواند باشد

```
m_players.erase(it);
```

پلیر یا همان Client حذف می شود

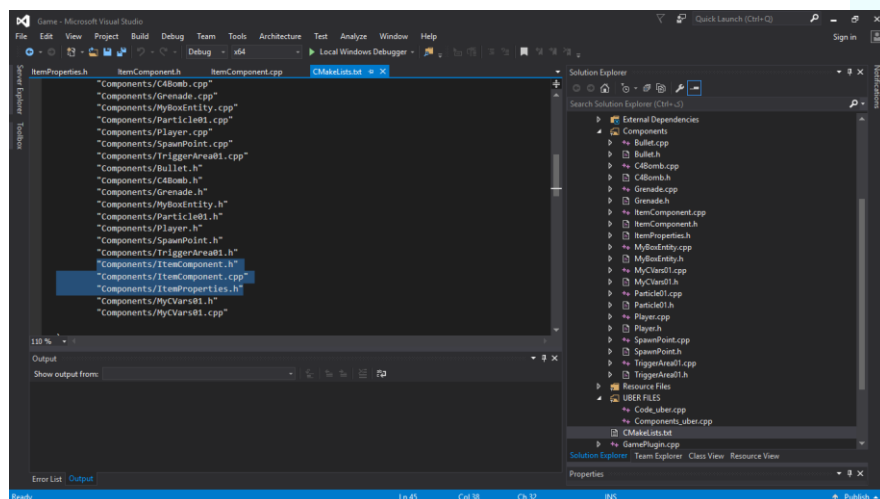
```
}
}
```

```
CRYREGISTER_SINGLETON_CLASS(CGamePlugin)
```

کلیه عملیات مربوطه همانطور که در کلاس CGamePlugin است باید در سندباکس و البته در خروجی فایل exe تحت شبکه یا تحت کامپیوتر خارج از شبکه ثبت شود و با این خط کد این عمل انجام می شود

ایجاد entity ها و component های ویژه

مانند آنچه که در مراحل قبلی برای هدرفایل ها و فایل های سی پی پی انجام دادید حالا سه فایل باید ایجاد کنید که به ترتیب با نام های `ItemProperties.h`، `ItemComponent.h` و `ItemComponent.cpp` است و این سه فایل باید در مسیر پروژه `RobotIsland\Code\Components` کپی کنید و از آنجایی که در صفحات قبلی خطوطی از کدها را توضیح دادم و این خطوط در هدرفایل ها و سی پی پی فایل ها تکرار خواهد شد، از تکرار توضیحات مجدد چشم پوشی می کنم و در ادامه صفحات نیز فقط به ذکر کدهای جدید بسنده خواهم کرد، لازم به ذکر مجدد است که از این سه فایل به عنوان یک Entity جدید یا component جدید می توانید استفاده کنید که دارای property های مختلف بوده و در پنجره properties در سندباکس می توانید مقادیر جدیدی را به آن property ها اختصاص دهید، به مثال زیر توجه کنید.



فراموش نکنید که هر entity یا component از ساده تا پیشرفته و جدید که می سازید مانند آنچه که در تصویر می بینید باید هدر فایل ها (*.h) و سی پی پی فایل ها (*.cpp) در داخل فایل CMakeLists.txt اضافه شود، در فصل قبلی نحوه اضافه کردن فایل ها و همچنین کامپایل کردن این فایل ها به صورت کامل توضیح دادم، در مثال های قبلی من entity و component هایی را کدنویسی کردم که دارای هیچ گونه properties ی نیستند و در داخل کدها باید این properties ها را عوض کنید، در حالی که این روش چندان جالبی نیست، مثلاً جعبه مهمات می تواند مقیاس های مختلفی در بزرگی و کوچکی داشته باشد و می تواند متریا ل یا شکل های مختلفی از مدل سه بعدی با وزن مختلف داشته باشد، در این بخش من به ساخت entity و component ی خواهم پرداخت که دارای properties هستند، ابتدا هدر فایل ItemProperties.h را توضیح می

دهم.

هدر فایل ItemProperties.h :

```
#pragma once

#include <CrySchematyc/ResourceTypes.h>
#include
<CrySchematyc/Reflection/TypeDesc.h>

#include <CrySchematyc/Utils/EnumFlags.h>
#include <CrySchematyc/Env/IEnvRegistry.h>
#include
<CrySchematyc/Env/IEnvRegistrar.h>
#include
<CrySchematyc/Env/Elements/EnvComponent.h>
#include
<CrySchematyc/Env/Elements/EnvFunction.h>
#include
<CrySchematyc/Env/Elements/EnvSignal.h>
#include <CrySchematyc/ResourceTypes.h>
#include <CrySchematyc/MathTypes.h>
#include
<CrySchematyc/Utils/SharedString.h>
```

این هدر فایل ها که با پیش پردازنده **include** تعریف شده اند وظایف مختلفی دارند و هریک به صورت مکمل برای ساخت **entity** ها و **component** ها کاربرد دارد و لازم است که از این هدر فایل ها استفاده کنیم.

```
enum ItemGeometrySlot
{
    GEOMETRY_SLOT=0
};
```

با استفاده از ساختمان داده `enum` یک شماره اسلات با عدد صفر را برای لود کردن مدل هندسی در نظر می گیریم

```
struct SRenderProperties
```

```
{
```

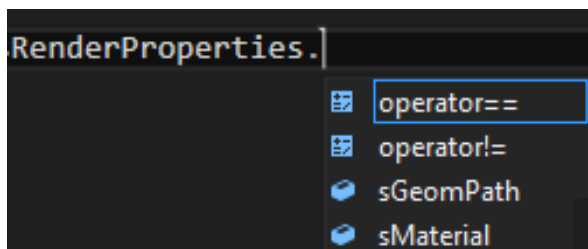
با استفاده از ساختمان داده `struct` یک `properties` را با نام `SRenderProperties` تعریف می کنیم که وظیفه آن به شرح زیر است.

```
inline bool operator ==(const  
SRenderProperties& rhs) const { return  
    .==memcmp(this,&rhs,sizeof(rhs)); }
```

با استفاده از دستور کلیدی `inline` عمل تخصیص حافظه را به همراه ایجاد و ساخت متد براساس `operator` مساوی برای ساختار `SRenderProperties` در نظر میگیریم، این ساختار به ما کمک می کند در پنجره `Properties` در ادیتور کرای انجین تشخیص دهیم مقدار `property` تغیر نیافته است

```
inline bool operator !=(const  
SRenderProperties& rhs) const { return . !=  
memcmp(this, &rhs, sizeof(rhs)); }
```

با استفاده از دستور کلیدی `inline` عمل تخصیص حافظه را به همراه ایجاد و ساخت متد براساس `operator` مخالف برای ساختار `SRenderProperties` در نظر میگیریم، این ساختار به ما کمک می کند در پنجره `Properties` در ادیتور کرای انجین تشخیص دهیم مقدار `property` تغیر یافته است.



```
Schematyc::GeomFileName sGeomPath;
```

متغیری با نام `sGeomPath` برای ایجاد یک `textbox` در پنجره `properties` و برای لود یک مدل سه بعدی در پنجره `assets browser` تعریف می کنیم، این متغیر از نوع داده `GeomFileName` منشاء سیماتیکی دارد به این معنا شما می توانید آن را نیز در سیماتیک ادیتور استفاده کنید

```
Schematyc::MaterialFileName sMaterial;
```

متغیری با نام `sMaterial` برای ایجاد یک `textbox` در پنجره `properties` برای لود یک متریال در پنجره `assets browser` تعریف می کنیم، این متغیر از نوع داده `MaterialFileName` منشاء سیماتیکی دارد به این معنا شما می توانید آن را نیز در سیماتیک ادیتور استفاده کنید

```
};
```

```
static void  
ReflectType(Schematyc::CTypeDesc<SRenderProper  
ties>& desc)
```

```
{
desc.SetGUID("{B۴۴۹D۲E۳-D۹۳۷-۴B۸۳-A۲EE-
۵۳BC۹۷E۳۵ECD}"_cry_guid);
desc.SetLabel("Render Properties");
desc.SetDescription("Render Properties");
```

تابع `ReflectType` عمل انعکاس این کدها را در سندباکس به نمایش می گذارد، فضایی را برای بخش `Render Properties` در پنجره `Properties` در نظر می گیرد تا بتوانید `textbox` های مختلف برای لود مدل سه بعدی و اختصاص متریال به مدل سه بعدی را ایجاد کنید

```
desc.AddMember(&SRenderProperties::sGeomPath,
'geom', "GeometryPath", "Geometry path",
"Geometry path of this item", "");
```

این خط یک `Textbox` را برای لود مدل هندسی در نظر میگیرد براساس متغیری که در بالا با نام `sGeomPath` تعریف کردم

```
desc.AddMember(&SRenderProperties::sMaterial,
'smat', "MaterialPath", "Material path",
"Material path of this item", "");
}
```

این خط یک `Textbox` را برای لود متریالی در نظر میگیرد براساس متغیری که در بالا با نام `sMaterial` تعریف کردم

```
struct SPhysicsProperties
{
```

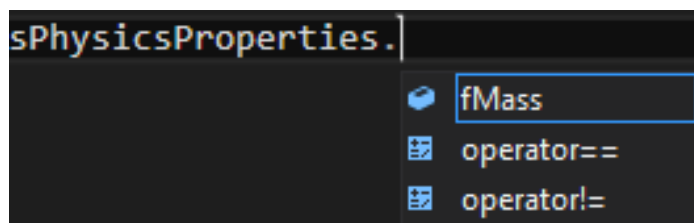
مجدداً یک ساختار با استفاده از ساختمان داده `struct` یک `properties` با نام `SPhysicsProperties` تعریف می کنیم که وظیفه آن به شرح زیر است.

با استفاده از دستور کلیدی `inline` عمل تخصیص حافظه را به همراه ایجاد و ساخت متد براساس `operator` مساوی برای ساختار `SPhysicsProperties` در نظر میگیریم، این ساختار به ما کمک می کند در پنجره `Properties` در ادیتور کرای انجین تشخیص دهیم مقدار `property` تغیر نیافته است.

```
inline bool operator ==(const
SPhysicsProperties& rhs) const { return . ==
memcmp(this, &rhs, sizeof(rhs)); }
```

و دوباره با استفاده از دستور کلیدی `inline` عمل تخصیص حافظه را به همراه ایجاد و ساخت متد براساس `operator` مخالف برای ساختار `SPhysicsProperties` در نظر میگیریم، این ساختار به ما کمک می کند در پنجره `Properties` در ادیتور کرای انجین تشخیص دهیم مقدار `property` تغیر یافته است

```
inline bool operator !=(const
SPhysicsProperties& rhs) const { return . !=
memcmp(this, &rhs, sizeof(rhs)); }
```



```
float fMass;
```

متغیری با نام `fMass` برای ایجاد یک `UpDownBox` با تعیین عددی برای جرم جسم (مدل سه بعدی) تعریف می کنیم، این متغیر از نوع داده `float` است

```
};
```

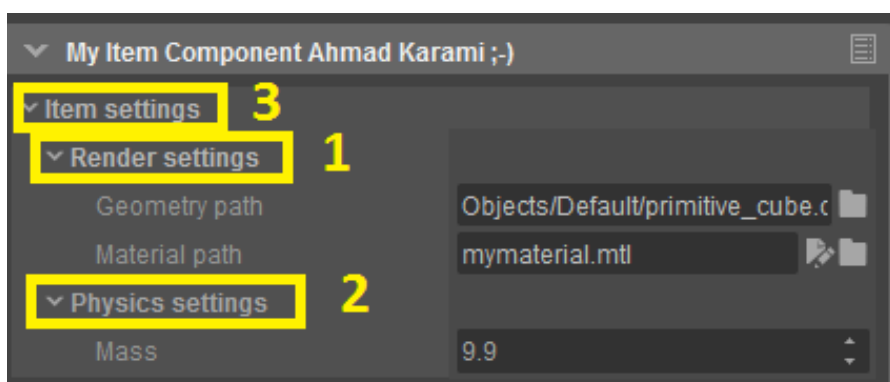
```
static void
ReflectType(Schematyc::CTypeDesc<SPhysicsProperties>& desc)
{
    desc.SetGUID("{CB۰۵۶۰۸۲-۰۶EA-۴CB۰-ACAY-۱۷۲۹۳۶A۳۴۴۹D}"_cry_guid);
    desc.SetLabel("Physics Properties");
    desc.SetDescription("Physics Properties");
}
```

تابع `ReflectType` عمل انعکاس این کدها را در سندباکس به نمایش می گذارد، فضایی را برای بخش `Physics Properties` در پنجره `Properties` در نظر می گیرد تا بتوانید `UpDownBox` مربوط به وارد کردن عددی برای جرم جسم (مدل سه بعدی) را ایجاد کنید

```
desc.AddMember(&SPhysicsProperties::fMass,
    'fmas', "Mass", "Mass", "Mass property of this item", ۰.۰f);
```

این خط یک UpDownBox را برای وارد کردن عدد برای جرم جسم (مدل سه بعدی) در نظر میگیرد براساس متغیری که در بالا با نام fMass تعریف کردم

}



به تصویر مربوطه و کادرهای زرد رنگ براساس شماره ها دقت کنید که عدد کادر زرد رنگ ۱ و عدد کادر زرد رنگ ۲ در کدهای قبلی بود که توضیح دادم و به بخش های Render settings و Physics settings مربوط است و حالا نوبت به آن رسیده است که عدد کادر زرد رنگ ۳ را توضیح دهم

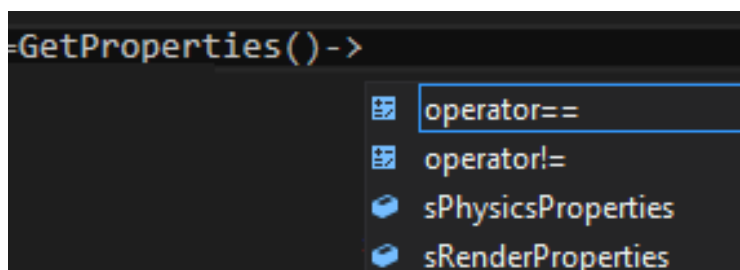
```
struct SItemProperties
{
    inline bool operator ==(const SItemProperties&
    rhs) const { return . == memcmp(this, &rhs,
    sizeof(rhs)); }
```

با استفاده از دستور کلیدی `inline` عمل تخصیص حافظه را به همراه ایجاد و ساخت متد براساس `operator` مساوی برای ساختار `SItemProperties` در نظر میگیریم

```
inline bool operator !=(const SItemProperties&
rhs) const { return · != memcmp(this, &rhs,
sizeof(rhs)); }
```

با استفاده از دستور کلیدی `inline` عمل تخصیص حافظه را به همراه ایجاد و ساخت متد براساس `operator` مخالف برای ساختار `SItemProperties` در نظر میگیریم

```
SRenderProperties sRenderProperties;
SPhysicsProperties sPhysicsProperties;
};
```



```
static void
ReflectType(Schematyc::CTypeDesc<SItemProperti
es>& desc)
{
desc.SetGUID("{۶۹۲۶۵B۳F-B۹۸۳-۴۷۹۹-۹AC۹-
۳۶۹۷۸۱۷E۵۱۸۹}"_cry_guid);
desc.SetLabel("Item Properties");
```



```
desc.SetDescription("Item Properties");
```

```
desc.AddMember(&SItemProperties::sRenderProperties, 'renp', "RenderProperties", "Render settings", "All Render settings of this item", SRenderProperties());
```

```
desc.AddMember(&SItemProperties::sPhysicsProperties, 'phyp', "PhysicsProperties", "Physics settings", "All physics settings of this item", SPhysicsProperties());
```

تابع `ReflectType` عمل انعکاس این کدها را در سندباکس به نمایش می گذارد، فضایی را برای بخش `Render Properties` و فضای دیگری برای بخش `Physics Properties` در نظر می گیرد تا `TextBox` ها یا `UpDownBox` را در پنجره `Properties` مقدار دهی کنید

```
}
```

هدر فایل `ItemComponent.h` :

این هدر فایل وظیفه اش پیاده سازی `entity` و یا `component` و همچنین توابع آن است که در حال ساخت آن در هدر فایل `ItemComponent.h` و فایل `ItemComponent.cpp` هستیم

```
#include
```

```
<CryEntitySystem/IEntityComponent.h>
```

```
#include "ItemProperties.h"
```

هدر فایل `ItemProperties.h` را به همراه هدر فایل `IEntityComponent.h` را اضافه می کنیم.

```
struct SItemComponent final :public
IEntityComponent
{

public:
SItemComponent()=default;
SItemComponent::~SItemComponent(){}

```

کلاسی با نام SItemComponent را تعریف کرده و سازنده و تخریب کننده آن کلاس را نیز تعریف می کنیم

```
virtual void Initialize() override;
virtual uint64 GetEventMask() const override;
virtual void ProcessEvent(SEntityEvent& event)
override;
static void
ReflectType(Schematyc::CTypeDesc<SItemComponen
t>& desc);

```

رویداد های سیستمی با نام تابع Initialize، GetEventMask، ProcessEvent، و ReflectType تعریف می کنیم، در فصل قبل و این فصل در رابطه با این رویدادها و مثال های مختلف آن را تشریح کرده ام

```
static void
RegisterItem(Schematyc::IEnvRegistrar&
registrator);

```

رویداد سیستمی RegisterItem عمل ثبت کلیه properties ها را در component و یا entity را انجام می دهد

```
SItemProperties *GetProperties() { return
&sItemProperties; }
```

این خصیصه از نوع اشاره گر با نام `GetProperties` به تمامی متغیرها و `Properties` های مختلف که در هدر فایل `ItemProperties.h` تعریف کردیم دسترسی دارد و مقدار مربوطه را براساس انتخاب در کد با کلمه کلیدی `return` و متغیر `sItemProperties` برمیگرداند

```
SItemProperties sItemProperties,
sPrevItemProperties;
```

دو متغیر با نام های `sItemProperties` و `sPrevItemProperties` را با ساختمان داده `SItemProperties` تعریف کرده ام که به انواع `Properties` هایی را که تعریف کرده ام دسترسی دارد

```
SRenderProperties sRenderProperties;
SPhysicsProperties sPhysicsProperties;
```

نوع `Properties` بر اساس متغیر `sRenderProperties` تعریف می کنم که به لود مدل سه بعدی و متریال برای مدل سه بعدی دسترسی دارد و همچنین نوع `Properties` بر اساس متغیر `sPhysicsProperties` تعریف می کنم که جرم جسم (مدل سه بعدی) دسترسی دارد

```
void LoadGeometry۲();
```

تابعی غیرسیستمی با نام LoadGeometry^۲ که لود مدل سه بعدی و اختصاص متریال به مدل سه بعدی را انجام می دهد

```
void Physicalize();
```

تابعی غیرسیستمی با نام Physicalize که عمل جاذبه را براساس عدد جرم جسم اختصاصی محاسبه و اجرا می کند

```
};
```

فایل کد ItemComponent.cpp :

حالا نوبت به آن رسیده که کلیه توابع و رویدادهای سیستمی و غیرسیستمی که در هدرفایل ItemComponent.h تعریف کردیم را در اینجا پیاده سازی کنیم

```
#pragma once
```

```
#include "StdAfx.h"
#include "ItemComponent.h"
#include "ItemProperties.h"
#include <urlmon.h>
```

هدر فایل های بالا مرتبط به کدهای نوشته شده توسط من و کدهایی را که کرای انجین به آن نیاز دارد را شامل می شود، هدر فایل های ItemComponent.h و ItemProperties.h را در چند صفحه قبل توضیح دادم

```
static void
RegisterItem(Schematyc::IEnvRegistrar&
registrar)
```

```

Schematyc::CenvRegistrationScope scope =
    registrar.Scope(Ientity::GetEntityScopeGUID())
;
{
Schematyc::CenvRegistrationScope
componentScope =
scope.Register(SCHEMATYC_MAKE_ENV_COMPONENT(SI
temComponent));
// Functions
{
}
}
}

```

با استفاده از تابع RegisterItem کلیه کدها براساس رابط کاربری سیماتیک و پنجره Properties لحاظ می شود، این کدها براساس ساختار تعریف شده با نام SItemComponent به صورت یک کامپونت یا اینتیتی با استفاده از ماکرو SCHEMATYC_MAKE_ENV_COMPONENT محاسبه و اجرا می شود و یک component یا entity جدید را ایجاد می کند

CRY_STATIC_AUTO_REGISTER_FUNCTION(&RegisterItem)

و در نهایت باید همه این کدها در هدر فایل هایی که نوشتم با ویژگی های مختلف با Properties های تعریف شده (RenderProperties و PhysicsProperties) در پنجره Properties نمایش داده شود، البته هر دو این Properties ها یعنی RenderProperties و PhysicsProperties زیر مجموعه ItemProperties هستند.

```
void SItemComponent::Initialize()
{
```

```
LoadGeometry۲();
```

با توجه به رویداد سیستمی `Initialize` که فقط برای یکبار اجرا می شود و در لحظه ساخت `component` و یا `entity` اجرا می شود، تابع `LoadGeometry۲` نوشته شده توسط برنامه نویس (غیرسیستمی) با نام `LoadGeometry۲` اجرا می شود

```
Physicalize();
```

با توجه به رویداد سیستمی `Initialize` که فقط برای یکبار اجرا می شود و در لحظه ساخت `component` و یا `entity` اجرا می شود، تابع `Physicalize` نوشته شده توسط برنامه نویس (غیرسیستمی) با نام `Physicalize` اجرا می شود

```
}
```

```
uint۶۴ SItemComponent::GetEventMask() const
{
return
BIT۶۴(ENTITY_EVENT_COMPONENT_PROPERTY_CHANGED);
}
```

از آنجایی که ممکن است مقادیر `textbox` ها یا `UpDownBox` در `Properties` ها تغییر کند، رویداد سیستمی `GetEventMask` به مکمل سازی این مهم می پردازد که عمل تغییر خصیصه ها در `properties` ها را با ثابت `ENTITY_EVENT_COMPONENT_PROPERTY_CHANGED` پردازش می نماید

```
void
SItemComponent::ProcessEvent(SEntityEvent&
event)
{
```

```
switch (event.event)
{
```

```
case ENTITY_EVENT_COMPONENT_PROPERTY_CHANGED :
{
```

با توجه به رویداد سیستمی `ProcessEvent` و نوع `event` و رفتار `event` با دستور `switch` بررسی می شود که آیا تغییر مقدار در `properties` ها با ثابت `case` در `ENTITY_EVENT_COMPONENT_PROPERTY_CHANGED` مورد اشاره اتفاق افتاده است؟ در اینجا یک بلوک `if` را تعریف کرده ام که شرح آن به این صورت است

```
if (sItemProperties != sPrevItemProperties)
{
```

در این بلوک `if` بررسی می شود که آیا متغیر `sItemProperties` مخالف متغیر `sPrevItemProperties` است ؟

اگر مخالف باشند یعنی `Operator!=` (در بالا شرح دادیم) اتفاق افتاده است پس دو تابع غیرسیستمی با نام های `LoadGeometry۲` و `Physicalize` اجرا می شوند اما اگر متغیر `sItemProperties` مساوی متغیر `sPrevItemProperties` باشد یعنی `Operator==` (در بالا شرح دادیم) اتفاق افتاده است و توابع `LoadGeometry۲` و `Physicalize` اجرا نمی شوند

```
LoadGeometry۲();
Physicalize();
}
```

```

}
break;
}
}

static void
ReflectType(Schematyc::CTypeDesc<SitemComponen
t>& desc)
{
    desc.SetGUID("{۹A۰۷۶CE۰-BB۳۸-۴FCA-AEF۰-
۵۷۳A۱۶B۶۱۰۰F}"_cry_guid);
    desc.SetLabel("My Item Component Ahmad Karami
😊");
    desc.SetDescription("Base item component");
}

```

حالا آخرین انعکاس با توجه به تابع سیستمی و استاتیک ReflectType اتفاق می افتد و همانطور که طبق تصویر زیر می بیند اسم component با نام My 😊 Item Component Ahmad Karami است و میدانم این اسم طولانی است اما متأسفانه در اینجا قانون کپی رایت رعایت نمی شود و مجبور شدم مرتباً از اسم خودم در بخش های مختلف استفاده کنم در هر صورت به شکل زیر نگاه کنید تا اسم کامپونت 😊 My Item Component Ahmad Karami را ببینید



```
desc.AddMember(&SItemComponent::sItemProperties, 'itep', "ItemProperties", "Item settings", "all properties of this item", SItemProperties());
}
```

و در نهایت ItemSettings اضافه می شود و دارای دو زیر مجموعه با نام های Render settings و Physics settings است

```
void SItemComponent::LoadGeometry2()
{
    string sGeometry =GetProperties()->sRenderProperties.sGeomPath.value;
```

حالا اگر دقت کنید خصیصه GetProperties به تمامی properties ها دسترسی دارد و از این طریق می توانید به متغیرها و مقادیر متغیرهای properties ها دسترسی داشته باشید، فراموش نکنید که متغیری که دارای textbox در داخل سندباکس است حتما از کلمه value برای آن استفاده کنید مانند sGeomPath.value و در نهایت مسیر فایل مدل سه بعدی برای لود در متغیر sGeometry قرار میگیرد

```
if(sGeometry.empty())
```

`return;`

در صورتی که متغیر `sGeometry` مقدارش خالی (`empty`) باشد یعنی مسیر فایل مدل سه بعدی به آن اختصاص داده نشود با استفاده از کلمه کلیدی `return;` کدهای دیگر در ادامه اجرا نخواهند شد، اگر مسیری برای فایل مدل سه بعدی در داخل `properties` لحاظ شود، مقدار متغیر `sGeometry` خالی نیست و بقیه کدها در پایین اجرا می شوند

```
m_pEntity->LoadGeometry(GEOMETRY_SLOT,
GetProperties()-
>sRenderProperties.sGeomPath.value);
```

این خط کد برای شما باید آشنا باشد، لود فایل سه بعدی را براساس `GEOMETRY_SLOT` در `enum` تعریف شده و مسیر فایل سه بعدی در `properties` را انجام می دهد

```
auto *pMaterial = gEnv->p3DEngine-
>GetMaterialManager()-
>LoadMaterial(GetProperties()-
>sRenderProperties.sMaterial.value);
```

این خط کد نیز یک متریال به مدل سه بعدی لود شده اختصاص می دهد، براساس `properties` تعریف شده به مسیر متریال دسترسی دارد و در داخل متغیر `pMaterial` ریخته می شود

```
m_pEntity->SetMaterial(pMaterial);
```

حالا با توجه به شی جاری یعنی `m_pEntity` متغیر متریال با نام `pMaterial` توسط تابع `SetMaterial` به مدل سه بعدی اختصاص داده می شود و بر روی مدل سه بعدی متریال مربوطه لود می شود

```
}
```

```
void SItemComponent::Physicalize()
{
```

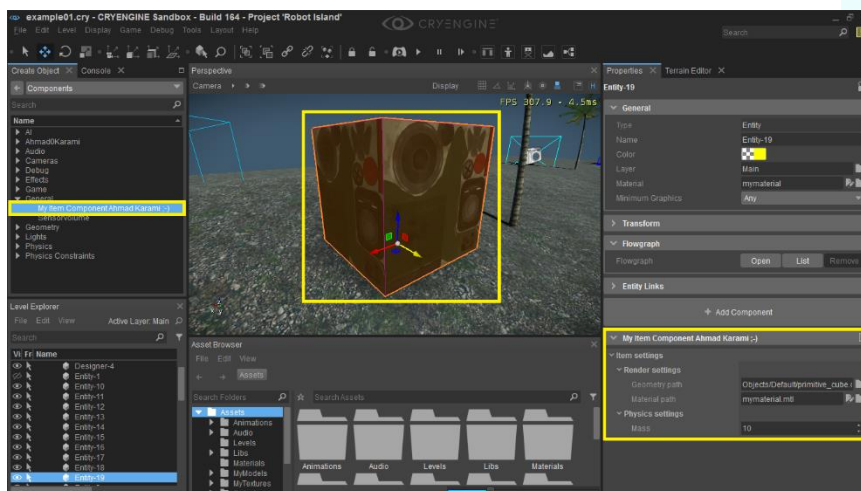
```
    SEntityPhysicalizeParams physParams;
    physParams.mass=GetProperties()-
    >sPhysicsProperties.fMass;
    physParams.type=PE_RIGID;
```

با توجه به متغیر تعریف شده `physParams` و با توجه به `properties` داده شده براساس متغیر `fMass`، نوع فیزیک شی به صورت `rigidbody` با ثابت `PE_RIGID` خواهد بود

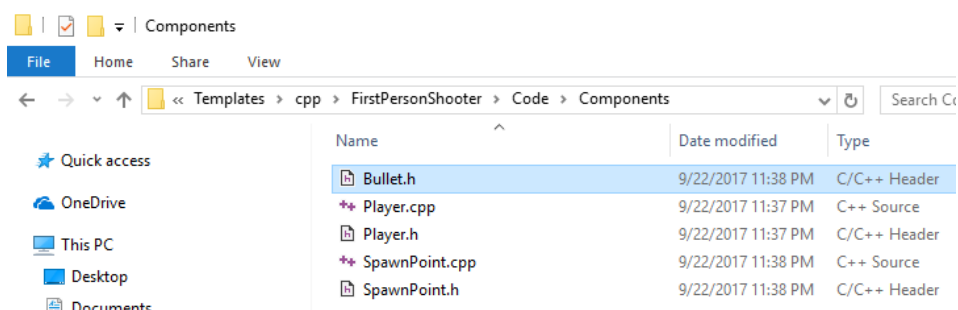
```
m_pEntity->Physicalize(physParams);
```

و در نهایت با توجه به شی جاری یعنی `m_pEntity` عمل جاذبه براساس جرم جسم (مدل سه بعدی) با آرگومان `physParams` محاسبه و اجرا شده و جسم به طرف بالا یا به طرف پایین می رود و یا در یکجا ثابت می ماند

```
}
```



آخرین موضوع این فصل مرتبط است با ساخت entity ها یا component هایی که فقط در بازی استفاده می شوند و برای ساخت هر Entity و استفاده در بازی (نه در زمان طراحی بازی یا نه در داخل ادیتور کرای انجین و در پنجره Object Create در بخش Components ها قابل دسترسی نیست) نیاز به حداقل یک هدر فایل (*.h) است، مانند هدر فایل Bullet.h در تمپلت FirstPersonShooter در تصویر زیر می بینید



سورس کد مربوط به هدر فایل Bullet.h در تمپلت FirstPersonShooter را در زیر مشاهده می کنید

`#pragma once`

برای جلوگیری از تعریف مجدد کلاس ها و برای جلوگیری از تعریف مجدد هدر فایل ها با پیش پردازنده include استفاده می شود، توصیه می شود برای ایجاد entity یا component در کرای انجین از این دستور استفاده کنید و فقط یکبار تعریف هدر فایل و فقط یکبار تعریف کلاس با این دستور انجام می شود

```
class CBulletComponent final : public
 IEntityComponent
```

```
{
    یک کلاس با نام CBulletComponent ارث بری کرده از رابط
    IEntityComponent را تعریف می کنیم
```

```
public:
virtual ~CBulletComponent() {}
```

تخریب کننده کلاس هیچ عملی را انجام نمی دهد زیرا داخل {} کدی وجود ندارد، همانطور که می دانید علامت ~ در کنار نام کلاس به تخریب کننده کلاس شهرت دارد

```
virtual void Initialize() override
{
```

رویداد سیستمی Initialize یکبار اجرا می شود و در هنگام اجرای بازی یا به هنگام drag & drop کردن entity یا component به داخل سندباکس رویداد Initialize اجرا می شود

```
const int geometrySlot = ۰;
```

ثابت مقداری geometrySlot تعریف می کنیم که برابر صفر است

```
m_pEntity->LoadGeometry(geometrySlot,
"Objects/Default/primitive_sphere.cgf");
```

شی جاری m_pEntity یک مدل سه بعدی با نام primitive_sphere.cgf را در slot۰ لود می کند (یک توپ لود می شود)

```
auto *pBulletMaterial = gEnv->p3DEngine-
>GetMaterialManager()-
>LoadMaterial("Materials/bullet");
```

یک متغیر با نام pBulletMaterial تعریف شده است که برای لود یک متریال با نام bullet مقداردهی می شود

```
m_pEntity->SetMaterial(pBulletMaterial);
```

شی جاری m_pEntity متریال لود شده را بر روی مدل سه بعدی (توپ) در slot۰ قرار می دهد

```
SEntityPhysicalizeParams physParams;
```

یک متغیر با نام physParams از نوع داده SEntityPhysicalizeParams تعریف می شود

```
physParams.type = PE_RIGID;
```

متغیر physParams به نوع rigidbody با ثابت PE_RIGID در نظر گرفته می شود

```
physParams.mass = ۲۰۰۰۰.f;
```

متغیر physParams به فیلد mass ۲۰۰۰۰ گرم مقداردهی می شود

```
m_pEntity->Physicalize(physParams);
```

متغیر physParams برروی شی جاری m_pEntity لحاظ شده و مدل سه بعدی از آنجایی که دارای rigidbody و ۲۰۰۰۰ گرم است، تحت تاثیر جاذبه در بازی سقوط می کند و مدل سه بعدی یا همان توپ به طرف زمین و رو به پایین می افتد.

```
GetEntity()->SetViewDistRatio(۲۵۵);
```

این توپ با توجه به تابع SetViewDistRatio تا فاصله ۲۵۵ متری قابل دیدن است.

```
if (auto *pPhysics = GetEntity()->GetPhysics())
{
```

طبق وجود بلوک if، شی جاری GetEntity یا همان توپ از آنجایی که دارای فیزیک است، خطوط بلوک if اجرا می شود.

```
pe_action_impulse impulseAction;
```

یک متغیر با نام impulseAction از نوع داده pe_action_impulse تعریف شده است.

```
const float initialVelocity = ۱۰۰۰.f;
```

ثابت `initialVelocity` نیروی به اندازه ۱۰۰۰ نیوتن دارد

```
impulseAction.impulse = GetEntity()-
>GetWorldRotation().GetColumn\()*
initialVelocity;
```

طبق متغیر `impulseAction` با فیلد `impulse` باید ضربه ای به توپ وارد شود، از آنجایی که `GetEntity` به شی جاری یا همان توپ اشاره دارد و میزان چرخش توپ توسط تابع `GetWorldRotation` برگشت داده می شود و جهت حرکت توپ به طرف عمق (رو به جلو) با تابع `GetColumn\` محاسبه می شود و در ثابت `initialVelocity` که دارای ۱۰۰۰ نیوتن است، ضرب می شود.

```
pPhysics->Action(&impulseAction);
```

و طبق تابع `Action` از متغیر گرفته شده `pPhysics` از شی جاری `GetEntity`، توپ با نیرویی به شدت ۱۰۰۰ نیوتن به طرف جلو پرت می شود.

```
}
}
```

```
static void
ReflectType(Schematyc::CTypeDesc<CBulletCompon
ent>& desc)
{
desc.SetGUID("{B۵۳A۹A۵F-F۲۷A-۴۲CB-۸۲C۷-
B۱E۳۷۹C۴۱A۲A}"_cry_guid);
}
```


همانطور که می بینید کلاس `CBulletComponent` در تابع استاتیک `ReflectType` انعکاس یافته است اما این انعکاس تنها در داخل بازی است، به این معنا که در سندباکس کرای انجین و در پنجره `Create Object` در بخش `Components` هیچ `entity` یا `component` ی با نام `bullet` وجود ندارد، زیرا که تنها یک خط با عنوان `desc.SetGUID` در تابع انعکاسی `ReflectType` موجود است و همچنین یک فایل با نام `Bullet.cpp` در تمپلیت `FirstPersonShooter` وجود ندارد.

```
virtual uint64 GetEventMask() const override {
return BIT64(ENTITY_EVENT_COLLISION); }
```

رویداد سیستمی `GetEventMask` با توجه به وجود ماکرو بیتی با ثابت `ENTITY_EVENT_COLLISION` عمل برخورد توپ به یک شی سفت و سخت را ماسک گذاری می کند

```
virtual void ProcessEvent(SEntityEvent &
event) override
{
```

با توجه به رویداد سیستمی `GetEventMask` که عمل ماسک گذاری را انجام داده و این عمل به رویداد سیستمی `ProcessEvent` ارسال می شود، در بلوک `if` زیر بررسی می شود که آیا توپ به شی سخت یا سطوح سفت و سخت یا به زمین برخورد کرده است؟ اگر این عمل برخورد توپ با شی سفت و سخت دیگری اتفاق افتاده باشد، خطوط کد زیر در بلوک `if` اجرا می شود.

```

if (event.event == ENTITY_EVENT_COLLISION)
{
gEnv->pEntitySystem-
>RemoveEntity(GetEntityId());

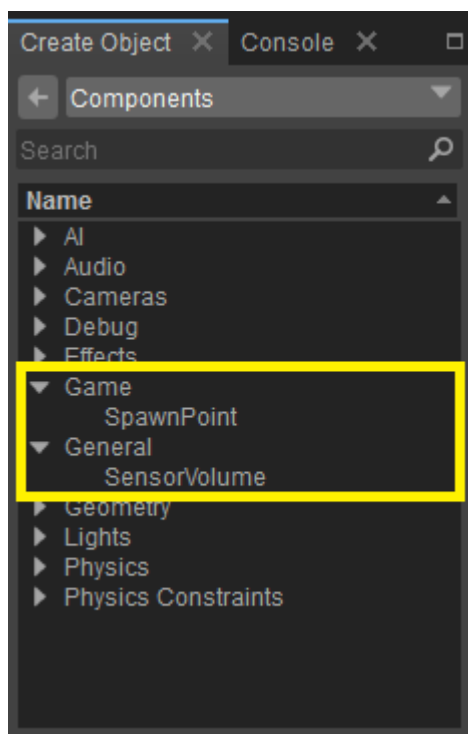
```

با توجه به آن که توپ به شی سفت و سختی برخورد کرده است و طبق تابع `GetEntityId`، این همان Id توپ است که توپ همان شی جاری است که پرتاب شده است و با توجه به تابع `RemoveEntity` توپ تاثیر نیروی خود را برروی شی سفت و سخت ایجاد می کند (در کدهایی بالا بررسی کردیم که توپ نیرو دارد و...) و در نهایت توپ از بین می رود

```

}
}
};

```



ساخت یک component یا entity مانند آنچه که در بالا به آن اشاره شد، فقط در بازی استفاده می شود و به وسیله فقط یک هدر فایل ساخته می شود و در طراحی مرحله نمی توان به آن دسترسی داشت اما به تشریح آن پرداختیم تا مطالب را کامل پوشش داده باشم، شما سعی کنید همیشه entity یا component ی را که می سازید در داخل سندباکس در پنجره Create Object در بخش Components قابل دسترسی باشد، پس از یک هدر فایل و یک سی پی پی فایل استفاده کنید، این کار باعث می شود که طراحی مرحله و استفاده از آن در بازی آسانتر باشد و اگر نمی خواهید entity یا component تان در سندباکس در پنجره Create Object در بخش Components قابل دسترسی باشد، فقط یک هدر فایل ایجاد کنید و به این صورت تنها در بازی می توانید به entity یا component که ساختید دسترسی داشته باشید، در تصویر بالا می بینید که در پنجره Create Object در بخش Components، گزینه یا option ی برای bullet (تویی که الان ساختیم) وجود ندارد و در سندباکس نمی توان به تویی که ساختیم دسترسی داشته باشیم زیرا که فایل bullet.cpp و عمل انعکاس را برای تویی (bullet) که الان ساختیم با ثبت در سندباکس انجام نشده است

ثابت های رویدادی

حالا برخی از ثابت های رویدادی که در توابع سیستمی GetEventMask و ProcessEvent کاربرد دارند و در هدر فایل IEntityBasicTypes.h را تشریح می کنم :

```
enum EEntityEvent
{
```

```
ENTITY_EVENT_XFORM = ۰,
```

یک enum با نام EEntityEvent تعریف شده است که از ثابت های رویدادی زیر تشکیل شده است و وقتی که entity ماتریکس انتقالش به صورت محلی یا سراسری با مدنظر قرار دادن موقعیت، چرخش، مقیاس تغییر کند، این رویداد قابل استفاده خواهد بود و به وسیله [۰] nParam ترکیبی از پرچم های با نوع EEntityXFormFlags خواهد بود

```
ENTITY_EVENT_UPDATE,
```

این رویداد برای به روز رسانی فریم ها در یک entity با توجه به [۰] nParam و ساختار SEntityUpdateContext انجام می شود.

```
ENTITY_EVENT_XFORM_FINISHED_EDITOR,
```

این رویداد وقتی که entity در داخل ادیتور حرکت داد شود، چرخش داده شود یا مقیاسش تغییر کند اتفاق می افتد و از طریق mouseButtonUp این عمل انجام می شود

```
ENTITY_EVENT_TIMER,
```

هنگامی که تایمر entity منقضی می شود، رویداد براساس [۰] nParam برای Id تایمر و [۱] nParam برای میلی ثانیه اتفاق می افتد.

```
ENTITY_EVENT_INIT,
```

وقتی که **entity** غیر قابل حذف بوده و تکثیر مجدد می شود، این رویداد اتفاق می افتد.

ENTITY_EVENT_DONE,

این رویداد هنگامی که **entity** حذف می شود کاربرد دارد.

ENTITY_EVENT_VISIBILITY,

این رویداد هنگامی که **entity** مرئی یا نامرئی می شود کاربرد دارد و برای مرئی شدن **nParam[۰]** با مقدار یک و برای نامرئی شدن **nParam[۰]** با مقدار صفر خواهد شد .

ENTITY_EVENT_RESET,

این رویداد وضعیت **entity** را در ادیتور **reset** می کند، یعنی هنگامی که در **game mode** (داخل بازی) هستیم **nParam[۰]** مقدارش یک و هنگامی که از **game mode** خارج می شویم **nParam[۰]** مقدارش صفر خواهد شد.

ENTITY_EVENT_ATTACH,

بعد از اینکه **child entity** اضافه شد برای **parent entity**، این رویداد فراخوانی می شود و **nParam[۰]** مقدارش همان **Id** مربوط به **child entity** خواهد بود

ENTITY_EVENT_ATTACH_THIS,

بعد از اینکه برای `parent entity` اضافه شد برای `child entity`، این رویداد فراخوانی می شود و `nParam[۰]` مقدارش همان `Id` مربوط به `parent entity` خواهد بود

`ENTITY_EVENT_DETACH,`

بعد از اینکه `child entity` حذف شد برای `parent entity`، این رویداد فراخوانی می شود و `nParam[۰]` مقدارش همان `Id` مربوط به `child entity` خواهد بود

`ENTITY_EVENT_DETACH_THIS,`

بعد از اینکه از `parent entity` حذف شد برای `child entity`، این رویداد فراخوانی می شود و `nParam[۰]` مقدارش همان `Id` مربوط به `parent entity` خواهد بود

`ENTITY_EVENT_LINK,`

بعد از اینکه برای `child entity` لینک شد برای `parent entity`، این رویداد فراخوانی می شود و `nParam[۰]` مقدارش به `IEntityLink` اشاره می کند

`ENTITY_EVENT_DELINK,`

قبل از اینکه `child entity` لینکش حذف شود برای `parent entity`، این رویداد فراخوانی می شود و `nParam[۰]` مقدارش به `IEntityLink` اشاره می کند

ENTITY_EVENT_HIDE,

هنگامی که entity می بایست مخفی شده باشد، این رویداد فراخوانی می شود

ENTITY_EVENT_UNHIDE,

هنگامی که entity نبایستی مخفی شده باشد، این رویداد فراخوانی می شود

ENTITY_EVENT_LAYER_HIDE,

هنگامی که entity می بایست در سطح لایه مخفی شده باشد، این رویداد فراخوانی می شود

ENTITY_EVENT_LAYER_UNHIDE,

هنگامی که entity نبایستی در سطح لایه مخفی شده باشد، این رویداد فراخوانی می شود

ENTITY_EVENT_ENABLE_PHYSICS,

هنگامی که فرآیند شبیه سازی قوانین فیزیک فعال می شود یا غیر فعال می شود، این رویداد اجرا می شود و [۰] nParam مقدارش یک باشد یعنی قوانین فیزیک فعال هستند و [۰] nParam مقدارش صفر باشد یعنی قوانین فیزیک غیر فعال هستند

ENTITY_EVENT_PHYSICS_CHANGE_STATE,

وقتی که entity حامل کامپوننت های فیزیک است، وقتی که وضعیت آن تغییر می کند، این رویداد فراخوانی می شود و اگر `nParam[۰]` مقدارش یک باشد، entity قوانین فیزیکش از سرگیری مجدد و بیدار شده برای اجرا شدن و اگر `nParam[۰]` مقدارش صفر باشد، entity قوانین فیزیکش به خواب رفته و غیرقابل اجرا می شود

ENTITY_EVENT_ENTERAREA,

یک ناحیه سه بعدی برای تریگر شدن تعریف می کنیم و آن ناحیه تریگری از فضای سه بعدی در entity منتظر وارد شدن entity دیگری است که به داخل ناحیه تریگری فرو رود، پس با وارد شدن entity به داخل ناحیه تریگری این رویداد فراخوانی می شود و `nParam[۰]` مقدارش Id entity که به تریگر وارد شده و `nParam[۱]` مقدارش Id ناحیه است و `nParam[۲]` به EntityId شی جاری اشاره دارد

ENTITY_EVENT_LEAVEAREA,

یک ناحیه سه بعدی برای تریگر شدن تعریف می کنیم و آن ناحیه تریگری از فضای سه بعدی در entity منتظر خارج شدن entity دیگری است که به داخل ناحیه تریگری فرو رفته بود، پس با خارج شدن entity از ناحیه تریگری این رویداد فراخوانی می شود و `nParam[۰]` مقدارش Id entity که از تریگر خارج شده و `nParam[۱]` مقدارش Id ناحیه است و `nParam[۲]` به EntityId شی جاری اشاره دارد

ENTITY_EVENT_ENTERNEARAREA,

یک ناحیه سه بعدی برای تریگر شدن تعریف می کنیم و آن ناحیه تریگری از فضای سه بعدی در entity منتظر نزدیک شدن است، یک entity دیگر است و با نزدیک شدن به ناحیه تریگری این رویداد فراخوانی می شود و `nParam[۰]` مقدارش Id entity است که به ناحیه تریگری نزدیک

شده است و `nParam[۱]` مقدارش `Id` ناحیه است و `nParam[۲]` به `EntityId` شی جاری اشاره دارد و `fParam[۰]` مقدارش میزان فاصله `entity` در حال نزدیک شدن به ناحیه تریگری است.

`ENTITY_EVENT_LEAVENEARAREA,`

یک ناحیه سه بعدی برای تریگر شدن تعریف می کنیم و آن ناحیه تریگری از فضای سه بعدی در `entity` منتظر خارج شدن است که به ناحیه تریگری نزدیک شده بود، پس با خارج شدن `entity` از ناحیه تریگری `entity` نزدیک شده این رویداد فراخوانی می شود و `nParam[۰]` مقدارش `Id` `entity` که از تریگر خارج شده و `nParam[۱]` مقدارش `Id` ناحیه است و `nParam[۲]` به `EntityId` شی جاری اشاره دارد.

`ENTITY_EVENT_MOVEINSIDEAREA,`

یک ناحیه سه بعدی برای تریگر شدن تعریف می کنیم و آن ناحیه تریگری از فضای سه بعدی منتظر حرکت `entity` دیگر داخل ناحیه تریگری است و با این عمل رویداد فراخوانی می شود و `nParam[۰]` مقدارش `Id` `entity` داخل آن حرکت کرده و `nParam[۱]` مقدارش `Id` ناحیه است و `nParam[۲]` به `EntityId` شی جاری اشاره دارد.

`ENTITY_EVENT_AI_DONE,`

`entity` که دارای کامبونت هوش مصنوعی است و `action` ی اجرا شد و پایان یافت، این رویداد فراخوانی می شود.

`ENTITY_EVENT_COLLISION,`

وقتی که entity با بدنه سفت و سخت با entity دیگری که دارای بدنه سفت و سخت باشد، برخورد می کند، این رویداد اجرا می شود.

ENTITY_EVENT_PREPHYSICSUPDATE,

از آنجایی که محاسبات فیزیک و اجرای کدهای گرافیکی در نمایش فیزیک به صورت بصری در بازی می تواند باعث افت شدید frame-rate شود، نیاز است که از این رویداد استفاده کنیم و در [۰] fParam میزان فریم زمانی مقداردهی می شود.

ENTITY_EVENT_LEVEL_LOADED,

وقتی که مرحله کامل لود می شود، این رویداد باید استفاده شود

ENTITY_EVENT_START_LEVEL,

هنگامی که مرحله شروع شد، از این رویداد باید استفاده شود

ENTITY_EVENT_MATERIAL,

وقتی که متریال entity تغییر می کند، این رویداد اتفاق می افتد و [۰] nParam دارای اشاره گری از IMaterial جدید است.

ENTITY_EVENT_MATERIAL_LAYER,

وقتی که لایه ماسک متریال ها تغییر می کند، این رویداد اجرا می شود.

ENTITY_EVENT_ANIM_EVENT,

وقتی که در داخل ادیتور یک رویداد انیمیشنی اجرا می شود در [۰] nParam یک متغیر با نام pEventParameters از نوع AnimEventInstance قابل آدرس دهی خواهد بود.

ENTITY_EVENT_EDITOR_PROPERTY_CHANGED,

وقتی که در داخل ادیتور یک اسکریپت Lua خصوصیاتش در **entity** عوض می شود، این رویداد فراخوانی می شود، این رویداد برای **IEntityPropertyGroup** نخواهد بود

```
ENTITY_EVENT_COMPONENT_PROPERTY_CHANGED =  
ENTITY_EVENT_EDITOR_PROPERTY_CHANGED,
```

وقتی که خصیصه ها و مقادیر در کامپونت در داخل **entity** تغییر کند، باید از این رویداد استفاده کنید و **nParam[۰]** دارای اشاره گری از **IEntityComponent** است یا به **nullptr** اشاره دارد و **nParam[۱]** به عضوی از **Id** از **Property** تغییر یافته دسترسی دارد.

```
ENTITY_EVENT_RELOAD_SCRIPT,
```

هنگامی که یک اسکریپت در حال لود مجدد است و درخواست لود مجدد انجام می شود به شرط آنکه در داخل ادیتور این اتفاق بیفتد، از این رویداد استفاده می شود.

```
ENTITY_EVENT_ACTIVATED,
```

وقتی که **entity** ی به لیست از **entity** ها اضافه شود، این رویداد اتفاق می افتد.

```
ENTITY_EVENT_DEACTIVATED,
```

وقتی که **entity** ی از لیست **entity** ها حذف شود، این رویداد اتفاق می افتد.

```
ENTITY_EVENT_NET_BECOME_LOCAL_PLAYER,
```

وقتی که یک پلیمر ویژه تکثیر یا ایجاد می شود به صورت `client`، یکبار این عمل اتفاق می افتد.

`ENTITY_FLAG_LOCAL_PLAYER`

وقتی که پلیمر ایجاد یا تکثیر می شود به صورت `client` یکبار این عمل اتفاق می افتد.

`ENTITY_EVENT_ADD_TO_RADAR,`

وقتی که باید `entity` به رادار اضافه شود، این رویداد اتفاق می افتد.

`ENTITY_EVENT_REMOVE_FROM_RADAR,`

وقتی که باید `entity` از رادار حذف شود، این رویداد اتفاق می افتد.

`ENTITY_EVENT_SET_NAME,`

وقتی که اسم `entity` مقداردهی شود، این رویداد باید استفاده شود.

`ENTITY_EVENT_AUDIO_TRIGGER_STARTED,`

وقتی که یک تریگر صوتی شروع به پخش شدن می کند، از این رویداد باید استفاده کرد.

`ENTITY_EVENT_AUDIO_TRIGGER_ENDED,`

وقتی که یک تریگر صوتی پایان می یابد یا با شکست مواجه می شود، از این رویداد باید استفاده کرد

<code>const</code>	<code>nParam[.]</code> دارای محتوای ثابت دوگانه
<code>const</code>	<code>*SRequestInfo</code> CryAudio:: است

ENTITY_EVENT_SLOT_CHANGED,

وقتی که یک **entity** اسلاتش تغییر می کند مثلاً یک مدل هندسی به آن اضافه شود، این رویداد فراخوانی می شود و شماره اسلات در `nParam[۰]` ذخیره می شود.

ENTITY_EVENT_PHYSICAL_TYPE_CHANGED,

وقتی که نوع فیزیک یک **entity** تغییر می کند، مثلاً فعال سازی یا غیرفعال سازی بدنه فیزیک در مواجهه با نوع تغییر رفتار فیزیک در رابطه با اشیاء مختلف، این رویداد فراخوانی می شود.

ENTITY_EVENT_LAST,

اگر هیچ کدام از رویدادهای بالا فراخوانی نشدند، این آخرین رویدادی است که می توان فراخوانی کرد

};

فصل هشتم

ساخت یک پلیر جدید برای بازی

الان نوبت به بررسی دو فایل سورس با نام های Player.h و Player.cpp می رسد، این دو فایل دارای خطوط کد بسیار طولانی هستند و برای ساخت یک پلیر یا همان قهرمان داستان کاربرد دارند، خطوط کدهای آن دو فایل را بررسی می کنم، کلیه مثال ها بر روی DVD در ضمیمه این کتاب موجود است.

فایل Player.h به شرح زیر است :

```
#pragma once
```

```
#include <array>
```

این خط کد تعریف آرایه ها و ساختمان داده های مختلف را برعهده دارد

```
#include <numeric>
```

این خط کد ساختارهای عددی و استفاده از روش های جدید با مفهوم های مختلف را تعریف می کند

```
#include <CryEntitySystem/IEntityComponent.h>
```

این خط کد سیستم جدید EntityComponent را تعریف می کند تا بتواند از کامپونت های مختلف برای ایجاد entity در سطح پلیر استفاده کند

```
#include <CryMath/Cry_Camera.h>
```

محاسبات مربوط به دوربین و توابع مختلف در ارتباط با مدیریت دوربین را تعریف می کند

```
#include <ICryMannequin.h>
```

سیستم مانکن ادیتور را در کرای انجین برای استفاده از انیمیشن های مختلف برای کاراکترها کاربرد دارد و در این خط کد دسترسی به سیستم مانکن و انیمیشن کاراکترها انجام می شود

```
#include  
<DefaultComponents/Cameras/CameraComponent.h>
```

ایجاد کامبونت دوربین با این خط کد امکان پذیر می شود

```
#include  
<DefaultComponents/Physics/CharacterController  
Component.h>
```

ایجاد کامبونتی برای حرکت پلیر و محاسبات فیزیک بدنه پلیر با این خط کد امکان پذیر می شود

```
#include  
<DefaultComponents/Geometry/AdvancedAnimationC  
omponent.h>
```

ایجاد کامبونت برای انیمیشن های پیشرفته در سطوح مختلف با این خط کد امکان پذیر می شود

```
#include  
<DefaultComponents/Input/InputComponent.h>
```


ایجاد کامپونت دستگاه های ورودی مانند صفحه کلید و ماوس با این خط کد امکان پذیر می شود

```
class CPlayerComponent final : public
IEntityComponent
{
```

یک کلاس نهایی فرزند با نام CPlayerComponent از رابط IEntityComponent را ارث بری و تعریف می کند

```
enum class EInputFlagType
{
Hold = ۰,
Toggle
};
```

در این چند خط کد نوعی enum را تعریف می کنیم که در سطح کلاس تعریف شده باشد و برای حرکت پلیر بوسیله صفحه کلید کاربرد دارد

```
typedef uint۸ TInputFlags;
```

در اینجا نوع داده ۸ بیتی بدون علامت با عنوان یک پرچم ورودی از این به بعد تعریف شده و با نام TInputFlags شناخته شده خواهد بود

```
enum class EInputFlag
: TInputFlags
{
MoveLeft = ۱ << ۰,
MoveRight = ۱ << ۱,
MoveForward = ۱ << ۲,
```

```
MoveBack = ۱ << ۳
};
```

ساختار مشتق شده از پرچم ورودی حرکت پلیر با صفحه کلید تعریف می شود که حرکت های به سمت چپ، به سمت راست، به سمت جلو، به سمت عقب با محاسبات بیتی انجام می پذیرد

```
template<typename T, size_t SAMPLES_COUNT>
class MovingAverage
```

اینجا کلاسی تعریف شده و الگوبرداری می شود که حرکت را طبق توابع محاسباتی در صفحه کلید و در ماوس برای دوربین پلیر و حرکت پلیر در نظر می گیرد

```
{
static_assert(SAMPLES_COUNT > ۰,
"SAMPLES_COUNT shall be larger than zero!");
```

```
public:
```

```
MovingAverage()
: m_values()
, m_cursor(SAMPLES_COUNT)
, m_accumulator()
{
}
```

```
MovingAverage& Push(const T& value)
{
if (m_cursor == SAMPLES_COUNT)
{
```

```
m_values.fill(value);
m_cursor = 0;
m_accumulator =
std::accumulate(m_values.begin(),
m_values.end(), T(0));
}
else
{
m_accumulator -= m_values[m_cursor];
m_values[m_cursor] = value;
m_accumulator += m_values[m_cursor];
m_cursor = (m_cursor + 1) % SAMPLES_COUNT;
}

return *this;
}

T Get() const
{
return m_accumulator / T(SAMPLES_COUNT);
}

void Reset()
{
m_cursor = SAMPLES_COUNT;
}

private:

std::array<T, SAMPLES_COUNT> m_values;
size_t m_cursor;
```

```
T m_accumulator;
};
```

چند تابع بالا حرکت ماوس را برای بازی شبیه سازی می کند و کنترل زاویه ها را بر عهده دارد، این توابع شتاب حرکت ماوس را نیز برعهده دارند

```
public:
```

```
CPlayerComponent() = default;
```

کلاس برای سازنده مقدار پیش فرضی را تعریف می کند

```
virtual ~CPlayerComponent() {}
```

کلاس تخریب کننده تعریف شده اما محاسباتی در داخل خود ندارد

```
virtual void Initialize() override;
```

این رویداد را قبلا بررسی کردم و یکبار اجرا می شود

```
virtual uint64 GetEventMask() const override;
```

این رویداد را قبلا بررسی کردم و عمل ماسک گذاری رویدادهای سیستمی را انجام می دهد

```
virtual void ProcessEvent(SEntityEvent& event)
override;
```

این رویداد را قبلا بررسی کردم و عمل ماسک گذاری شده را پردازش می کند

```
static void
```

```
ReflectType(Schematyc::CTypeDesc<CPlayerCompon
ent>& desc)
```

```
{
```

```
desc.SetGUID("{۶۳F۴C۰C۶-۳۲AF-۴ACB-۸FB۰-
۵۷D۴۵DD۱۴۷۲۵}"_cry_guid);
```

```
desc.SetEditorCategory("Game");
```

```
desc.SetLabel("MyPlayer");
```

```
desc.SetDescription("This Player , you can  
use");  
desc.SetComponentFlags({  
    IEntityComponent::EFlags::Transform,  
    IEntityComponent::EFlags::Socket,  
    IEntityComponent::EFlags::Attach });  
}
```

این تابع را قبلا بررسی کردم و عمل انعکاس کامبونت یا اینتیتی را در
سندباکس نمایش می دهد

```
void Revive();
```

احیا کردن و صفر کردن متغیرها و پارامترهایی که به پلیر وابسته بوده را
مجددا محاسبه و مقداردهی می کند

```
protected:
```

```
void UpdateMovementRequest(float frameTime);
```

این تابع حرکت پلیر را به روزرسانی می کند

```
void UpdateLookDirectionRequest(float  
frameTime);
```

این تابع جهت پلیر را به روزرسانی می کند

```
void UpdateAnimation(float frameTime);
```

این تابع انیمیشن های پلیر را در رابطه با سیستم مانکن ادیتور به روزرسانی
می کند

```
void UpdateCamera(float frameTime);
```

این تابع چرخش دوربین پلیر را به همراه پردازش دوربین پلیر را به روزرسانی ، محاسبه و اجرا می کند

```
void OnUpdate(float deltaTime);
```

این تابع عملیات دیگری از پلیر را به روزرسانی می کند.

```
void SpawnAtSpawnPoint();
```

این تابع تعریف شده در رابطه با نقطه ای است که پلیر از آنجا gameplay بازی را شروع می کند.

```
void FKey();
```

هنگامی که دکمه F بر روی صفحه کلید زده شد، از این تابع استفاده می شود

```
void GKey();
```

هنگامی که دکمه G بر روی صفحه کلید زده شد، از این تابع استفاده می شود

```
void HKey();
```

هنگامی که دکمه H بر روی صفحه کلید زده شد، از این تابع استفاده می شود

```
void IKey();
```

هنگامی که دکمه I بر روی صفحه کلید زده شد، از این تابع استفاده می شود

```
void OKey();
```

هنگامی که دکمه O بر روی صفحه کلید زده شد، از این تابع استفاده می شود

```
void JKey();
```

هنگامی که دکمه J بر روی صفحه کلید زده شد، از این تابع استفاده می شود

```
void ESCKey();
```

هنگامی که دکمه **Escape** بر روی صفحه کلید زده شد، از این تابع استفاده می شود.

```
void YMouse();
```

حرکت عمودی برای ماوس با این تابع تعریف می شود

```
void XMouse();
```

حرکت افقی برای ماوس با این تابع تعریف می شود

```
void MoveBack();
```

حرکت به سمت عقب برای پلیر تعریف می شود

```
void MoveForward();
```

حرکت به سمت جلو برای پلیر تعریف می شود

```
void MoveRight();
```

حرکت به سمت راست برای پلیر تعریف می شود

```
void MoveLeft();
```

حرکت به سمت چپ برای پلیر تعریف می شود

```
void DefineInput();
```

کامبونت دستگاه های ورودی را با در نظر گرفتن یک متغیر تعریف شده در این تابع فعال می شود

```
void DefineAnimationModel();
```

با توجه به متغیر تعریف شده داخل این تابع برای نمایش دادن انیمیشن های کاراکتر و پیدا کردن فایل های سه بعدی کاراکتر برای ایجاد پلیر از این تابع استفاده می شود.

```
void LoadAnimationClips();
```

انیمیش های `Idle` ، `Walk` و چرخش کاراکتر را به متغیرهای مربوطه در این تابع نسبت می دهد

```
void DefineCharacterController();
```

ایجاد کامبونت برای بدنه فیزیک پلیر با نام `Character` و اختصاص محاسبات ریاضی به آن `Controller`

```
void DefineCameraPlayer();
```

ایجاد کامبونت `Camera` و متغیر تعریف شده در این تابع اختصاص داده می شود.

```
void ControlFKey(float deltaTime);
```

کنترل شلیک ها از طریق دکمه `F` برروی صفحه کلید با تاخیرهای زمانی را برعهده دارد.

```
void SpaceKey();
```

عمل پرش پلیر را بر اساس جهت حرکت انجام می دهد

```
void AllShiftKey();
```

سرعت حرکت پلیر با گرفتن و رها کردن دکمه های `Shift` برروی صفحه کلید افزایش و کاهش می یابد

```
void ShowWeaponAmmo();
```


نمایش رابط کاربری بسیار ساده برای نمایش دادن میزان خرابی اسلحه های موجود

```
void SimulateExplosionRigidbody();
```

شبیه سازی عمل انفجار به صورت محاسبات فیزیک بر روی اشیاء ی که دارای Rigidbody هستند

```
void SpawnParticleSystemHit();
```

با توجه به نقطه برخورد Hit عمل تکثیر پارتیکل سیستم انجام می شود و این تابع شماره ۱ است

```
void SpawnParticleSystemHit۲();
```

با توجه به نقطه برخورد Hit عمل تکثیر پارتیکل سیستم انجام می شود و این تابع شماره ۲ است، این تابع در بازی استفاده نشده است

```
void SpawnParticleSystemHit۳();
```

با توجه به نقطه برخورد Hit عمل تکثیر پارتیکل سیستم انجام می شود و این تابع شماره ۳ است

```
void HidePlayer(bool hp);
```

پلیر را می توان مخفی یا نشان داد، برای تمرین می توانید یک نوع اسلحه با مقدار خرابی مشخص به بازی اضافه کنید که پلیر با دریافت آن مخفی شود و با گذشت مثلا ۱۵ ثانیه دوباره پلیر ظاهر شود

```
void SetPlayerOnceInit();
```

یکبار تغییر موقعیت پلیر انجام می شود

```
void AKShooting();
```

در حال حاضر این تابع می تواند رفتار کلاشینکف یا اسلحه آتش زا و یا حتی مسلسل را شبیه سازی کند، در این پروژه اسلحه آتش زا شبیه سازی شده است

```
void CreateWeapon(const char *name);
```

این تابع برای ساخت انواع اسلحه استفاده شود، من از این تابع استفاده نکرده ام و این یک تابع خالی از دستورات است، این به صورت پیش فرض در داخل تمپلت کرای انجین ۵،۴ در کدهای C++ موجود است

```
void HandleInputFlagChange(TInputFlags flags,  
int activationMode, EInputFlagType type =  
EInputFlagType::Hold);
```

نحوه نگه داشتن و رها کردن دکمه های صفحه کلید با این تابع پیاده سازی می شود

```
protected:  
Cry::DefaultComponents::CCameraComponent*  
m_pCameraComponent = nullptr;
```

یک متغیر از نوع کامپوننت دوربین با نام m_pCameraComponent برای پلیر تعریف می می شود

```
Cry::DefaultComponents::CCharacterControllerCo  
mponent* m_pCharacterController = nullptr;
```

یک متغیر از نوع کامپونت `CharacterController` با نام `m_pCharacterController` برای پلیر تعریف می شود

```
Cry::DefaultComponents::CAdvancedAnimationComponent* m_pAnimationComponent = nullptr;
```

یک متغیر از نوع کامپونت انیمیشن با نام `m_pAnimationComponent` برای پلیر تعریف می شود.

```
Cry::DefaultComponents::CInputComponent* m_pInputComponent = nullptr;
```

یک متغیر از نوع کامپونت ورودی دستگاه های صفحه کلید و ماوس با نام `m_pInputComponent` برای پلیر تعریف می شود.

```
FragmentID m_idleFragmentId;
```

یک متغیر با نام `m_idleFragmentId` تعریف می کند که آدرس قطعه انیمیشن `Idle` را باید به آن اختصاص داد.

```
FragmentID m_walkFragmentId;
```

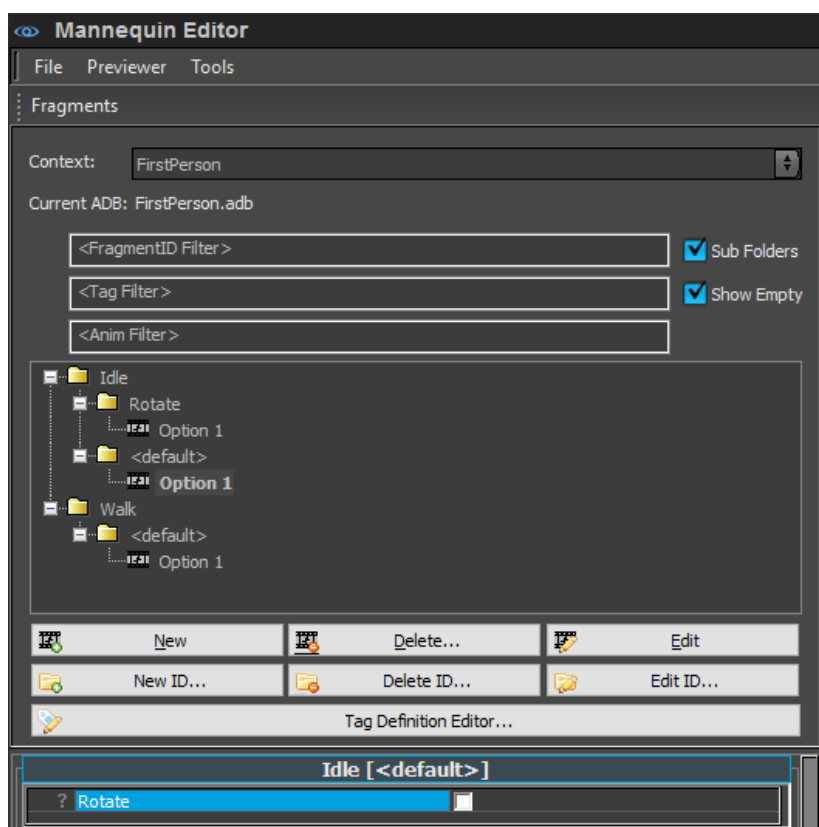
یک متغیر با نام `m_walkFragmentId` تعریف می کند که آدرس قطعه انیمیشن `Walk` را باید به آن اختصاص داد.

```
TagID m_rotateTagId;
```

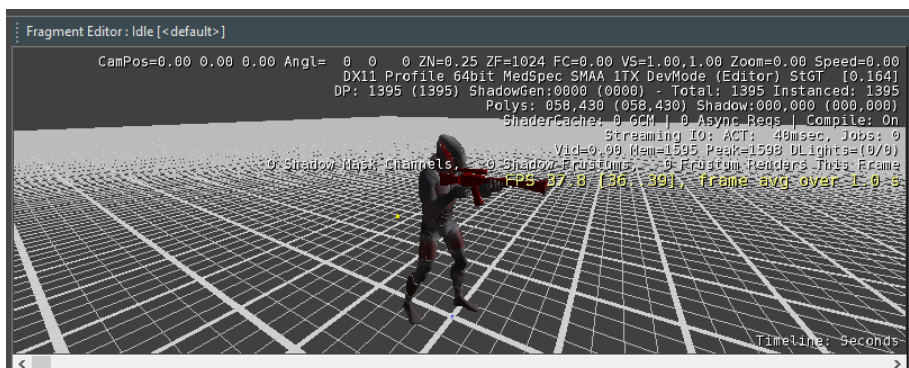
یک متغیر با نام `m_rotateTagId` تعریف می کند که فقط تگ `Id` مربوط به چرخش یا `Rotate` را باید به آن اختصاص داد.

مانکن ادیتور جایی است که شما باید تگ ها و انیمیشن ها را با لایه های مختلف ایجاد کنید، هدف من آشنایی شما با این پنجره است و شرح کامل آن نیست، زیرا که کتابچه راهنمای دیگری را می طلبد و در این کتاب نمی گنجد، به شکل های زیر نگاه کنید :

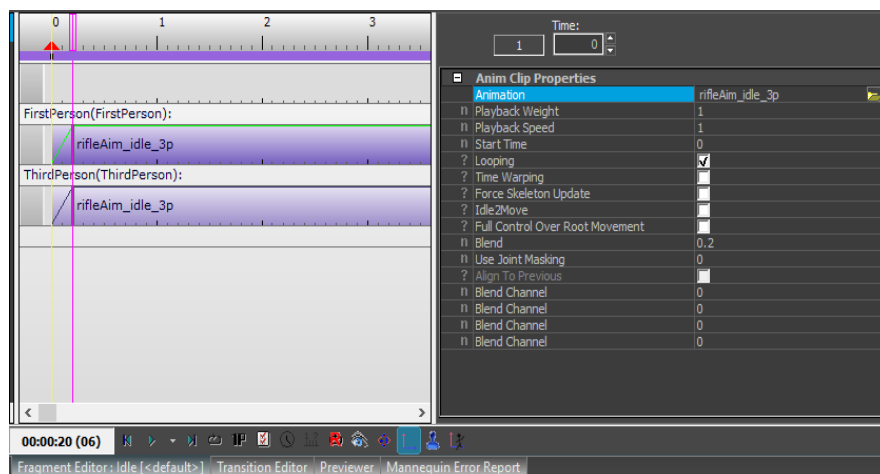
در پنجره زیر تگ هایی که انیمیشن های مختلف پلیر قابلیت پخش را دارند، نمایش می دهد، این تگ ها می توانند با دکمه های مختلف که وجود دارد، اضافه شوند، حذف شوند، ویرایش شوند و.... شما می توانید پوشه بندی های مختلف برای انیمیشن ها را انجام دهید



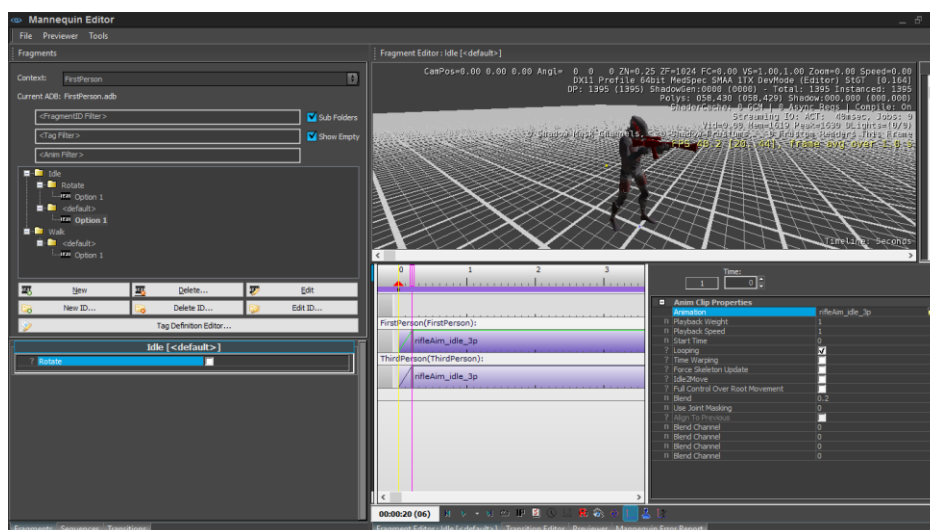
پیش نمایش پلیر با مدل سه بعدی و انیمیشن های قابل پخش در فضای سه بعدی مانکن ادیتور در زیر می بینید، نتایج و عکس العمل های انیمیشنی پلیر در این پنجره نمایش داده می شود



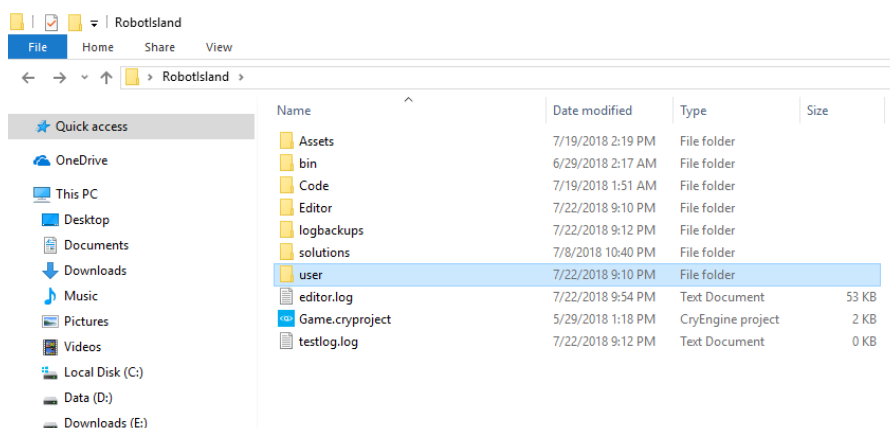
خط زمانی (Time Line) پخش انیمیشن ها در لایه های مختلف را می بینید که می توان ارتباط این لایه ها و تگ ها را با کدهای C++ بالا مقایسه کنید، به توضیحات این پنجره در ادامه این فصل بیشتر می پردازم.



نمای مانکن ادیتور در یک نگاه را در تصویر زیر مشاهده کنید.



مسئله ای مهم که لازم به ذکر است، در صورتی که Layout مانکن ادیتور بهم خورد و Layout آن تغییر یافت می توانید پوشه User در مسیر پروژه مثل آنچه که در شکل می بینید حذف کنید :



تنظیمات Layout به حالت پیش فرض برمیگردد.

`TInputFlags m_inputFlags;`

تعریف پرچم دستگاه های ورودی با نام متغیر `m_inputFlags` در نظر گرفته می شود

```
Vec2 m_mouseDeltaRotation;
```

تعریف برش های چرخش ماوس با نام متغیر `m_mouseDeltaRotation` از نوع بردار دو بعدی در دستگاه دکارتی پیکسلی صفحه نمایش اختصاص می یابد

```
MovingAverage<Vec2, ۱۰>  
m_mouseDeltaSmoothingFilter;
```

میانگین حرکت ماوس با نوع داده ژنریک در متغیر با نام `m_mouseDeltaSmoothingFilter` تعریف می شود تا بتوانید بر روی برش های حرکتی ماوس کنترل داشته باشید

```
const float m_rotationSpeed = ۰,۰۰۲f;
```

سرعت چرخش ماوس با نام متغیر `m_rotationSpeed` به صورت ثابت تعریف می شود

```
int m_cameraJointId = -۱;
```

نقطه اتصال `Id` دوربین در متغیری از نوع صحیح با نام `m_cameraJointId` تعریف شده است.

```
FragmentID m_activeFragmentId;
```

انیمیشن فعلی جاری پلیر که در حال پخش است در این متغیر آدرس دهی شده است.

```
Quat m_lookOrientation;
```

جهت نگاه پلیر در نوع داده Quaternion با نام متغیری m_lookOrientation تعریف شده است.

```
float m_horizontalAngularVelocity;
```

متغیر دیگری با نام m_horizontalAngularVelocity تعریف شده است که شتاب زاویه ای را به صورت افقی برای پلیر محاسبه می کند.

```
MovingAverage<float, ۱۰>
```

```
m_averagedHorizontalAngularVelocity;
```

متغیری ژنریکی با نام

m_averagedHorizontalAngularVelocity تعریف شده است که

میانگین حرکت زاویه ای افقی با شتاب پلیر را در خود ذخیره می کند

```
private:
```

```
bool KillPlayerFire=false;
```

متغیر KillPlayerFire نشان می دهد که آیا پلیر در حال مرگ است، از این متغیر در بازی استفاده نشده است، فقط برای آینده و توسعه کدها رزرو شده است.

```
float sum=۰;
```


متغیر `sum` برش زمانی را در خود ذخیره می کند.

```
float sum۲=۰;
```

متغیر `sum۲` برش زمانی را در خود ذخیره می کند.

```
bool isFire = true;
```

متغیر `isFire` نشان می دهد که آیا اسلحه در حال شلیک است، این متغیر از نوع `Boolean` بوده و تنها می تواند مقدار `true` یا `false` را در خود ذخیره کند، مقدار اولیه این متغیر `true` است.

```
bool bRayHit=false;
```

متغیر `bRayHit` تعریف می کند که آیا اشعه برخوردی (`Raycast`) فعال شده است یا نه؟

```
void SetPositionPlayer(float x, float y ,  
float z);
```

تابع `SetPositionPlayer` با سه پارامتر `x, y, z` تعریف شده است که موقعیت پلیر را در فضای سه بعدی مقدار دهی می کند.

```
float moveSpeed = ۲۰,۵f;
```

متغیر `moveSpeed` سرعت پلیر را تغییر می دهد، در اینجا مقدار `۲۰,۵` به آن اختصاص داده شده است اما در ادامه این فصل می بینید که با زدن دکمه های `shift` مقدار این متغیر عوض می شود و با رها کردن `shift` دوباره به مقدار `۲۰,۵` بر میگردد.

```
float viewOffsetForward = -۰,۱f /* ۰,۸۵f */ ;
```

دوربین پلیر با متغیر `viewOffsetForward` می تواند نسبت به پلیر
تغیر مکان در جهت عمق داشته باشد.

```
float viewOffsetUp = ۰,۲۶f;
```

دوربین پلیر با متغیر `viewOffsetUp` می تواند نسبت به پلیر تغییر
مکان در جهت بالا داشته باشد.

```
bool isAKActive=false;
```

متغیر `isAKActive` نشان می دهد که آیا اسلحه در حال شلیک
است، این متغیر از نوع `Boolean` بوده و تنها می تواند مقدار `true` یا
`false` را در خود ذخیره کند، مقدار اولیه این متغیر `true` است.

```
void AttachOneObjectToPlayer();
```

تابع `AttachOneObjectToPlayer` می تواند یک شی را مانند
چراغ قوه (اینجا یک نور ساده است) به پلیر بچسباند.

```
public :  
string pNameClass = "";
```

متغیری از نوع رشته که با نام `pNameClass` تعریف شده است، نام کلاس
اشیاء را در خود ذخیره می کند، این متغیر در حالت فعلی خالی است و بدون
محتوا است.

```
IPhysicalEntity* pHitEntity;
```

متغیر اشاره گری `pHitEntity` می تواند اشیاء ی که فیزیک دارند
(`Rigidbody` و `Collider`) را آدرس دهی کند و آدرس این نوع اشیا
را در خود حفظ کند (فقط یک شی در هر بار `Raycast` شدن).

```
IEntity* pEntity;
```

متغیر اشاره گری `pEntity` می تواند هر نوع اشیاء ی را که نیاز دارد را آدرس دهی کند.

```
IEntityClass* pClass;
```

متغیر اشاره گری `pClass` می تواند هر نوع کلاسی را که نیاز دارد را آدرس دهی کند.

```
IEntity *pParticleEntity = nullptr;
```

متغیر اشاره گر `pParticleEntity` می تواند هر نوع شی را در خود حفظ کند، در اینجا هدف پارتیکل سیستم است و آدرس دهی یا همان محتوای این متغیر خالی است و به هیچ آدرسی اشاره نمی کند

```
IEntity* m_pPlayer;
```

متغیر `m_pPlayer`، برای آدرس دهی هر `Entity` استفاده می شود اما هدف در اینجا آدرس دهی پلیر است و این متغیر در کدها استفاده نشده است.

```
void GetDirCamera();
```

تابع `GetDirCamera` جهت دوربین پلیر را محاسبه می کند، این تابع هیچ چیزی را بر نمی گرداند چون خروجی آن از نوع `void` است.

```
CPlayerComponent *cpc;
```

یک متغیر از نوع داده کلاسی `CPlayerComponent` با نام `cpc` تعریف شده است، این متغیر یک اشاره گر است و در کدها استفاده نشده است.

```
void RayCast();
```

تابع `RayCast`، محاسبات اشعه های برخورد فیزیکی را انجام می دهد.

```
void DoingKillFireCondition(ray_hit hit);
```

تابع `DoingKillFireCondition` بر اساس پارامتر `hit` بررسی می کند که آیا اشعه برخورد فیزیکی اتفاق افتاده است یا نه؟

```
void DoneKillFireCondition();
```

تابع `DoneKillFireCondition` از آنجایی که اشعه برخوردی را براساس شرط درونی تابع قبلی `DoingKillFireCondition` را بررسی می کند و برخورد اتفاق افتاده است، عملیات بعد از برخورد شی در تابع `DoneKillFireCondition` انجام می شود

```
void SpawnMyPlayer(IEntity* otherEntity);
```

عمل تکثیر پلیر با تابع `SpawnMyPlayer` انجام می شود و طبق پارامتر ورودی `otherEntity` انجام می شود، البته در اینجا تکثیر قرارگیری پلیر در فضای سه بعدی است.

```
};
```

فایل Player.cpp دارای تعداد خطوط بسیار طولانی است و به شرح زیر است:

```
#include "StdAfx.h"
#include "Player.h"
#include "Bullet.h"
#include "Grenade.h"
#include "CfBomb.h"
#include "Particle\1.h"
#include "SpawnPoint.h"
```

اگر به یاد آورید من قبلاً این هدر فایل ها را با مثال های مختلف برای شما بیان کرده ام و از توضیحات مجدد آن پرهیز می کنم.

```
#include <CryParticleSystem\IParticlesPfx۲.h>
#include <CryParticleSystem\IParticles.h>
```

این دو هدر فایل برای تعریف و پیاده سازی پارتيكل سیستم های مختلف به شیوه نسخه کرای انجین ۳ و نسخه کرای انجین ۵ است.

```
#include <CryRenderer/IRenderAuxGeom.h>
```

این هدر فایل ها برای استفاده از سیستم رندرینگ متن و کشیدن اشکال هندسی سه بعدی و دریافت مشخصات صفحه نمایش کاربرد دارد

```
#include <CrySchematyc/Env/IEnvRegistrar.h>
#include
<CrySchematyc/Env/Elements/EnvComponent.h>
```

این دو هدر فایل برای ثبت کامپونت ها و اینتیتی ها در سندباکس و سیماتیک کاربرد دارد.

```
#include <CryPhysics\physinterface.h>
```

این هدر فایل تعریف سیستم فیزیک با تمام روابط آن و اشکال هندسی آن را برعهده می گیرد.

```
#define MOUSE_DELTA_TRESHOLD ۰,۰۰۰۱f
```

میزان حساسیت حرکت ماوس را تعریف می کند.

```
static void
RegisterCPlayerComponent(Schematyc::IEnvRegistrar& registrar)
{
    Schematyc::CEnvRegistrationScope scope =
    registrar.Scope(IEntity::GetEntityScopeGUID())
    ;
    {
        Schematyc::CEnvRegistrationScope
        componentScope =
        scope.Register(SCHEMATYC_MAKE_ENV_COMPONENT(CP
        layerComponent));
        // Functions
        {
        }
    }
}
```

قبلا این تابع را توضیح داده ام، عمل ثبت کامپونت **Player** را در سیماتیک و سندباکس انجام می دهد

```
CRY_STATIC_AUTO_REGISTER_FUNCTION(&RegisterCPlayerComponent)
```

برای نمایش دادن کامپونت ثبت شده باید از این خط کد استفاده کنید.

```
void CPlayerComponent::SpawnMyPlayer(IEntity*  
otherEntity)  
{  
otherEntity->SetWorldTM(m_pEntity->  
GetWorldTM());  
}
```

این تابع را قبلا توضیح داده ام اما عملکرد آن به این صورت است که **entity** ایجاد شده را در فضای سه بعدی بازی قرار می دهد.

```
void CPlayerComponent::Initialize()  
{
```

اولین تابعی که نقش مقدار دهنده را برعهده میگیرد و یکبار اجرا می شود تابع **Initialize** است، البته این تابع با رویداد سیستمی هم شناخته شده است

```
DefineCameraPlayer();
```

این تابع دوربین پلیر را تعریف می کند.

```
DefineCharacterController();
```

این تابع فیزیک بدنه پلیر را تعریف می کند

```
DefineAnimationModel();
```

این تابع مدل های انیمیشن دار پلیر را تعریف می کند.

```
LoadAnimationClips();
```

این تابع عمل بارگذاری انیمیش های پلیر را تعریف می کند.

```
DefineInput();
```

این تابع دستگاه های صفحه کلید و ماوس را برای پلیر تعریف می کند.

```
MoveLeft();
```

این تابع حرکت به سمت چپ را برای پلیر تعریف می کند.

```
MoveRight();
```

این تابع حرکت به سمت راست را برای پلیر تعریف می کند

```
MoveForward();
```

این تابع حرکت به سمت جلو را برای پلیر تعریف می کند.

```
MoveBack();
```

این تابع حرکت به سمت عقب را برای پلیر تعریف می کند.

```
XMouse();
```

این تابع حرکت افقی را برای ماوس پلیر تعریف می کند.

```
YMouse();
```


این تابع حرکت عمودی را برای ماوس پلیر تعریف می کند.

ESCKey();

این تابع موقعیت پلیر را به نقطه آغاز مرحله با زدن دکمه ESC برروی صفحه کلید برمیگرداند

FKey();

با این تابع با زدن دکمه F برروی صفحه کلید، اسلحه F فعال و شلیک می شود (ShotGun)

GKey();

با این تابع با زدن دکمه G برروی صفحه کلید، اسلحه G فعال و شلیک می شود (C4Bomb)

HKey();

با این تابع با زدن دکمه H برروی صفحه کلید، اسلحه H فعال و شلیک می شود (Grenade)

IKey();

با این تابع با زدن دکمه I برروی صفحه کلید، دوربین پلیر از مدل پلیر تغییر فاصله می دهد (نوع اول)

OKey();

با این تابع با زدن دکمه O برروی صفحه کلید، دوربین پلیر از مدل پلیر تغییر فاصله می دهد (نوع دوم)

JKey();

با این تابع با زدن دکمه `J` برروی صفحه کلید، اسلحه `J` فعال و شلیک می شود (Fire)

```
SpaceKey());
```

با این تابع با زدن دکمه `Space` برروی صفحه کلید، عمل پرش پلیر انجام می شود

```
AllShiftKey());
```

این تابع کلیدهای `Shift` برروی صفحه کلید را فعال می کند که سرعت پلیر کاسته شده یا افزایش می یابد

```
Revive());
```

این تابع کلیه مقادیر متغیر و یا توابع موردنظر در ارتباط با پلیر را صفر می کند تا عملیات مربوط به پلیر دوباره آغاز شود

```
SetPlayerOnceInit());
```

این تابع فقط یکبار موقعیت پلیر را به نقطه آغاز بازی بر میگرداند زیرا که رویداد `Initialize` فقط یکبار اتفاق می افتد، توابعی برای صفحه کلید هستند که مقیم حافظه می شوند و باعث می شود که توابع صفحه کلید در داخل رویداد `Initialize` همیشه اجرا شوند

شما به سادگی می توانید از API کرای انجین با نام تابع `FindEntityByName` استفاده کنید که اینتیتی `StartPoint` را در مرحله پیدا کند و پلیر را به آن نقطه انتقال دهد (خیلی ساده ست) مثال زیر را ببینید و داخل این تابع اعمال کنید:

```
IEntity* StartPoint = gEnv->pEntitySystem-
>FindEntityByName("StartPoint");
GetEntity()->SetPos(StartPoint->GetPos());
```

```
HidePlayer(true);
```

این تابع پلیر را مخفی می کند زمانی که مقدار `true` به تابع اختصاص داده شود و اگر مقدار `false` استفاده شود، پلیر نمایش داده شده و مخفی نیست

```
// if(gEnv->IsEditor())
// return;
```

این دو خط اگرچه توضیح (زیرا با رنگ سبز) است اما اگر از حالت توضیح خارج شود یعنی علامت `//` را برداریم، در دستور `if` بررسی می کند که اگر در داخل ادیتور کرای انجین (سندباکس) بودیم، عمل `return` را انجام بده یعنی انتهای تابع باشد و یا دستور یا دستورات بعدی را اجرا نکن

```
// const string w1= "W1";
```

یک متغیر با نام `w1` که از نوع رشته بوده و مقدار `w1` را در خود دارد و محتوا را نمی توان تغییر داد (اگر داخل کد تغییر داده شود خطای کامپایلری رخ می دهد).

```
// IEntity* pChild = gEnv->pEntitySystem-
>FindEntityByName("W1");
```

یک متغیر با نام `pChild` تعریف می کند که داخل مرحله به دنبال یک `entity` با نام `W1` می گردد و اگر پیدا شد به داخل متغیر آدرس دهی می شود، زیرا که متغیر از نوع اشاره گر است، دقت کنید که خطوط سبز رنگ هستند یعنی حالت توضیح هستند و اجرا نمی شوند.

```
// pChild->SetPos(Vec3(0,2,1));
```

شی `W1` یا همان `entity` پیدا شده را در موقعیت فضای سه بعدی در مختصات `z=1` ، `y=2` ، `x=0` قرار می دهد.

```
// pChild->SetScale(Vec3(0,5,0,5,0,5));
```

شی `W1` یا همان `entity` پیدا شده تغییر مقیاس داده می شود و به صورت مربعی تغییر مقیاس با اندازه برداری `(Vec3)` ، `y=0,5` ، `x=0,5` ، `z=0,5` قرار می گیرد.

```
// GetEntity()->AttachChild(pChild);
```

حالا شی `W1` یا `entity` پیدا شده به شی جاری که کد در حال اجرا است (همین `entity`) الصاق و اضافه می شود.

```
//CryLog("GREENLIGHT");
```

یک پیغام با نام `GREENLIGHT` در پنجره خروجی ویژوال استودیو و یا در پنجره کنسول سندباکس کرای انجین نمایش داده می شود

```
}
```

```
void CPlayerComponent::SetPlayerOnceInit()
{
    GetEntity()->SetPos(Vec3(۶۵.f, ۵۵.f, ۳۳.f));
}
```

همانطور که می بیند تابع `SetPlayerOnceInit` عمل زیر را انجام می دهد: شی جاری یا همان `GetEntity` (پلیر) را در موقعیت فضای سه بعدی با مختصات برداری $x=۶۵$ ، $y=۵۵$ ، $z=۳۳$ قرار می دهد

```
void CPlayerComponent::HidePlayer(bool hp)
{
    if(hp)
        GetEntity()->SetViewDistRatio(۰);
    else
        GetEntity()->SetViewDistRatio(۲۵۵);
}
```

تابع `HidePlayer` براساس پارامتر `hp` بررسی می کند که اگر `hp` دارای محتوای `true` بود، شی جاری یا همان `GetEntity` (پلیر) را با استفاده از تابع `SetViewDistRatio` نمایش داده نمی شود و در غیر اینصورت اگر `hp` دارای محتوای `false` باشد، شی جاری یا همان `GetEntity` (پلیر) با استفاده از تابع `SetViewDistRatio` نمایش داده خواهد شد

حتما دقت کنید، از آنجایی این کدها در فایل `player.cpp` اجرا می شوند، منظور از `GetEntity` همان پلیر است

```
}
```

```
void CPlayerComponent::JKey()
{
    m_pInputComponent->RegisterAction("player",
    "J", [this](int activationMode, float value)
    {
```

داخل تابع JKey متغیر فعال ساز یعنی m_pInputComponent برای دستگاه های ورودی صفحه کلید و ماوس باید یک کلید یا کلیک را با استفاده از رویداد سیستمی با نام RegisterAction برای دکمه J بر روی صفحه کلید ثبت نماید

```
if (activationMode == eIS_Pressed)
{
```

پارامتر activationMode تعیین می کند که آیا دکمه J فشرده شده است، که اگر فشرده شده باشد، بلوک if اجرا می شود

```
isAKActive=true;
```

متغیر isAKActive مقدار true را میگیرد

```
}
```

```
if (activationMode == eIS_Released)
{
```

پارامتر activationMode تعیین می کند که آیا دکمه J رها شده است، که اگر رها شده باشد، بلوک if اجرا می شود

```
isAKActive = false;
```

متغیر isAKActive مقدار false را میگیرد

```
}
```

```
});
```

```
m_pInputComponent->BindAction("player", "J",  
eAID_KeyboardMouse, EKeyId::eKI_J);
```

با توجه به رویداد سیستمی BindAction، عمل مقیم سازی دکمه J
بر روی صفحه کلید در حافظه با استفاده از متغیر
m_pInputComponent انجام می شود

```
}
```

نکته بسیار مهم در رابطه با enum از EInputState تعریف شده
در هدر فایل IInput.h، دکمه های صفحه کلید و ماوس این است
که نباید فراموش کنید ۲ activationMode یا
activationMode می تواند حالت های زیر را داشته باشد :

eIS_Pressed : وقتی بر روی دکمه، عمل فشردن شدن انجام شود
و یکبار این عمل انجام می شود

eIS_Released : وقتی که عمل رها کردن دکمه، انجام می شود و
یکبار این عمل انجام می شود

eIS_Down : تا زمانی که دکمه فشرده شده است و آن را رها نکرده ایم، این عمل انجام می شود، این عمل می تواند چندین و چندبار انجام شود، در این کتاب به این رویداد پرداخته نشده است و شما می توانید از دستور **if** و **CryLog** برای مشاهده نتایج استفاده کنید

```
void CPlayerComponent::IKey()
{
    m_pInputComponent->RegisterAction("player",
    "I", [this](int activationMode, float value)
    {
```

داخل تابع **IKey** متغیر فعال ساز یعنی **m_pInputComponent** برای دستگاه های ورودی صفحه کلید و ماوس باید یک کلید یا کلیک را با استفاده از رویداد سیستمی با نام **RegisterAction** برای دکمه **I** برروی صفحه کلید ثبت نماید

```
if (activationMode == eIS_Pressed)
{
    پارامتر activationMode تعیین می کند که آیا دکمه I فشرده شده است، که اگر فشرده شده باشد، بلوک if اجرا می شود
```

```
viewOffsetForward=۰,۸۵f;
```

دوربین پلیر نسبت به مدل سه بعدی پلیر به اندازه ۰,۸۵ متر به جلو اختصاص میابد

```
viewOffsetUp=۰,۲۶f;
```


دوربین پلیر نسبت به مدل سه بعدی پلیر به اندازه ۰,۲۶ متر به بالا اختصاص میابد.

```
CryLog("IIIIII...");
```

یک پیغام در ویژوال استودیو یا در کنسول سندباکس کرای انجین با محتوای
IIIIIIII... نمایش داده می شود

```
}
```

```
});
```

```
m_pInputComponent->BindAction("player", "I",  
eAID_KeyboardMouse, EKeyId::eKI_I);
```

با توجه به رویداد سیستمی BindAction، عمل مقیم سازی دکمه I
بر روی صفحه کلید در حافظه با استفاده از متغیر
m_pInputComponent انجام می شود

```
}
```

```
void CPlayerComponent::OKey()
```

```
{
```

```
m_pInputComponent->RegisterAction("player",  
"O", [this](int activationMode, float value)
```

```
{
```

```
if (activationMode == eIS_Pressed)
```

```
{
```

اگر دکمه 0 بر روی صفحه کلید فشرده شود، دستورات زیر اجرا می شوند.

```
viewOffsetForward = -۱.f;
```

دوربین پلیر نسبت به مدل سه بعدی پلیر به اندازه ۱ متر به عقب اختصاص میابد (نسبت به فشرده شدن دکمه I عقب می رود)

```
viewOffsetUp = ۱.۶f;
```

دوربین پلیر نسبت به مدل سه بعدی پلیر به اندازه ۱,۶ متر به بالا اختصاص میابد (نسبت به فشرده شدن دکمه I بالاتر می رود)

```
}  
});
```

```
m_pInputComponent->BindAction("player", "0",  
eAID_KeyboardMouse, EKeyId::eKI_0);
```

```
}
```

با توجه به رویداد سیستمی BindAction، عمل مقیم سازی دکمه 0 بر روی صفحه کلید در حافظه با استفاده از متغیر m_pInputComponent انجام می شود.

```
void CPlayerComponent::HKey()  
{  
m_pInputComponent->RegisterAction("player",  
"H", [this](int activationMode, float value)  
{  
if (activationMode == eIS_Pressed)
```

{
اگر دکمه H بر روی صفحه کلید فشرده شود بلوک های if زیر اجرا می شوند:

متغیر HKeyAmmo در هدر فایل StdAfx.h تعریف شده است و در اینجا بررسی میشود که اگر HKeyAmmo کمتر یا مساوی صفر باشد دو دستور را انجام دهد، یک صدا با نام تریگری Null_AmmoHKey پخش شود و سپس متغیر HKeyAmmo برابر صفر شود، با اجرا این بلوک if یعنی صدای ماشه اسلحه که خالی است پخش شود، اینجا اسلحه ما نارنجک است، می توانید صدای تریگری را در پنجره Audio Controls Editor قرار دهید که به پخش نماید نارنجک تمام شده است یا صدای غیر پخش شود... خط مربوط به پخش صدا به حالت توضیح درآورده ام و شما باید تریگر صوتی مربوط به نام Null_AmmoHKey را ایجاد کنید

```
if (HKeyAmmo <= ۰)
{
//PublicPlayMySound("Null_AmmoHKey");
HKeyAmmo = ۰;
}

// PublicPlayMySound("ThrowGrenade");
```

این دستور صدای پرت کردن نارنجک را با نام تریگری صوتی ThrowGrenade پخش می کند، شما باید تریگر مربوط به این صدا را در Audio Controls Editor ایجاد کنید و این خط را به حالت توضیح درآورده ام.

در بلوک if زیر بررسی می شود که اگر متغیر HKeyAmmo بزرگتر یا مساوی یک شود، دستورات زیر اجرا شود :

```
if(HKeyAmmo>=۱)
```

```
{
```

```
HKeyAmmo--;
```

یک واحد از محتوای عددی صحیح متغیر HKeyAmmo کاسته می شود

```
SEntitySpawnParams spawnParams;
```

یک متغیر با نام spawnParams ایجاد می شود که با این متغیر عملیات تکثیر نارنجک را آدرس دهی می کنم

```
spawnParams.pClass = gEnv->pEntitySystem-  
>GetClassRegistry()->GetDefaultClass();
```

با توجه به وجود کلاس نارنجک در چند خط کد دیگر، نوع کلاس پیش فرضی را تعیین می کنم که باید عمل تکثیر انجام شود

```
spawnParams.vPosition =GetEntity()->GetPos()+  
Vec3(۰،۰،۳);
```

موقعیت کلاس پیش فرض ایجاد شده در فضای سه بعدی توسط دو عملوند تابعی انجام می شود:

۱- عملوند تابعی GetEntity()->GetPos() به موقعیت پلیر در مختصات فعلی در مرحله اشاره می کند.

۲- عملوند تابعی Vec3(۰،۰،۳) به بردار سه بعدی اشاره دارد که سه متر ارتفاع دارد.

نتیجه موقعیت فعلی پلیر با سه متر ارتفاع محور Z جمع می شود و مختصات حاصل می شود که کلاس پیش فرض به آن مختصات نیاز دارد.

```
spawnParams.qRotation = Quat(Vec3(۰، ۰، ۰));
```

عمل چرخش کلاس در حال تولید با توجه به تابع چرخش ساز با بردار Zero یا همان صفر مقدار دهی می شود، یعنی در هنگام پرت کردن کلاس در حال تولید، entity تولید شده هیچ زاویه ای نداشته باشد و زاویای آن صفر باشد.

```
const float GrenadeScale = ۱,۰f;
```

ثابت محتوایی (متغیر بدون عوض شدن مقدار) با نام GrenadeScale از نوع داده اعشاری برابر یک تعریف و مقدار دهی می شود.

```
spawnParams.vScale = Vec3(GrenadeScale);
```

کلاس در حال تولید باید اندازه ای داشته باشد، این اندازه با بردار مربعی (Vec3(GrenadeScale) در محورهای x, y, z مقداردهی می شود یعنی $x=1$ ، $y=1$ و $z=1$ است.

```
IEntity* pEntity = gEnv->pEntitySystem->SpawnEntity(spawnParams);
```

یک متغیر با نام pEntity تعریف کرده ام که باید کلاس را به شی (Entity) تبدیل نماید و آدرس شی در حال تولید در این متغیر قرار گیرد.

```
pEntity-
>CreateComponentClass<CGrenadeComponent>();
```

با توجه به تعریف متغیر `pEntity` در بالا حالا باید کلاس `CGrenadeComponent` را ایجاد کنیم، این کلاس را در صفحات قبلی در هدر فایل و فایل های `Grenade.cpp` ، `Grenade.h` بررسی کردیم و با این خط نرنجک ایجاد می شود

```
}

} // if spwan

});
```

```
m_pInputComponent->BindAction("player", "H",
eAID_KeyboardMouse, EKeyId::eKI_H);
```

با توجه به رویداد سیستمی `BindAction`، عمل مقیم سازی دکمه `H` بر روی صفحه کلید در حافظه با استفاده از متغیر `m_pInputComponent` انجام می شود

```
}
```

```
void CPlayerComponent::SetPositionPlayer(float
x, float y, float z)
{
```

در صورتی که متغیر `KillPlayerFire` مقدار `true` داشته باشد، پلیر که آتش گرفته است (در حال سوختن است)، آتش خاموش می

شود، اگر متغیر KillPlayerFire مقدار true داشته باشد، دستورات زیر اجرا می شود (در هدر فایل StdAfx.h شما می توانید یک متغیر استاتیک با نام HealthPlayer تعریف کنید و در زمان سوختن و آتش گرفتن پلیر از مقدار متغیر HealthPlayer بکاهید)

```
if(KillPlayerFire)
{
    GetEntity()->FreeSlot(.);
```

شکاف صفر یا slot۰ را برای پلیر آزاد کنید تا عمل لود پارتیکل سیستم انجام شود

```
IParticleEffect* pEffectPlayer=gEnv-
>pParticleManager->FindEffect("blank.pfx");
```

پارتیکل سیستم خالی با نام blank.pfx جستجو می شود و در متغیر اشاره گر pEffectPlayer آدرس دهی می شود

```
GetEntity()->LoadParticleEmitter(. ,
pEffectPlayer);
```

عمل لود پارتیکل سیستم خالی در slot۰ در پلیر انجام می شود و عملاً آتش خاموش می شود

```
KillPlayerFire=false;
```

متغیر KillPlayerFire مقدار false می گیرد، تا بلوک if مجدداً اجرا نشود

```
}
```

```
GetEntity()->SetPos(Vec3(x, y, z));
```

طبق پارمترهای برداری $\text{Vec3}(x, y, z)$ پلیر به موقعیت آرگومانی در ناحیه فراخوانی تابع `SetPositionPlayer` باز می گردد

```
}
```

```
void CPlayerComponent::DefineCameraPlayer()
{
```

```
m_pCameraComponent = m_pEntity-
>GetOrCreateComponent<Cry::DefaultComponents::
CCameraComponent>();
```

تابع `DefineCameraPlayer`، یک متغیر برای پلیر با نام `m_pCameraComponent` تعریف و یک دوربین را ایجاد می کند و به صورت اتوماتیک هرفریم عمل پردازش و به روزرسانی برای دوربین انجام می شود

```
}
```

```
void
CPlayerComponent::DefineCharacterController()
{
```

```
m_pCharacterController = m_pEntity-
>GetOrCreateComponent<Cry::DefaultComponents::
CCharacterControllerComponent>();
```


تابع `DefineCharacterController` یک متغیر با نام `m_pCharacterController` را تعریف می‌شود و یک کاراکتر کنترلر ایجاد می‌شود

```
m_pCharacterController-
>SetTransformMatrix(Matrix3f::Create(Vec3(۱.f),
IDENTITY, Vec3(۰, ۰, ۱.f)));
```

سپس یک ماتریکس یکانی واحد برای ارتفاع کاراکتر کنترلر به اندازه ۱ متر اختصاص می‌دهد

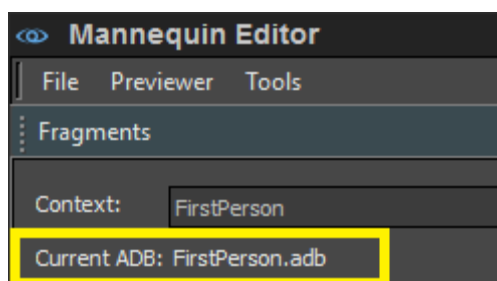
```
}

void CPlayerComponent::DefineAnimationModel()
{
```

```
m_pAnimationComponent = m_pEntity-
>GetOrCreateComponent<Cry::DefaultComponents::
CAdvancedAnimationComponent>();
```

تابع `DefineAnimationModel` یک متغیر با نام `m_pAnimationComponent` را تعریف و یک سیستم انیمیشن پیشرفته را برای پلیر ایجاد می‌کند، این سیستم انیمیشن پیشرفته با مانکن ادیتور در ارتباط است

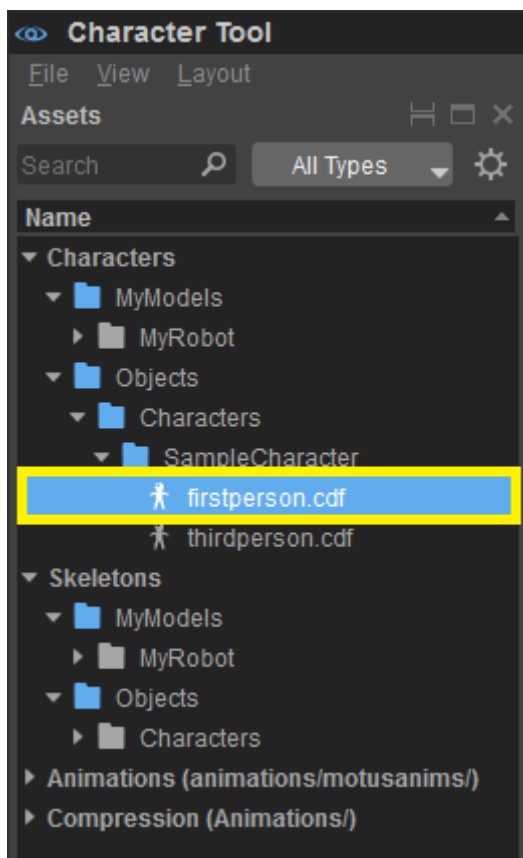
```
m_pAnimationComponent-
>SetMannequinAnimationDatabaseFile("Animations
/Mannequin/ADB/FirstPerson.adb");
```



طبق مسیر پوشه های تودرتو و ختم شده به فایل `FirstPerson.adb`، انیمیشن های مختلف برای سیستم پیشرفته در متغیر تعریف می شود

```
m_pAnimationComponent-
>SetCharacterFile("Objects/Characters/SampleCharacter/firstperson.cdf");
```

با توجه به مسیر تودرتو و ختم شده به فایل `firstperson.cdf`، مدل سه بعدی ریگ شده به متغیر اختصاص داده می شود و در پنجره `Character Tool` فایل `firstperson.cdf` و `thirdperson.cdf` وجود دارند، به شکل زیر نگاه کنید.

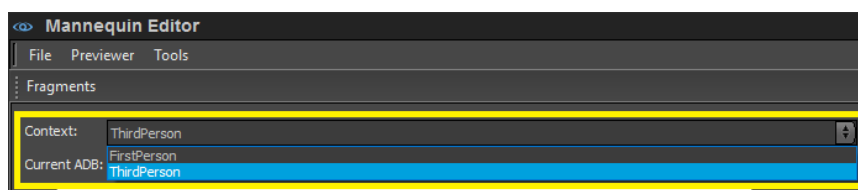


```
m_pAnimationComponent-
>SetControllerDefinitionFile("Animations/Mannequin/ADB/FirstPersonControllerDefinition.xml");
```

کلیه لایه های انیمیشنی و کلیپ های انیمیشنی Walk ، Idle در مانکن ادیتور FirstPersonControllerDefinition.xml در یک فایل با نام قرار داده شده است و این فایل توسط مانکن ادیتور ایجاد می شود.

```
m_pAnimationComponent-
>SetDefaultScopeContextName("FirstPersonCharacter");
```

طبق تصویر زیر نوع `context` مدل که از نوع سه بعدی اول شخص یا سوم شخص خواهد بود با این خط کد در بالا نوع `context` باید از `FirstPersonCharacter` تبعیت کند.



```
m_pAnimationComponent-
>SetDefaultFragmentName("Idle");
```

با توجه به صف کلیپ های انیمیشنی باید یک انیمیشن را به حالت پیش فرض انتخاب کنیم که پلیر آن را اجرا نماید، با توجه به این خط انیمیشن `Idle` نمایش داده می شود و به صورت پیش فرض و همیشگی این انیمیشن اجرا خواهد شد، مگر اینکه کلیدهای دیگری برای پخش انیمیشن های دیگر تعریف شود، مثل انیمیشن راه رفتن که در خطوط بعدی به این مسئله می پردازم.

```
m_pAnimationComponent-
>SetAnimationDrivenMotion(false);
```

این خط کنترل می کند که توسط فیزیک حرکت انجام شود، در اینجا افست مربوط به نقاط اتصال ها غیرفعال شده است چون مقدار `false` برای تابع `SetAnimationDrivenMotion` فرستاده شده است.

```
m_pAnimationComponent->LoadFromDisk();
```

عمل بارگذاری یا لود کردن داده های کاراکتر و مانکن ادیتور برای پلیر با این خط انجام می شود

```
}
```

```
void
```

```
CPlayerComponent::LoadAnimationClips()
```

```
{
```

تابع LoadAnimationClips عمل بارگذاری یا لود کلیپ های انیمیشنی پلیر را برعهده دارد.

```
m_idleFragmentId = m_pAnimationComponent->GetFragmentId("Idle");
```

این خط با توجه به متغیر m_idleFragmentId آدرس دسترسی به انیمیشن Idle را برای پلیر مهیا می کند.

```
m_walkFragmentId = m_pAnimationComponent->GetFragmentId("Walk");
```

این خط با توجه به متغیر m_walkFragmentId آدرس دسترسی به انیمیشن Walk را برای پلیر مهیا می کند.

```
m_rotateTagId = m_pAnimationComponent->GetTagId("Rotate");
```

این خط با توجه به متغیر m_rotateTagId آدرس دسترسی به تگ چرخش با نام Rotate را برای پلیر مهیا می کند.

```
}
```

```
void CPlayerComponent::DefineInput()
{
```

تابع `DefineInput` یک متغیر با نام `m_pInputComponent` تعریف می کند و یک سیستم ورودی برای دستگاه های ورودی مانند صفحه کلید و ماوس ایجاد می کند

```
m_pInputComponent = m_pEntity-
>GetOrCreateComponent<Cry::DefaultComponents::
CInputComponent>();
```

```
}
```

```
void CPlayerComponent::MoveLeft()
{
```

```
m_pInputComponent->RegisterAction("player",
"moveleft", [this](int activationMode, float
value) {
HandleInputFlagChange((TInputFlags)EInputFlag:
:MoveLeft, activationMode); });
```

```
m_pInputComponent->BindAction("player",
"moveleft", eAID_KeyboardMouse,
EKeyId::eKI_A);
```

در داخل تابع `MoveLeft` عمل حرکت پلیر به سمت چپ با ثابت `enum` کلید `A` با نام `EKeyId::eKI_A` تعریف می شود و تابع مقیم شده در حافظه با نام `BindAction` با توجه به کنترل و وجود ثابت `enum`

eAID_KeyboardMouse انجام می شود، قبلا در رابطه با این توابع در صفحات قبلی توضیحات مبینی ارائه کرده ام

```
}
```

```
void CPlayerComponent::MoveRight()
{
    m_pInputComponent->RegisterAction("player",
    "moveright", [this](int activationMode, float
    value) {
        HandleInputFlagChange((TInputFlags)EInputFlag:
        :MoveRight, activationMode); });
    m_pInputComponent->BindAction("player",
    "moveright", eAID_KeyboardMouse,
    EKeyId::eKI_D);
}
```

در داخل تابع MoveRight عمل حرکت پلیر به سمت راست با ثابت enum کلید D با نام EKeyId::eKI_D تعریف می شود و تابع مقیم شده در حافظه با نام BindAction با توجه به کنترل و وجود ثابت eAID_KeyboardMouse enum انجام می شود، قبلا در رابطه با این توابع در صفحات قبلی توضیحات مبینی ارائه کرده ام

```
}
```

```
void CPlayerComponent::MoveForward()
{

    m_pInputComponent->RegisterAction("player",
    "moveforward", [this](int activationMode,
    float value) {
        HandleInputFlagChange((TInputFlags)EInputFlag:
        :MoveForward, activationMode); });
}
```

```
m_pInputComponent->BindAction("player",  
"moveforward", eAID_KeyboardMouse,  
EKeyId::eKI_W);
```

در رابطه با این خطوط کد توضیحات کافی ارائه شده است، تنها اشاره می کنم که تابع MoveForward عمل حرکت به جلو با کلید W را برای پلیر مهیا می کند

```
}
```

```
void CPlayerComponent::MoveBack()  
{
```

```
m_pInputComponent->RegisterAction("player",  
"moveback", [this](int activationMode, float  
value) {  
HandleInputFlagChange((TInputFlags)EInputFlag:  
:MoveBack, activationMode); });  
m_pInputComponent->BindAction("player",  
"moveback", eAID_KeyboardMouse,  
EKeyId::eKI_S);
```

در رابطه با این خطوط کد توضیحات کافی ارائه شده است، تنها اشاره می کنم که تابع MoveBack عمل حرکت به عقب با کلید S را برای پلیر مهیا می کند.

```
}
```

```
void CPlayerComponent::XMouse()  
{
```



```

m_pInputComponent->RegisterAction("player",
    "mouse_rotateyaw", [this](int activationMode,
    float value) { m_mouseDeltaRotation.x -=
    value; });
m_pInputComponent->BindAction("player",
    "mouse_rotateyaw", eAID_KeyboardMouse,
    EKeyId::eKI_MouseX);

```

تابع XMouse عمل چرخش برای پلیر و دوربین پلیر را طبق محور mouse_rotateyaw در جهت افقی انجام می دهد و حرکت افقی در ثابت enum eKI_MouseX کاملاً مشخص است، اگر دقت کنید متغیر m_mouseDeltaRotation طبق محور X چرخش را انجام می دهد، دقت کنید که کاراکتر کنترلر و دوربین هم رتبه هستند یعنی در یک entity ریشه با نام Player هستند

```

}

```

```

void CPlayerComponent::YMouse()
{
m_pInputComponent->RegisterAction("player",
    "mouse_rotatepitch", [this](int
    activationMode, float value) {
m_mouseDeltaRotation.y -= value; });
m_pInputComponent->BindAction("player",
    "mouse_rotatepitch", eAID_KeyboardMouse,
    EKeyId::eKI_MouseY);
}

```

تابع YMouse عمل چرخش برای پلیر و دوربین پلیر را طبق محور mouse_rotatepitch در جهت عمودی انجام می دهد و حرکت عمودی در ثابت enum eKI_MouseY کاملاً مشخص است، اگر دقت کنید متغیر m_mouseDeltaRotation طبق محور Y چرخش را

انجام می دهد، دقت کنید که کاراکتر کنترلر و دوربین هم رتبه هستند
یعنی در یک entity ریشه با نام Player هستند

```
}
```

```
void CPlayerComponent::SpaceKey()  
{  
m_pInputComponent->RegisterAction("player",  
"Space", [this](int activationMode, float  
value) {
```

در تابع SpaceKey، زمانی که کلید Spacebar صفحه کلید فشرده
شود، بلوک if زیر اجرا می شود

```
if (activationMode == eIS_Pressed)  
{  
بلوک if زیر بررسی می کند که اگر Player بر روی زمین بود، کد بعدی  
را اجرا کند
```

```
if(m_pCharacterController->IsOnGround())
```

با توجه به اینکه الان پلیر بر روی زمین است، با استفاده از تابع
AddVelocity شتابی بر حسب متر بر مجذور ثانیه با واحد m/s^2 بر
اساس محور ارتفاع یعنی محور Z ایجاد می شود، بردار سه بعدی تشکیل می
شود و شتاب با ۱۰ واحد ارسال می شود و پلیر پرش می کند.

```
m_pCharacterController-  
>AddVelocity(Vec(۰,۰,۱۰));  
  
}  
});
```

```
m_pInputComponent->BindAction("player",
                                "Space", eAID_KeyboardMouse,
                                EKeyId::eKI_Space);
```

تابع BindAction قبلا توضیح داده شده است.

```
}

void CPlayerComponent::ESCKey()
{
    m_pInputComponent->RegisterAction("player",
    "ESC", [this](int activationMode, float
    value) {
        if (activationMode == eIS_Pressed)
        {
            SetPositionPlayer(۶۵.f, ۵۵.f, ۳۳.f);
        }
    });

    m_pInputComponent->BindAction("player", "ESC",
    eAID_KeyboardMouse, EKeyId::eKI_Escape);
```

تابع ESCKey بررسی می کند که اگر دکمه Escape بر روی صفحه کلید فشرده شد، پلیر با استفاده از تابع SetPositionPlayer به موقعیت فضای سه بعدی آرگومان های داده شده انتقال یابد، در واقع پلیر به position یا موقعیت داده شده منتقل می شود و تابع BindAction قبلا توضیح داده شده است

```
}
```

```
void CPlayerComponent::Gkey()
{
    m_pInputComponent->RegisterAction("player",
    "G", [this](int activationMode, float value)
    {
        if (activationMode == eIS_Pressed)
        {
            وقتی که کلید G بر روی صفحه کلید فشرده می شود، اعمال زیر اتفاق می افتند :
            if (GKeyAmmo <= ۰)
            {
                //PublicPlayMySound("Null_AmmoGKey");
                GkeyAmmo = ۰;
            }
        }
    });
}
```

با توجه به متغیر تعریف شده در هدر فایل StdAfx.h برای بمب C۴، بلوک `if` بررسی می کند که اگر متغیر `GKeyAmmo` کمتر یا مساوی صفر شد، یعنی خشاب خالی است و صدای تریگری خشاب خالی یا هر صدای غیر دیگری با نام `Null_AmmoGKey` تولید شود و متغیر `GKeyAmmo` صفر گردد، صدای تریگری به حالت توضیح درآمده است و باید این تریگر را در پنجره `Audio Controls Editor` ایجاد کنید

```
// PublicPlayMySound("InsertC۴Bomb");
```

صدای تریگری با نام `InsertC۴Bomb` به حالت توضیح درآمده است و باید این تریگر را در پنجره `Audio Controls Editor` ایجاد کنید، این خط با فعال شدن صدایی را تولید می کند که C۴ در یکجا تعبیه و جاسازی می شود.

در بلوک `if` بعدی بررسی می شود که اگر `GKeyAmmo` بزرگتر یا مساوی یک بود، یعنی خشاب `C۴` خالی نیست و اعمال زیر انجام شود :

```
if (GKeyAmmo >= ۱)
{
Vec۳ pos = GetEntity()->GetPos() + Vec۳(۰, ۰,
۲);
```

یک متغیر محلی برداری با نام `pos` ایجاد می شود که تنها این متغیر داخل این بلوک `if` شناخته شده است و خارج از بلوک `if` این متغیر از بین می رود، اما داخل متغیر `pos` چیست؟

- ۱- موقعیت پلیر در فضای سه بعدی را محاسبه می کند
- ۲- یک بردار سه بعدی که ارتفاع آن براساس محور `Z` ۲ متر است
- ۳- عملوند های موقعیت پلیر و بردار سه بعدی باهم جمع می شوند و یک موقعیت را در متغیر `pos` تولید می شود

```
int ObjType = ent_all;
```

متغیر `ObjType` محلی است و همه اشیاء در بازی را با اشعه برخوردی آدرس دهی می کند، این اشیاء می توانند شامل

<code>ent_static</code>		
<code>ent_sleeping_rigid</code>		<code>ent_rigid</code>
<code>ent_living</code>		<code>ent_independent</code>
		<code>ent_terrain</code> , باشد

```
const unsigned int rayFlags =
rwi_stop_at_pierceable | rwi_colltype_any;
```

متغیر `rayFlags` محلی و ثابت است و پرچم های اشعه برخوردی را فعال می کند

```
gEnv->pPhysicalWorld-
>RayWorldIntersection(pos, ahmadval, ObjType,
rayFlags, &hit, ۱);
```

تابع `RayWorldIntersection` همان تابع اشعه برخوردی است که به آن `Raycast` نیز گفته می شود و با توجه به آرگومان های زیر، نقاط برخورد و نوع شی یا همان `entity` را برای ما در متغیر `hit` می ریزد :

`pos` : با توجه به متغیر تعریف شده در بالا، موقعیت پلیر با ارتفاع ۲ متر، شروع نقطه برخوردی است

`ahmadval` : جهت رو به جلو یا همان عمقی است که دوربین پلیر به آن نگاه می کند و انتهای نقطه برخوردی است

`ObjectType` : با توجه به تعریف متغیر بالا، نوع شناسایی `entity` ها در اثر اشعه برخوردی حاصل می شود

`rayFlags` : با توجه به تعریف متغیر بالا، نوع پرچم به کار رفته را در اشعه برخوردی محاسبه می کند

`hit` : مجموعه اطلاعاتی که هنگام اشعه برخوردی حاصل می شود و در داخل متغیر `hit` ذخیره می شود، مانند مسافت پلیر از نقطه برخوردی، نوع `entity` برخوردی چیست، نقطه برخوردی در فضای سه بعدی، بردار نرمال سه بعدی حاصل شده از نقطه برخوردی و...

۱ : عدد یک همیشه ثابت است و نباید تغییر داده شود، حاصل عدد یک حاصل متراژ نقطه برخوردی تا آخرین نقطه از دور دست ها را شامل می شود.

```
if (hit.dist < ۲,۵f && hit.dist >= ۰,۳f)
{
```

این خط از دستور `if` ، بررسی می کند که ،اگر فاصله پلیر از نقطه برخوردی در بازه ی بزرگتر یا مساوی ۳۰ سانتی متر تا کوچکتر از ۲ متر و ۵۰ سانتی متر بود،عملیات زیر انجام شود.

`GKeyAmmo--;`

یک واحد از خشاب بمب C۴ کم شود.

من قصد دارم که وقتی بمب C۴ را در یک مکان از فضای سه بعدی جاسازی و قرار دهم،برای گیمر متنی نمایش داده شود که در آن مکان در فضای سه بعدی بمب C۴ جاسازی شده است و یک متن با عنوان " `Bomb C۴` here" نمایش داده شود

```
IPersistentDebug *iPD = gEnv->pGameFramework-
>GetIPersistentDebug();
```

ابتدا باید یک یک متغیر اشاره گر را با نام `iPD` تعریف کنم و فضایی از حافظه RAM را برای آن اختصاص دهم و این فضا نهایتاً در صفحه نمایش بازی نمایش داده شود

```
iPD->Begin("EnshaAllah", false);
```

در خط بعدی با یک رشته که می تواند هر مقداری داشته باشد و با آرگومان `false` اختصاص داده شده است،تا عمل شروع نمایش متن در فضای سه بعدی آغاز شود.

```
iPD->AddTextRD(hit.pt, ۲,۱۵f, ColorF(۰, ۱, ۱),
۵.f, "C۴ Boom here");
```



حالا با استفاده از تابع `AddText3D` باید متن را طبق توضیحات زیر در فضای سه بعدی و در مکانی که بمب `C4` جاسازی خواهد شد، این متن قرار گیرید:

`hit.pt`: مکانی که اشعه برخوردی به آن تابش داده شده است و حاصل آن نقطه ای از فضای سه بعدی است که باید متن در آن نمایش داده شود، بمب `C4` نیز در همان مکان قرار خواهد گرفت، اما این نقطه فعلا مربوط به فقط عنوان متن نمایش داده شده خواهد بود

`۲,۱۵f`: اندازه متن قابل نمایش در فضای سه بعدی چقدر باید باشد، اندازه متن یا همان اندازه فونت `۲,۱۵f` خواهد بود

`ColorF`: این تابع تعیین می کند که رنگ متن چه باشد؟ این تابع دارای سه آرگومان از سمت چپ به راست قرمز `r=۰` و `g=۱` سبز و آبی `b=۱` است و متن با توجه به آرگومان های عددی داده شده به صورت آبی کم رنگ نمایش داده خواهد شد

`۵.f`: این پارامتر تعیین می کند که چندثانیه متن در فضای سه بعدی نمایش داده شود و سپس بعد از ۵ ثانیه متن از بین می رود

`C4 Bomb here`: این آرگومان، همان عنوان متنی است که برای درج و جاسازی بمب `C4` باید در فضای سه بعدی نمایش داده شود.

که در بالا تعریف کردیم به توابع زیر نیز دسترسی دارد و iPD دقت کنید که متغیر شما می توانید شکل های هندسی، خطوط و ... را به بازی اضافه کنید، این توابع شامل تعاریف زیر می باشد :

```
virtual void AddSphere(const Vec3& pos, float  
radius, ColorF clr, float timeout) = .;
```

```
virtual void AddDirection(const Vec3& pos,  
float radius, const Vec3& dir, ColorF clr,  
float timeout) = .;
```

```
virtual void AddLine(const Vec3& pos1, const  
Vec3& pos2, ColorF clr, float timeout) = .;
```

```
virtual void AddPlanarDisc(const Vec3& pos,  
float innerRadius, float outerRadius, ColorF  
clr, float timeout) = .;
```

```
virtual void AddCone(const Vec3& pos, const  
Vec3& dir, float baseRadius, float height,  
ColorF clr, float timeout) = .;
```

```
virtual void AddCylinder(const Vec3& pos,  
const Vec3& dir, float radius, float height,  
ColorF clr, float timeout) = .;
```

```

virtual void AddrDText(const char* text, float
size, ColorF clr, float timeout) = ۰;

virtual void AddText(float x, float y, float
size, ColorF clr, float timeout, const char*
fmt, ...) = ۰;

virtual void AddTextrD(const Vecr& pos, float
size, ColorF clr, float timeout, const char*
fmt, ...) = ۰;

virtual void AddrDLine(float x۱, float y۱,
float x۲, float y۲, ColorF clr, float timeout)
= ۰;

//  iPD->AddSphere(hit.pt, ۰.۳f, ColorF(۱, ۱,
۰), ۱۲,۱f);

```

من مثال دیگری را از متغیر تعریف شده برای دسترسی به تابع `AddSphere` که به حالت توضیح (سبزنگ) در آمده است، بیان میکنم، این تابع یک کره سه بعدی را در فضای سه بعدی در بازی رسم می کند:

`hit.pt`: مکانی که اشعه برخوردی به آن تابش داده شده است و حاصل آن نقطه ای از فضای سه بعدی است که باید کره سه بعدی در آن نمایش داده شود

۰,۳f : اندازه شعاع کره سه بعدی

ColorF : این تابع تعیین می کند که رنگ کره سه بعدی چه باشد؟ این تابع دارای سه پارامتر است که به طبع نیز شامل سه آرگومان از سمت چپ به راست قرمز $r=1$ و $g=1$ سبز و $b=0$ آبی است و کره سه بعدی با رنگ زرد رسم می شود

۱۲,۱f : این پارامتر تعیین می کند که چندثانیه کره سه بعدی در فضای سه بعدی نمایش داده شود و سپس بعد از ۱۲,۱ ثانیه کره از بین می رود

```
SEntitySpawnParams spawnParams;
spawnParams.pClass = gEnv->pEntitySystem-
>GetClassRegistry()->GetDefaultClass();

spawnParams.vPosition =hit.pt;

const float CxBomScale = ۰,۲۵f;
spawnParams.vScale = Vec3(CxBomScale);

 IEntity* pEntity = gEnv->pEntitySystem-
>SpawnEntity(spawnParams);

pEntity-
>CreateComponentClass<CCxBombComponent>();
```

اگر به صفحات فصول قبل مراجعه کنید، در رابطه با این چند خط کد توضیحات مفصل و خوبی را ارائه داده ام اما به صورت خلاصه بیان کنم که در نهایت یک مدل سه بعدی از یک **entity** بمب C۴ با تابع **ژنریک CreateComponentClass** با توجه به وجود کلاس **CC۴BombComponent** ایجاد می شود.

```

} // if GKeyAmmo

} // if spwan

} // if check press G key

});

m_pInputComponent->BindAction("player", "G",
eAID_KeyboardMouse, EKeyId::eKI_G);

```

تابع **BindAction** نیز در چند صفحه قبلی به صورت جامع توضیح داده ام و تمام عملیاتی که با زدن دکمه **G** بر روی صفحه کلید انجام می شود، پایان می یابد، در واقع زمانی که شما دکمه **G** بر روی صفحه کلید را می زنید، کارهای زیر انجام می شود:

۱- نقطه اشعه برخوردی و موقعیت پلیر محاسبه شده و در متغیر **hit** در هدر فایل **player.h** ریخته می شود.

۲- طبق متغیر **hit** و محتوای آن ، یک متن با عنوان **C۴ Bomb here** در نقطه برخوردی نمایش داده می شود.

۳- اگر تابع **AddSphere** را از توضیح خارج کنید، یک کره سه بعدی در نقطه برخوردی رسم می شود.

۴- مدل سه بعدی بمب C۴ به صورت یک entity در نقطه برخوردی ایجاد می شود

```
}
```

```
void
CPlayerComponent::SimulateExplosionRigidbody()
{
```

تابع SimulateExplosionRigidbody شبیه سازی انفجار را به دو صورت انجام می دهد :

۱- نیرویی برای فیزیک انفجار را با پرت کردن اشیاء در نقطه انفجار تولید می کند.

۲- پارتیکل سیستمی را به صورت آتش در نقطه انفجار تولید می کند (شما می توانید یک پارتیکل سیستم انفجار تولید کنید)

```
pe_explosion pee;
```

یک متغیر با نام pee را از نوع داده با ساختمان pe_explosion را تعریف کرده ام.

```
pee.epicenter = pee.epicenterImp = hit.pt;
```

مختصات مرکز نیروی انفجار و مختصات مرکز ضربه ای انفجار را برابر نقطه اشعه برخوردی در نظر گرفته ام.

```
pee.nGrow = ۲;
```

میزان رشد انفجار را بر اساس عدد ۲ تنظیم کرده ام.

```
pee.rminOcc = ۰.۰۷f;
```

حداقل نیروی تولید انفجار را برابر ۰,۰۷ نیوتن قرار داده ام

```
pee.rmin = pee.r = ۰,۵;
```

حداقل شعاع انفجار را و دامنه انفجار را به صورت ۵۰ سانتی متر یا همان ۰,۵ متر تنظیم کرده ام

```
pee.rmax = ۱۵;
```

حداکثر شعاع انفجار را برابر ۱۵ متر در نظر گرفته ام

```
pee.impulsivePressureAtR = ۲۵۰۰۰;
```

نیروی ضربه ای به نقطه انفجار را برابر ۲۵۰۰۰ نیوتن در نظر گرفته ام تا اشیاء بهتر و شدیدتر پرت شوند، شما می توانید نیروی کمتری را اختصاص دهید، دقت کنید که از اعداد اغراق آمیز استفاده نکنید، اینجا فقط بحث آموزشی است.

```
gEnv->pPhysicalWorld->SimulateExplosion(&pee);
```

حالا با توجه به تابع `SimulateExplosion` در کتابخانه `pPhysicalWorld` در ریشه تمام دستورات کرای انجین با `gEnv` عمل انفجار اتفاق می افتد.

```
}
```

حالا نوبت به اسلحه `F` یا همان `shotgun` انفجاری مرسوم تا بتوانم از تابع انفجاری که در بالا نوشته ام استفاده کنم و به بررسی تابع `FKey` می پردازم، این تابع از روی نامش مشخص است و زمانی که دکمه `F` بر روی صفحه کلید را می زنیم قابل استفاده خواهد بود، قبلا نیز در رابطه با توابع

مقیم در حافظه که باعث فعال سازی کلیدهای صفحه کلید میشد، توضیحات کافی و جامع ارائه نموده ام و از توضیح مجدد آن پرهیز می کنم.

```
void CPlayerComponent::FKey()
{

// Register the shoot action
m_pInputComponent->RegisterAction("player",
"FKey", [this](int activationMode, float value)
{
// Only fire on press, not release
if (activationMode == eIS_Pressed )
{
```

وقتی که دکمه F بر روی صفحه کلید فشار داده می شود، اعمال زیر اتفاق می افتند:

با توجه به وجود متغیر FKeyAmmo که در هدر فایل StdAfx.h تعریف شده است و خشاب اسلحه F را برعهده دارد، اگر مقدار متغیر FKeyAmmo کمتر یا مساوی صفر باشد، دو دستور العمل اجرا می شود :

۱- با توجه به تابع استاتیک تعریف شده در هدر فایل StdAfx.h با نام Null_AmmoFKey یک صدای تریگری با نام PublicPlayMySound پخش خواهد شد، این صدا همان صدای خشاب خالی اسلحه F است.

۲- مقدار متغیر FKeyAmmo صفر خواهد شد.

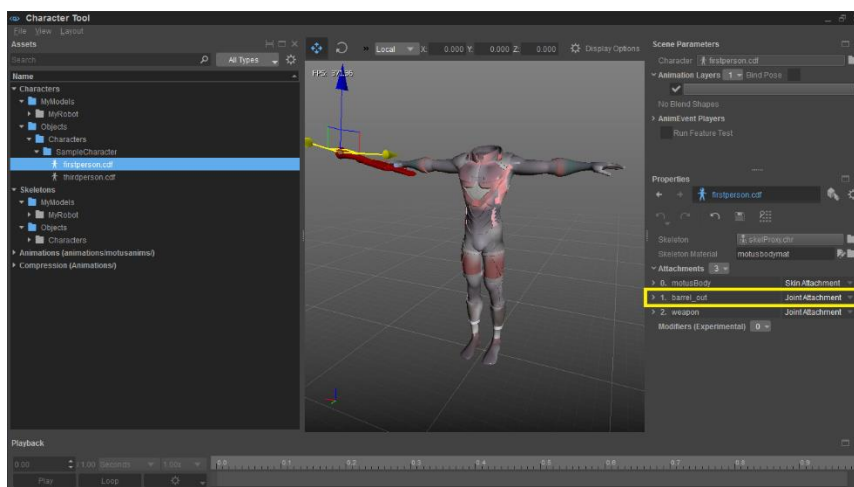
```
if(FKeyAmmo<=0)
{
//PublicPlayMySound("Null_AmmoFKey");
FKeyAmmo=0;
}
```

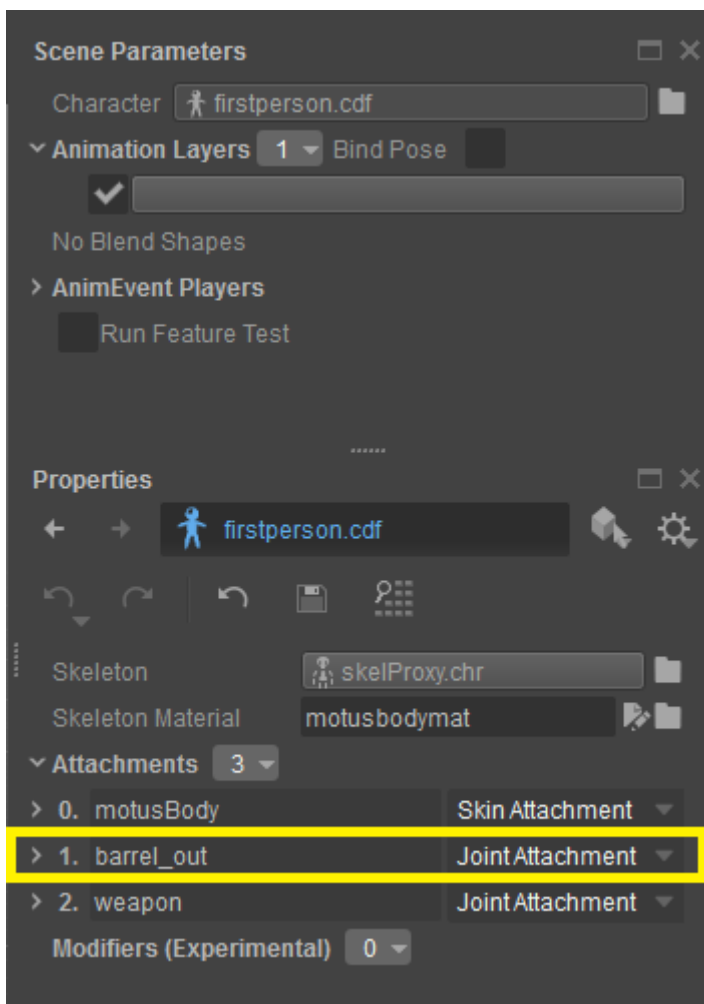
```
if (ICharacterInstance *pCharacter =
m_pAnimationComponent->GetCharacter())
{
```

بلوک if جاری، ابتدا یک متغیر با نام pCharacter از نوع داده ساختمان ICharacterInstance تعریف می شود که ساختمان جاری و ستون فقرات کاراکتر یا همان مدل سه بعدی پلیمر را بدست می گیرد

```
auto *pBarrelOutAttachment = pCharacter-
>GetIAttachmentManager()-
>GetInterfaceByName("barrel_out");
```

و سپس با استفاده از تعریف متغیر pBarrelOutAttachment نقطه ای از مکان تعریف شده با نام join ها را با توجه به پنجره Character Tool دریافت خواهد کرد، به کادر زرد رنگ تصویر پایین برگرفته از پنجره Character Tool دقت کنید که نام join مربوطه با تعریف رشته barrel_out در داخل کد C++ هم نام است.





با گرفتن نقطه یا `join` مربوط با نام `barrel_out` در متغیر `pBarrelOutAttachment` بلوک `if` دیگری بررسی می شود که اگر متغیر `pBarrelOutAttachment` خالی نباشد، به این معنی است که `join` مربوط با نام `barrel_out` در مدل پلیر وجود دارد و دستورالعمل های بلوک `if` اجرا خواهد شد

```
if (pBarrelOutAttachment != nullptr)
{
```

```
QuatTS bulletOrigin = pBarrelOutAttachment-
>GetAttWorldAbsolute();
```

یک متغیر با نام `bulletOrigin` تعریف شده است و وظیفه آن دریافت موقعیت جهانی نقطه یا `join` مربوط با نام `barrel_out` است، اگر چه چند خط کد پایین در اسلحه `F` و در این پروژه (`Robot Island`) کاربردی ندارد اما آن را توضیح دادم تا بیشتر با `API` های کرای انجین ۵،۴ آشنا شوید.

```
SEntitySpawnParams spawnParams;
spawnParams.pClass = gEnv->pEntitySystem-
>GetClassRegistry()->GetDefaultClass();
```

```
spawnParams.vPosition = bulletOrigin.t;
spawnParams.qRotation = bulletOrigin.q;
```

```
const float bulletScale = ۰.۰۵f;
spawnParams.vScale = Vec3(bulletScale);
```

چند خط کد بالا نیز در فصول قبلی توضیحات جامعی برای آن بیان نمودم و از توصیف و شرح مجدد آن پرهیز میکنم، نکته ای که نباید فراموش کنید، خطوط کد بالا در مسیر `template` های موجود در پوشه `cpp` در پروژه `FirstPersonShooter` کاربرد پرتاب توپ در نقطه یا `join` نامبرده استفاده شده است و در این پروژه و در این کتاب کاربردی ندارد اما همانطور که اشاره کردم، برای آشنایی شما `API` های مربوط به آن را توضیح دادم.

```
CRYENGINE_۵،۴\Templates\cpp\FirstPersonShooter
```

حالا به بلوک `if` بعدی می‌رسیم تا بتوانیم مرحله پایانی اسلحه `F` را توضیح دهیم، در این بلوک `if` بررسی می‌شود که آیا عمل شلیک در حال انجام است، که اگر مقدار متغیر `isFire` درست یا همان `true` باشد، دستورالعمل‌های زیر انجام می‌شود:

```
if (isFire)
{
```

```
PublicPlayMySound("bullet_impact");
```

پخش صدایی با نام `bullet_impact` که همان صدای شلیک اسلحه `F` است

```
SpawnParticleSystemHit();
```

تابع `SpawnParticleSystemHit` اجرا می‌شود و کار آن تولید پارتیکل سیستم انفجاری است.

```
SimulateExplosionRigidbody();
```

تابع `SimulateExplosionRigidbody` اجرا می‌شود و کار آن تولید فیزیک انفجار و پرت کردن اشیاء است.

```
isFire = false;
```

و در نهایت متغیر `isFire` مقدارش `false` می‌شود تا زمانی که مقدارش به `true` برنگردد، شما اجازه شلیک با اسلحه `F` را ندارید، در واقع باید تاخیر زمانی بین شلیک‌های این اسلحه وجود داشته باشد تا بتوانیم `shotgun` انفجاری را شبیه‌سازی کرده باشیم.

```

    } // isFire
    }
    }
    }
    });

    // Bind the shoot action to left mouse click
    m_pInputComponent->BindAction("player",
    "FKey", eAID_KeyboardMouse, EKeyId::eKI_F /*
    EKeyId::eKI_Mouse\ */);

```

در پایان نیز عمل مقیم سازی دکمه F بر روی صفحه کلید با تابع BindAction اتفاق می افتد

```

}

uint64 CPlayerComponent::GetEventMask() const
{
    return BIT64(ENTITY_EVENT_START_GAME) |
    BIT64(ENTITY_EVENT_UPDATE);
}

```

رویداد سیستمی GetEventMask دو عمل را برای تابع گزاری و اجرای دستورالعمل ها انجام می دهد که در فصول قبل توضیح کاملی ارائه شد، این رویداد با ماکرو آرگومانی ENTITY_EVENT_START_GAME باعث یکبار اجرا کدها می شود و ماکرو دیگر با عنوان ENTITY_EVENT_UPDATE از هنگام زمان شروع بازی تا هنگام زمان پایان بازی کدها یا همان دستورالعمل ها اجرا خواهند شد

```

void
CPlayerComponent::ProcessEvent(SEntityEvent&
event)
{

```

```
switch (event.event)
{
case ENTITY_EVENT_START_GAME:
{
Revive();
```

تعریف مجدد محتوای متغیرها و صفر کردن آنها و همچنین آغاز مجدد اجرای کدها را به صورت فقط یکبار با تابع **Revive** در رابطه با پلیر انجام می شود، این تابع با نام **Revive** وقتی اجرا می شود که گیم-پلی بازی شروع شده باشد

```
}
break;
case ENTITY_EVENT_UPDATE:
{
SEntityUpdateContext* pCtx =
(SentityUpdateContext*)event.nParam[.];
```

با توجه به متغیر **pCtx** باید برش های زمانی را دریافت کنیم

```
UpdateMovementRequest(pCtx->fFrameTime);
```

تابع **UpdateMovementRequest** که برش های زمانی را به صورت آرگومان دریافت می کند، وظیفه اش شبیه سازی حرکت فیزیکی پلیر است

```
UpdateLookDirectionRequest(pCtx->fFrameTime);
```

تابع **UpdateLookDirectionRequest** که برش های زمانی را به صورت آرگومان دریافت می کند وظیفه اش ورودی دستگاه ماوس برای کنترل دوربین است

```
UpdateAnimation(pCtx->fFrameTime);
```

تابع `UpdateAnimation` که برش های زمانی را به صورت آرگومان دریافت می کند وظیفه اش به روز رسانی کلیپ های انیمیشنی پلیر است

```
UpdateCamera(pCtx->fFrameTime);
```

تابع `UpdateCamera` که برش های زمانی را به صورت آرگومان دریافت می کند وظیفه اش به روز رسانی چرخش دوربین و پردازش دوربین پلیر را برعهده دارد

```
OnUpdate(pCtx->fFrameTime);
```

تابع `OnUpdate` که برش های زمانی را به صورت آرگومان دریافت می کند وظیفه اش کنترل، مدیریت و استفاده از اسلحه های پلیر را برعهده دارد

```
}
break;
}
}
```

```
void CPlayerComponent::AKShooting()
```

```
{
```

تابع `AKShooting` عمل شبیه سازی مسلسل، کلاشینکف یا اسلحه آتش زا را انجام می دهد.

```
if(isAKActive && JKeyAmmo>=۱)
```

```
{
```

بلوک `if` با توجه به متغیر `isAKActive` که می تواند مقدار `true/false` را داشته باشد و با توجه به عملگر منطقی `and` با دو کاراکتر `&&` و متغیر `JKeyAmmo` که مقدارش باید بزرگتر یا مساوی یک باشد، عمل شبیه سازی شلیک های پی در پی و پشت سرهم را بوسیله تابع `SpawnParticleSystemHit` انجام می دهد

```
SpawnParticleSystemHit۳();
```

```
JKeyAmmo--;
```

و بلافاصله متغیر JKeyAmmo یک واحد، یک واحد از آن کاسته می شود

```
}  
}
```

تابع `OnUpdate` بیشترین کاربرد را برای کنترل، مدیریت و استفاده از اسلحه های مختلف را در این پروژه برعهده دارد، همانطور که می دانید این پروژه برگرفته از `Template` آموزشی با ژانر اول شخص یا همان `FirstPersonShooter` در پوشه `Template` ها در کرای انجین است، همانطور که می بینید برش های زمانی به صورت آرگومان به تابع `UpdateOn` ارسال شد و حالا برش های زمانی در تابع `UpdateOn` به صورت پارامتر با نوع داده اعشاری `float` و اسم پارامتری `deltaTime` دریافت می شود، می دانید که آرگومان داده ای یا داده هایی را به تابع ارسال می کند و پارامتر داده ای یا داده هایی را به تابع دریافت می کند

```
void CPlayerComponent::OnUpdate(float  
deltaTime)  
{
```

```
RayCast();
```

تابع `RayCast` در داخل تابع `OnUpdate` اجرا می شود، تابع `RayCast` محاسبه اشعه های برخوردی را برای اسلحه ها انجام می دهد

```
AKShooting();
```

تابع `AKShooting` در داخل تابع `OnUpdate` اجرا می شود، تابع `AKShooting` شبیه سازی شلیک های پی در پی اسلحه هایی مانند مسلسل، کلاشینکف، آتش زا را انجام می دهد

```
//RayCastPoint(deltaTime);
```

تابع RayCastPoint نیز مانند RayCast است اما به حالت توضیح درآمده است، یعنی اجرا نمی شود و تابع RayCastPoint داخل تابع OnUpdate است

```
ControlFKey(deltaTime);
```

تابع ControlFKey در داخل تابع OnUpdate اجرا می شود، تابع ControlFKey نحوه شلیک اسلحه F را با بازه های زمانی ثابت انجام می دهد یعنی تاخیر در هر بار شلیک را محاسبه و اجرا می کند

```
GetDirCamera();
```

تابع GetDirCamera در داخل تابع OnUpdate اجرا می شود، تابع GetDirCamera جهت دوربین پلیمر را محاسبه می کند و متغیر ahmadval را به روزرسانی می کند.

```
//DoingKillFireCondition(hit);
```

تابع DoingKillFireCondition در داخل تابع OnUpdate اجرا می شود، تابع DoingKillFireCondition به حالت توضیح درآمده است و اجرا نمی شود، تابع DoingKillFireCondition عمل در حال آتش گرفتن پلیمر را شبیه سازی می کند، مثلاً وقتی پلیمر به یک آتش نزدیک شود، پلیمر نیز آتش بگیرد یا هنگام استفاده از اسلحه آتش زا و اسلحه آتش یا هر اسلحه دیگری پلیمر شروع به آتش گرفتن بگیرد

```
ShowWeaponAmmo();
```


تابع `ShowWeaponAmmo` در داخل تابع `OnUpdate` اجرا می شود، تابع `ShowWeaponAmmo` نمایش خشاب اسلحه ها را برعهده دارد

```
}

```

```
void CPlayerComponent::ShowWeaponAmmo()
{
    gEnv->pAuxGeomRenderer->Draw2dLabel(۱۰, ۱۰, ۲,
    ColorF(۱, ۰, ۰), false, "Explosion Point : " +
    ToString(FKeyAmmo));
}
```

با استفاده از تابع `Draw2dLabel` که `y=۱۰` ، `x=۱۰` و اندازه فونت برابر ۲ و رنگ قرمز با آرگومان `ColorF` و آرگومان دیگری با مقدار `false` و عنوان رشته با محتوای دلخواه و متغیر `FKeyAmmo`، یک متن با محتوای میزان خشاب اسلحه `F` بر روی صفحه نمایش، نشان داده می شود

```
gEnv->pAuxGeomRenderer->Draw2dLabel(۱۰, ۳۰, ۲,
ColorF(۱, ۰, ۰), false, "C۴ Bomb : " +
ToString(GKeyAmmo));
```

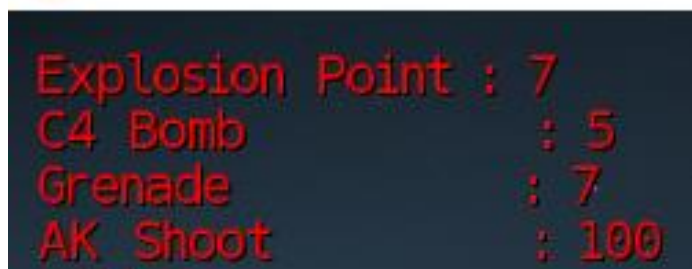
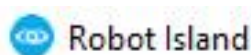
با استفاده از تابع `Draw2dLabel` که `y=۳۰` ، `x=۱۰` و اندازه فونت برابر ۲ و رنگ قرمز با آرگومان `ColorF` و آرگومان دیگری با مقدار `false` و عنوان رشته با محتوای دلخواه و متغیر `GKeyAmmo`، یک متن با محتوای میزان خشاب اسلحه `G` بر روی صفحه نمایش، نشان داده می شود

```
gEnv->pAuxGeomRenderer->Draw2dLabel(۱۰, ۵۰, ۲,
ColorF(۱, ۰, ۰), false, "Grenade : " +
ToString(HKeyAmmo));
```

با استفاده از تابع `Draw2dLabel` که `y=۵۰` و `x=۱۰` و اندازه فونت برابر ۲ و رنگ قرمز با آرگومان `ColorF` و آرگومان دیگری با مقدار `false` و عنوان رشته با محتوای دلخواه و متغیر `HKeyAmmo`، یک متن با محتوای میزان خشاب اسلحه `H` بر روی صفحه نمایش، نشان داده می شود

```
gEnv->pAuxGeomRenderer->Draw2dLabel(۱۰, ۷۰, ۲,
ColorF(۱, ۰, ۰), false, "AK Shoot : " +
ToString(JKeyAmmo));
```

با استفاده از تابع `Draw2dLabel` که `y=۷۰` و `x=۱۰` و اندازه فونت برابر ۲ و رنگ قرمز با آرگومان `ColorF` و آرگومان دیگری با مقدار `false` و عنوان رشته با محتوای دلخواه و متغیر `JKeyAmmo`، یک متن با محتوای میزان خشاب اسلحه `J` بر روی صفحه نمایش، نشان داده می شود



```
}
```

```
void
CPlayerComponent::DoingKillFireCondition(ray_h
it hit)
{
```

```
IParticleEffect* pEffect = gEnv->pParticleManager->FindEffect("fire.pfx");
```

یک متغیر `pEffect` تعریف کردم که پارتیکل سیستم `fire.pfx` را در پنجره `Asset Browser` پیدا می کند

```
if (pEffect)
{
    if
```

در صورتی که پارتیکل سیستم `fire.pfx` وجود داشته باشد، بلوک `if` اجرا می شود و کدهای زیر اجرا خواهند شد

```
pHitEntity = hit.pCollider;
```

متغیر `pHitEntity` دارای محتوایی است که منشاء آن اشعه برخوردی است که آن اشعه با استفاده از متغیر `hit`، شی ای که فیزیک `collider` و `rigidbody` را دارند در خود حمل می کند.

```
pEntity = gEnv->pEntitySystem->GetEntityFromPhysics(pHitEntity);
```

با این خط کد با استفاده از تابع `GetEntityFromPhysics`، شی ای را دریافت می کند که حتما باید فیزیک `collider` و `rigidbody` را داشته باشد، سپس شی در داخل متغیر `pEntity` قرار می گیرد.

```
if (pEntity)
{
```

در صورتی که متغیر `pEntity` دارای شی فیزیک با کامپوننت `collider` و `rigidbody` باشد، خطوط کد زیر با استفاده از بلوک `if` اجرا خواهد شد.

```
pClass = pEntity->GetClass();
```

با استفاده از تابع `GetClass` از شی `pEntity`، کلاس موردنظر شی در داخل متغیر `pClass` قرار می گیرد

```
pNameClass = pClass->GetName();
```

و سپس با استفاده از تابع `GetName` نام کلاس شی در داخل متغیر `pNameClass` قرار میگیرد

```
}
```

```
// CryLog("ClassName : "+ pNameClass);
```

این خط توضیح نیز اگر فعال شود، نام کلاس شی ای که اشعه برخوردی به آن تابش داده شده باشد در پنجره `output` ویزوال استودیو یا در پنجره `console` کرای انجین نمایش داده می شود، در اینجا هدف بیشتر آشنایی با توابع کرای انجین است و این خط به حالت توضیح درآورده ام و با استفاده از تابع `CryLog` می توانید اسم کلاس را نمایش دهید

```
}
```

```
if (pEntity && hit.dist >= . && hit.dist <= ۴.f  
&& pNameClass == "Entity" && pEntity->  
>GetOpacity() == ۱,۲f)  
{
```

بلوک `if` در این خط کد، کمی طولانی است و نیاز به توضیح بیشتر دارد، این بلوک از ۵ شرط تشکیل شده است و شرح آن به این صورت است :

`pEntity` : دارای شی ای که فیزیک `collider` و `rigidbody` داشته باشد

`hit.dist >= ۰` : فاصله پلیمر از اشعه برخوردی (عمق و طرف جلو) بزرگتر از صفر یا مساوی صفر باشد.

`hit.dist <= ۴.f` : فاصله پلیمر از اشعه برخوردی (عمق و طرف جلو) کوچکتر از ۴ یا مساوی ۴ باشد.

این دو به این معنی است که پلیمر در بازه صفر تا ۴ متر کمتر باشد و عملی اتفاق بیفتد.

`pNameClass == "Entity"` : نام کلاس در رابطه با اشعه برخوردی بر روی هر سطحی از زمین یا غیرزمین باید از نوع `Entity` باشد

`pEntity->GetOpacity() == ۱,۲f` : شفاف شدن پلیمر یا همان میزان قابل مرئی بودن پلیمر با استفاده از عملگر برابری با ۱,۲ شود، عملی اتفاق بیفتد.

اگر این ۵ شرط برقرار باشند، کدهای زیر اجرا می شوند و اگر یکی از این ۵ شرط برقرار نباشد، کدهای زیر اجرا نخواهند شد

```
IParticleEffect *pEffect = gEnv->pParticleManager->FindEffect("fire.pfx");
```

مانند مثال قبلی یک پارتیکل را پیدا می کند

```
KillPlayerFire=true;
```

متغیر `KillPlayerFire` دارای محتوای `true` می شود، یعنی پلیمر باید آتش بگیرد

```
GetEntity()->FreeSlot(۰);
```

با توجه به وجود `slot۰`، باید آزاد شود چون پارتیکل `Blank.pfx` را به همراه داشت و باید الان پارتیکل دیگری را در `slot۰` لود نماییم.

```
GetEntity()->LoadParticleEmitter(۰, pEffect);
```

پارتیکل `fire.pfx` لود می شود و پلیر شروع به آتش گرفتن خواهد شد

دقت کنید که دایره و محدوده تعریف متغیرها در داخل بلوک ها به بلوک ها وابسته اند و هر چه متغیرها داخل بلوک های تودرتو تر قرار گیرند، عمر متغیرها کوتاه تر خواهند بود و هر چه متغیر از بلوک های تودرتو فاصله بگیرد و خارج تر از بلوک ها تعریف شوند، عمر متغیرها زیاد تر خواهد شد.

```
}
```

```
}
```

```
void  
CPlayerComponent::SpawnParticleSystemHit۲()  
{
```

تابع `SpawnParticleSystemHit۲` در بازی استفاده نشده است اما عملکرد آن را توضیح می دهیم

```
if (IParticleEffect *pEffect = gEnv->
pParticleManager->FindEffect("fire.pfx",
"Particles"))
{
```

در صورتی که اگر متغیر `pEffect` پارتيكل سيستم `fire.pfx` را پیدا کند و نوع آن هم از داده های `Particles` است، بلوک `if` بعدی اجرا خواهد شد

```
if (!pParticleEntity)
{
```

در صورتی که اگر متغیر `pParticleEntity` خالی باشد و هیچ پارتيكلي داخل آن نباشد، کدهای زیر اجرا می شوند.

```
SEntitySpawnParams spawn;
spawn.pClass = gEnv->pEntitySystem->
GetClassRegistry()->GetDefaultClass();
pParticleEntity = gEnv->pEntitySystem->
SpawnEntity(spawn);
```

کلاسی پیش فرض از نوع پارتيكل توليد شود و آن را تکثير می کند، این خطوط قبلا بارها و بارها توضيح داده شده است.

```
}
if (pParticleEntity)
{
```

متغیر `pParticleEntity`، حالا دارای پارتیکل است و با بررسی شرط `if`، کدهای زیر اجرا می شوند

```
pParticleEntity->SetPos(hit.pt);
```

پارتیکل تولید شده که حالا وجود دارد، باید به موقعیت فضای سه بعدی در مختصات اشعه برخوردی به وسیله متغیر `hit.pt` انتقال داده شود، یعنی پارتیکل سیستم به مکان `hit.pt` تغیر پوزیشن می دهد

```
pParticleEntity->FreeSlot(.);
```

اسلات صفر آزاد می شود تا پارتیکل های قبلی از بین برود (این ترفند بهینه سازی است که اسلات ها را با تابع `FreeSlot` آزاد کنیم).

```
pParticleEntity->LoadParticleEmitter(. ,  
pEffect);
```

در اسلات صفر و متغیر پارتیکلی که کلاس پارتیکل تکثیر شد، حالا پارتیکل `fire.pfx` در متغیر `pEffect` لود می شود

تابع `SpawnParticleSystemHit3` تکامل یافته تابع `SpawnParticleSystemHit2` است، پیشنهاد می کنم که برای انجام عمل تکثیر پارتیکل انفجار یا آتش یا هر پارتیکل دیگر از تابع `SpawnParticleSystemHit3` استفاده کنید، زیرا که تابع `SpawnParticleSystemHit2` مستعد باگ است و در تابع `SpawnParticleSystemHit3` هیچ باگ تکثیری در ایجاد پارتیکل ها وجود ندارد.


```

    }
    }
    }

```

```

void
CPlayerComponent::SpawnParticleSystemHit3()
{

```

```

    SEntitySpawnParams spawnParams2;
    spawnParams2.pClass = gEnv->pEntitySystem-
    >GetClassRegistry()->GetDefaultClass();
    spawnParams2.vPosition = hit.pt;

```

حالا یک بار دیگر یک متغیر با نام `spawnParams2` را تعریف می کنیم و یک کلاس را دریافت کرده و عمل ایجاد متغیر کلاس را برای `entity` پارتيكل ایجاد می کنیم، این چند خط کد بالا بارها و بارها توضیح داده ام

```

const float ParticleScale = ۰.۱f;

```

مقیاس پارتيكل با توجه به متغیر `ParticleScale` تغییر اندازه نمی شود، یعنی اسپریت ها بزرگ و کوچک نمی شوند، این متغیر بر روی فیزیک پارتيكل تاثیر می گذارد.

```

spawnParams2.vScale = Vec3(ParticleScale);
IEntity* pEntity = gEnv->pEntitySystem-
>SpawnEntity(spawnParams2);

```

حالا کلاسی را با استفاده متغیر `pEntity` به وسیله تابع `SpawnEntity` ایجاد می کنیم و در نهایت `entity` را در خط بعدی ایجاد می کنیم.

```
pEntity->CreateComponentClass<CParticle۰۱>();
```

با استفاده از تابع ژنریک `CreateComponentClass` کلاس `CParticle۰۱` را ایجاد می کنیم و در متغیر `pEntity` قرار می گیریم، فراموش نکنید که هدر فایل `*.h` و سی پی پی فایل `*.cpp` باعث ایجاد کلاس های جدید می شوند، کلاس `CParticle۰۱` از هدر فایل `Particle۰۱.h` و `Particle۰۱.cpp` نشأت می گیرد.

```
pHitEntity = hit.pCollider;
pEntity = gEnv->pEntitySystem-
>GetEntityFromPhysics(pHitEntity);
```

در چند صفحه قبل این دو خط کد را توضیح دادم و باعث می شود که هر شی که با توجه به اشعه برخوردی `hit.pCollider` دارای فیزیک `entity` مربوط به کامبونت های `collider` و `rigidbody` داشته باشد، آدرس `entity` آن در متغیر `pEntity` قرار می گیرد

```
if (pEntity)
{
    اگر شی pEntity دارای فیزیک کامبونت collider و rigidbody باشد، شی مورد نظر نامرئی و مخفی خواهد شد
```

```
pEntity->Invisible(true);
}
```

```

/*
SEntitySpawnParams spawnParams۳;
spawnParams۳.pClass = gEnv->pEntitySystem-
>GetClassRegistry()->GetDefaultClass();
spawnParams۳.vPosition = hit.pt;
const float ParticleScale۳ = ۰.۲۵f;
spawnParams۳.vScale = Vec۳(ParticleScale۳);

IEntity* pEntity۲ = gEnv->pEntitySystem-
>SpawnEntity(spawnParams۳);

pEntity۲-
>CreateComponentClass<CC۴BombComponent>();

*/

```

این چند خط کد که با رنگ سبز به صورت توضیح درآورده ام، عمل تکثیر کلاس با استفاده از متغیر کلاسی را برای ساخت **entity** که با نام بمب C۴ است، انجام خواهد شد، فراموش نکنید که ساخت هر **entity** به هدر فایل و سی پی پی فایل وابسته است و در اینجا بمب C۴ نشأت گرفته از فایل های C۴Bomb.h و C۴Bomb.cpp است.

```

}

void CPlayerComponent::DoneKillFireCondition()
{
pEntity->SetOpacity(۱.۲f);

```

تابع `DoneKillFireCondition` عمل کشته شدن پلیر طبق آتش گرفتن پلیر را با مقداردهی کردن شفافیت `entity` با تابع `SetOpacity` انجام می دهد، این تابع در این پروژه به کار گرفته نشده است اما قصد من این است که با API های کرای انجین بیشتر آشنا شوید، به کار گیری این تابع داخل بازی بسیار ساده است

```
}
```

```
void CPlayerComponent::SpawnParticleSystemHit()
{
```

تابع `SpawnParticleSystemHit` بسیار شبیه تابع `SpawnParticleSystemHit` است و ایجاد متغیر کلاسی، تکثیر کلاس و در نهایت تکثیر پارتيكل را به همراه دارد، این ساختار قبلا به خوبی در این کتاب توضیح داده ام اما دوباره آن را توضیح می دهم زیرا که دارای نکته ظریفی است که زمان آن رسیده است به آن بپردازم

```
IParticleEffect* pEffect = gEnv-
>pParticleManager->FindEffect("fire.pfx");
```

```
if (pEffect)
{
    pHitEntity=hit.pCollider;
    pEntity=gEnv->pEntitySystem-
>GetEntityFromPhysics(pHitEntity);
```

```
if(pEntity)
{
    // DoneKillFireCondition();
```

در صورتی که با اشعه برخوردی، شی فیزیک collider و rigidbody در داخل متغیر pEntity شناسایی و قرارگیری شد، بلوک if اجرا می شود و تابع DoneKillFireCondition عمل کشته شدن پلیر برای آتش گرفتن پلیر انجام می دهد اما همانطور که می بینید تابع DoneKillFireCondition به حالت توضیح درآورده ام و این تابع در این پروژه استفاده نمی شود، اگرچه با انجام تغییراتی در سطح پروژه این تابع و دیگر توابع تبدیل شده به توضیح قابل استفاده خواهند بود

```
pEntity->LoadParticleEmitter(·,pEffect);
```

در slot· با توجه به تابع LoadParticleEmitter، پارتیکل آتش لود می شود

```
pEntity->Invisible(true);
```

با استفاده از تابع Invisible که مقدار آن true است، entity که دارای فیزیک کامپوننت های collider و rigidbody است، نامرئی و مخفی خواهد شد، این عمل نشان می دهد که entity از بین رفته است

البته بهتر است روش هوشمندانه تری به کار ببندید و به جای استفاده از تابع Invisible از خط زیر استفاده کنید، اگر می خواهید entity تان از بین برود :

```
gEnv->pEntitySystem->RemoveEntity(pEntity->GetId());
```

نکته بسیار جالب دیگر در این بخش این است که اگر شما از دو تابع زیر استفاده نکنید :

```
pEntity->Invisible(true);
gEnv->pEntitySystem->RemoveEntity(pEntity-
>GetId());
```

آتش تولید شده به entity که کامبونت های collider و rigibody را دارد، اضافه شده و نوعی الصاق و attach شدن اتفاق می افتد

```
}
}
```

```
SEntitySpawnParams spawnParams۳;
spawnParams۳.pClass = gEnv->pEntitySystem-
>GetClassRegistry()->GetDefaultClass();
spawnParams۳.vPosition = hit.pt;
```

```
const float ParticleScale۳ = ۰.۲۵f;
spawnParams۳.vScale = Vec۳(ParticleScale۳);
```

```
Ientity* pEntity۲ = gEnv->pEntitySystem-
>SpawnEntity(spawnParams۳);
```

```
pEntity۲->CreateComponentClass<CParticle۰۱>();
```

به غیر از آتش تولید شده توسط تابع LoadParticleEmitter، من یک پارتيكل را با استفاده از دستورالعمل های بالا تکثير می کنم که

دارای فیزیک است و می تواند بعد از تولید آتش ضربه را دریافت کنید، این بخش نیز کلاسی برای تولید `entity` با استفاده از متغیر `spawnParams` از نوع `SEntitySpawnParams` و تابع `GetDefaultClass` برگرفته از `GetClassRegistry` انجام می شود و با تابع `SpawnEntity` عمل تکثیر کلاس انجام می شود و متغیر `pEntity` از نوع اشاره گری آماده ایجاد `entity` خواهد بود و در نهایت تابع ژنریک `CreateComponentClass` از متغیر `pEntity` کلاس `CParticle` را به `entity` پارسیکل سیستم تبدیل می کند

```
}
```

```
void CPlayerComponent::RayCastPoint(float
deltaTime)
{
```

تابع `RayCastPoint`، اگرچه در این پروژه استفاده نشده است اما حاوی API های مهم در کرای انجین است که کاربرد آن را در اینجا می توانید ببینید

```
IPersistentDebug *iPD = gEnv->pGameFramework-
>GetIPersistentDebug();
```

ابتدا یک متغیر اشاره گر از نوع داده `IPersistentDebug` با نام `iPD` تعریف می کنم که در خطوط بعدی کدها از آن استفاده خواهیم کرد.

```
gEnv->pAuxGeomRenderer->DrawSphere(ahmadval,
۰٫۲, ColorF(۱, ۰, ۰));
```

در مختصات بردار سه بعدی `ahmadval` با استفاده از تابع `DrawSphere`، کره ای قرمز رنگ به اندازه شعاع ۲۰ سانتی متری (۲، ۰، ۰ متر) رسم می شود.

```
sum += deltaTime;
```

با استفاده از متغیر `sum` برش های زمانی (`deltaTime`) را جمع می کنیم.

```
if (sum > ۰ && sum <= ۲)
```

```
{
```

اگر متغیر `sum` در بازه زمانی صفر الی دو ثانیه قرار گیرد، دستورالعمل های زیر اجرا خواهند شد.

```
iPD->Begin("Great Work Ahmad!", false);
```

متغیر `iPD` که در بالا برای اختصاص فضایی برای رسم اطلاعات بر روی صفحه نمایش تعریف شده است، حالا با استفاده از تابع `Begin` شروع به کار می کند، رشته ای دلخواه به آن اختصاص دهید و مقدار `false` را به آن اختصاص دهید.

```
iPD->AddText3D(Vec3(۶۰, ۶۰, ۳۳), ۲, ColorF(۰, ۰, ۰), ۱۰, "Ahmad Karami ;-);
```

تابع `AddText3D`، متنی سه بعدی را رسم می کند که در مختصات بردار سه بعدی `Vec3(۶۰, ۶۰, ۳۳)` قرار می گیرد که اندازه فونت متن ۲ واحد و رنگ متن به صورت سیاه و `Ahmad Karami` نمایش داده می شود.


```
gEnv->pAuxGeomRenderer->Draw2dLabel(۱۰۰, ۵۰, ۳,
ColorF(۱, ۰, ۰), false, "Ahmad Karami");
```

با استفاده از تابع `Draw2dLabel` یک متن دو بعدی با عنوان Ahmad Karami در مختصات پیکسلی صفحه نمایش با توجه به $x=۱۰۰$ و $y=۵۰$ ، اندازه فوت ۳ واحد و رنگ متن قرمز با آرگومان پنجم که `false` است ،رسم می شود

```
gEnv->pAuxGeomRenderer->DrawSphere(Vec3(۶۰, ۶۰,
۳۳), ۱, ColorF(۰, ۱, ۰));
```

با توجه به تابع `DrawSphere` در مختصات بردار سه بعدی `Vec3(۶۰, ۶۰, ۳۳)` و شعاع ۱ متری و رنگ سبز اختصاصی، کره ای سه بعدی رسم می شود

```
}
```

```
if (sum >= ۳)
```

```
{
```

با بزرگتر شدن یا مساوی سه شدن جمع برش های زمانی در متغیر `sum` ،بلوک `if` اجرا خواهد شد

```
sum = -۲;
```

متغیر `sum=-۲` می شود.

```
iPD->Reset();
```

همه متن و اشکال هندسی با استفاده از تابع `Reset` از متغیر `iPD` پاک می شوند.

این عمل به صورت متناوب نمایش داده می شود و مانند چشمک زدن ثابت این پاک شدن و رسم شدن متن و اشکال هندسی اتفاق می افتند.

```
}
}
```

تابع `AllShiftKey` تغییر سرعت پلیر را محاسبه می کند، به این صورت که زمانی دکمه `shift` بر روی صفحه کلید زده شد، سرعت پلیر افزایش می یابد و زمانی که دکمه `shift` رها شد، سرعت پلیر کاهش می یابد

```
void CPlayerComponent::AllShiftKey()
```

```
{
```

در صفحات قبلی توضیحات کافی برای استفاده از صفحه کلید ارائه شد و عملکرد تابع `RegisterAction` بیان شد.

```
m_pInputComponent->RegisterAction("player",
"LShift", [this](int activationMode, float
value) {
```

```
if (activationMode == eIS_Pressed)
```

```
{
```

در این خط کد، زمانی که دکمه `shift` سمت چپ بر روی صفحه کلید زده شد، متغیر `moveSpeed` مقدارش به ۱۰۰٫۵ تغییر کند

```
moveSpeed = ۱۰۰٫۵f;
```

```
}
```

```
if (activationMode == eIS_Released)
```

```
{
```

در این خط کد، زمانی که دکمه `shift` سمت چپ بر روی صفحه کلید رها شد، متغیر `moveSpeed` مقدارش به ۲۰,۵ بر می گردد

```
moveSpeed = ۲۰,۵f;
```

```
}
```

```
});
```

```
m_pInputComponent->BindAction("player",
    "LShift", eAID_KeyboardMouse,
    EKeyId::eKI_LShift );
```

با استفاده از تابع `BindAction` عمل مقیم سازی دکمه `shift` چپ بر روی صفحه کلید انجام می شود. حالا نوبت به دکمه `shift` سمت راست می رسد.

```
m_pInputComponent->RegisterAction("player",
    "RShift", [this](int activationMode, float
    value) {
    if (activationMode == eIS_Pressed)
```

```
{
```

در این خط کد، زمانی که دکمه `shift` سمت راست بر روی صفحه کلید زده شد، متغیر `moveSpeed` مقدارش به ۱۰۰,۵ تغییر کند

```
moveSpeed = ۱۰۰,۵f;
```

```
}
```

```

if (activationMode2 == eIS_Released)
{
    در این خط کد، زمانی که دکمه shift سمت راست بر روی صفحه کلید
    زده شد، متغیر moveSpeed مقدارش به ۲۰,۵ بر می گردد
    moveSpeed = ۲۰,۵f;
}

});

m_pInputComponent->BindAction("player",
    "RShift", eAID_KeyboardMouse,
    EKeyId::eKI_RShift);

```

با استفاده از تابع BindAction عمل مقیم سازی دکمه shift راست بر روی صفحه کلید انجام می شود

```

}
```

تابع ControlFKey کنترل شلیک ها توسط دکمه F بر روی صفحه کلید با تاخیرهای زمانی ثابت را برعهده دارد

```

void CPlayerComponent::ControlFKey(float
deltaTime)
{
    sum2 += deltaTime;
}

```

متغیر sum2 جمع برش های زمانی را با فاصله زمانی ثابت را انجام می دهد، مثلاً در اینجا هر ۲,۵ ثانیه یکبار شلیک گلوله انجام می شود، اینجا اسلحه shotgun انفجاری شبیه سازی میشود

```
if (sum۲ >= ۲,۵ && isFire==false && FKeyAmmo >=
۱)
{
```

با توجه به بلوک `if` سه شرط وجود دارد و با درستی این سه شرط دستورات عمل ها در این بلوک اجرا خواهند شد :

`sum۲ >= ۲,۵` : جمع برش زمانی در متغیر `sum` ، اگر بیشتر از ۲,۵ ثانیه شد ، شرط `true` خواهد شد.

`isFire==false` : زمانی که مجوز شلیک گلوله یعنی متغیر `isFire` برابر `false` باشد، شرط `true` خواهد شد

`FKeyAmmo >= ۱` : اگر میزان خشاب `shotgun` انفجاری یا همان متغیر `FKeyAmmo` بزرگتر از یک یا مساوی یک باشد، شرط `true` خواهد شد.

با درست شدن و برقراری سه شرط بالا ، دستورات عمل های زیر اجرا می شوند
`FKeyAmmo-- ;`

یک واحد از خشاب اسلحه `F` یا همان `shotgun` انفجاری کاهش می یابد
`sum۲ = ۰ ;`

متغیر `sum۲` یا همان جمع برش های زمانی صفر خواهد شد و تا ۲,۵ ثانیه بعدی با توجه به وجود این متغیر شلیک گلوله انجام نخواهد شد.

```
isFire = true;
```

مجوز شلیک گلوله باطل می شود تا بازه زمانی ثابت با متغیر `sum۲` فرا برسد.

```
}
}
```

تابع `UpdateMovementRequest` حرکت پلیر را با توجه به وجود کامپوننت فیزیک `CharacterController` محاسبه و اجرا می کند، این تابع نیز دارای برش زمانی با پارامتر `frameTime` است.

```
void
CPlayerComponent::UpdateMovementRequest(float
frameTime)
{

if (!m_pCharacterController->IsOnGround())
return;
```

در صورتی که پلیر در هوا بود عملی انجام نشود `m_pCharacterController->IsOnGround`، دلیل اینکه پلیر در هوا بود عملگر منطقی `!` است که شرط را معکوس می کند و عملی انجام نشود عبارت `return;` است.

```
Vec3 velocity = ZERO;
```

متغیر برداری `velocity` شتاب حرکتی برای پلیر را نگه می دارد، اما شتاب اولیه در اینجا صفر است و با عبارت `ZERO` قرار داده می شود یعنی `Vec3(0,0,0)` خواهد بود.

با بلوک `if` های مکرر برخورد می کنید و هر بلوک `if` عملکرد حرکتی پلیر را محاسبه و اجرا می کند، این حرکات به ترتیب زیر است:

پارامتر `frameTime` همان برش زمانی است که در فرمول حرکت در متغیر سرعت پلیر ضرب می شود و شتاب حاصله را به پلیر اختصاص می دهد

و از متغیر `velocity` استفاده می‌کنم زیرا منشاء حرکت پلیر را دربردارد.

$$a(\text{velocity}) = \frac{\text{meter}}{\text{second}} * \frac{1}{\text{second}} = \frac{\text{meter}}{\text{Second}^2}$$

در واقع بر اساس فرمول `a` محاسبه متغیر `velocity` را انجام داده و حرکت پلیر را اجرا می‌کنیم

در اینجا منظور از $\frac{\text{meter}}{\text{second}}$ همان متغیر `moveSpeed` است و منظور از $\frac{1}{\text{second}}$ همان پارامتر `frameTime` است

```
if (m_inputFlags &
    (TInputFlags)EInputFlag::MoveLeft)
{
    velocity.x -= moveSpeed * frameTime;
```

برای حرکت به سمت چپ از خصیصه اول یا `X` با منفی سازی متغیر `velocity` استفاده می‌کنم

```
}
if (m_inputFlags &
    (TInputFlags)EInputFlag::MoveRight)
{
    velocity.x += moveSpeed * frameTime;
```

برای حرکت به سمت راست از خصیصه اول یا x با مثبت سازی متغیر `velocity` استفاده می‌کنم

```

}
if (m_inputFlags &
(TInputFlags)EInputFlag::MoveForward)
{
velocity.y += moveSpeed * frameTime;

```

برای حرکت به سمت عمق یا روبه جلو از خصیصه دوم یا y با مثبت سازی متغیر `velocity` استفاده می‌کنم

```

}
if (m_inputFlags &
(TInputFlags)EInputFlag::MoveBack)
{
velocity.y -= moveSpeed * frameTime;

```

برای حرکت به سمت عمق یا روبه عقب از خصیصه دوم یا y با منفی سازی متغیر `velocity` استفاده می‌کنم

```

}

```

```

m_pCharacterController-
>AddVelocity(GetEntity()->GetWorldRotation() *
velocity);

```

با توجه به بردار `quat` گرفته شده تابع `GetWorldRotation` در متغیر `velocity` ضرب می‌شود و محاسبه اضافه شدن شتاب با تابع

AddVelocity اجرا می شود و باعث حرکت کامپوننت ChatacterController پلیر خواهد شد

```
}

```

با استفاده از تابع UpdateLookDirectionRequest جهت دوربین پلیر را محاسبه و اجرا می کنید، این تابع نیز دارای برش های زمانی با پارامتر frameTime است

```
void
CPlayerComponent::UpdateLookDirectionRequest(float frameTime)
{
    const float rotationSpeed = ۰,۰۰۲f;

```

ابتدا متغیر محلی rotationSpeed را با مقدار ثابت و بدون تغییر تعریف می کنم، کار این متغیر سرعت چرخش دوربین پلیر است

```
const float rotationLimitsMinPitch = -۰,۵۴f;

```

سپس متغیر ثابت و بدون تغییر rotationLimitsMinPitch را تعریف می کنم و کار این متغیر میزان محدودیتی که برای چرخش دوربین پلیر در بازه ی منفی وجود دارد، در واقع کمترین pitch را برای چرخش یا همان محور منفی X در نظر میگیرم.

```
const float rotationLimitsMaxPitch = ۱,۵f;

```

سپس متغیر ثابت و بدون تغییر rotationLimitsMaxPitch را تعریف می کنم و کار این متغیر میزان محدودیتی که برای چرخش دوربین

پلیر در بازه ی مثبت وجود دارد و در واقع بیشترین pitch را برای چرخش یا همان محور X مثبت در نظر میگیرم

```
if (m_mouseDeltaRotation.IsEquivalent(ZERO,
MOUSE_DELTA_TRESHOLD))
return;
```

این بلوک if برآورد می کند که میزان چرخش دوربین به عدد صفر میل کرده است و آیا دوربین در حال حرکت است؟ اگر در حال حرکت نبود بقیه دستورات تابع UpdateLookDirectionRequest اجرا نمی شود، در غیراینصورت دستورات تابع UpdateLookDirectionRequest اجرا خواهند شد

```
m_mouseDeltaRotation =
m_mouseDeltaSmoothingFilter.Push(m_mouseDeltaR
otation).Get();
```

این خط کد طولانی میزان برش های جزئی چرخش ماوس را با گرفتن و محاسبه کردن برای نشان دادن و صاف کردن آن انجام می دهد

```
m_horizontalAngularVelocity =
(m_mouseDeltaRotation.x * rotationSpeed) /
frameTime;
m_averagedHorizontalAngularVelocity.Push(m_hor
izontalAngularVelocity);
```

این خط کد طولانی تر است و نیاز به توضیحات بیشتری دارد، این خط میزان شتاب زاویه افقی را با توجه به چرخش بر اساس محور افقی ماوس

`m_mouseDeltaRotation.x` و ضرب سرعت چرخش `rotationSpeed` را بر برش های زمانی `frameTime` تقسیم می کند و میزان متوسط این چرخش زاویه ای را در حرکت ماوس ثبت می کند

```
Ang3 ypr =
CCamera::CreateAnglesYPR(Matrix33(m_lookOrienta
tion));
```

باید توجه داشته باشید که آخرین تغییرات ورودی ماوس در ایجاد زاویه درست برای دوربین باید محاسبه شود و در اینجا ماتریکس سه بعدی برای عمل چرخش و زاویه درست برای دوربین پلیر محاسبه می شود

```
ypr.x += m_mouseDeltaRotation.x *
rotationSpeed;
```

در این خط محور افقی ماوس با سرعت چرخش حرکت و برش های جزئی حرکت ماوس در دستگاه دکارتی پیکسلی محاسبه می شود

```
Ypr.y = CLAMP(ypr.y + m_mouseDeltaRotation.y *
rotationSpeed, rotationLimitsMinPitch,
rotationLimitsMaxPitch);
```

در این خط کد با توجه به ماکرو `CLAMP` فاصله بین کمترین مقدار و بیشترین مقدار برای حرکت عمودی ماوس با ثبت این حرکات در دوربین پلیر باید محاسبه شود، در آرگومان های ورودی سه گانه برای این ماکرو، میزان تغییرات فعلی با این فرمول `ypr.y + m_mouseDeltaRotation.y * rotationSpeed` محاسبه می شود و کمترین مقدار چرخش با ثابت `rotationLimitsMinPitch` و بزرگترین مقدار چرخش با ثابت

`rotationLimitsMaxPitch` انجام می شود، در نهایت حرکت عمودی ماوس برای دوربین پلیر ثبت می شود

```
ypr.z = ۰;
```

محور Z برای دوربین پلیر کاربرد ندارد مگر در موارد خاص که پلیر در یک مرحله به حالت جاسوسی در یک گوشه دیوار به آن طرف و در عمق مخفیانه سر را کج می کند و به نگاه کردن به محیط اطراف مشغول است

```
m_lookOrientation =  
Quat(CCamera::CreateOrientationYPR(ypr));
```

حالا `quat` مربوط به محورهای افقی X و عمودی Y در ماوس ثبت شود و به متغیر `lookOrientation` ریخته شود، در اینجا جهت نگاه دوربین برای پلیر با تابع `CreateOrientationYPR` محاسبه می شود

```
m_mouseDeltaRotation = ZERO;
```

با توجه به این خط کد، مقدار چرخش جزئی ماوس صفر می شود و دوباره باید محاسبات بالا لحاظ شود، این تغییرات با برش های زمانی در ارتباط بوده و اساس مفهوم حرکت ماوس را برای دوربین پلیر تشکیل می دهد

```
}
```

تابع `UpdateAnimation` عملیات به روزرسانی انیمیشن های پلیر را براساس پارامتر برش های زمانی `frameTime` انجام می دهد، تابع `UpdateAnimation` در ارتباط با سیستم انیمیشن پیشرفته مانکن ادیتور است.

```
void CPlayerComponent::UpdateAnimation(float  
frameTime)  
{
```

```
const float angularVelocityTurningThreshold =
    ۰٫۱۷۴;
```

میزان شتاب چرخش زاویه ای انیمیشن پلیر با ثابت `angularVelocityTurningThreshold` در نظر گرفته می شود که واحد آن رادیان در ثانیه است

```
const bool isTurning =
    std::abs(m_averagedHorizontalAngularVelocity.Get()) > angularVelocityTurningThreshold;
```

با توجه به میزان چرخش افقی متوسط شتاب زاویه ای ماوس برای تغییرات مطلق مثبت با تابع `abs` محاسبه می شود، در اینجا بررسی می شود که آیا عمل چرخش `isTurning` اتفاق افتاده است یا نه؟

```
m_pAnimationComponent->SetTagWithId(m_rotateTagId, isTurning);
```

این خط کد تغییرات انیمیشن را با توجه به وجود آستانه تغییر با `true` یا `false` شدن متغیر `isTurning` ثبت می کند و تگ مربوط به انیمیشن را با `id` مربوط به آن انیمیشن در نظر می گیرد

```
if (isTurning)
{
```

```
const float turnDuration = ۱٫۰f;
```

میزان تغییر انیمیشن با ثابت `turnDuration` برابر یک ثانیه در نظر گرفته می شود

```
m_pAnimationComponent-
>SetMotionParameter(eMotionParamID_TurnAngle,
m_horizontalAngularVelocity * turnDuration);
```

در این خط کد برای انجام دادن عمل حرکت انیمیشنی باید تابع `SetMotionParameter` را به `_____` ثابت `eMotionParamID_TurnAngle` در نظر بگیریم و میزان تغییرات انیمیشن را در متغیر `m_horizontalAngularVelocity` ضرب نمایم

```
}
```

```
const auto& desiredFragmentId =
m_pCharacterController->IsWalking() ?
m_walkFragmentId : m_idleFragmentId;
```

حالا نوبت به آن میرسد که تعیین کنیم که آیا پلیر در حالت قدم زدن `IsWalking` است یا نه؟ اگر در حالت قدم زدن باشد باید این روند ادامه داشته باشد تا زمانی که کلید `W` رها شود و در صورتی که کلید `W` رها شد باید حالت بعدی انیمیشن معطلی `Idle` پخش شود

```
if (m_activeFragmentId != desiredFragmentId)
{
m_activeFragmentId = desiredFragmentId;
m_pAnimationComponent-
>QueueFragmentWithId(m_activeFragmentId);
}
```

حالا با توجه به بلوک `if` باید تعیین شود که آیا انیمیشن در نظر گرفته در صف انیمیشن های موجود همان انیمیشن جاری یا فعلی در حالت پخش است یا نه؟
 (اگر مخالف `m_activeFragmentId != desiredFragmentId`) باشند انیمیشن در نظر گرفته شده همان انیمیشن فعلی خواهد شد
 (اگر `m_activeFragmentId = desiredFragmentId`) و سپس انیمیشن فعلی در صف کلیپ های انیمیشنی قرار میگیرد و بوسیله کامبونت انیمیشن به وسیله تابع `QueueFragmentWithId` پخش خواهد شد.

```
Ang3 ypr =
```

```
CCamera::CreateAnglesYPR(Matrix33(m_lookOrientation));
```

حالا وقتی که چرخش انجام می شود این چرخش باید بر روی پلیر تاثیر بگذارد و این عمل با نمایش انیمیشن `idle` یا `walk` ادامه داشته باشد و این تاثیر بر اساس محور `X` انجام می شود

```
ypr.y = 0;
```

```
ypr.z = 0;
```

محورهای مانند `y` و `z` همیشه صفر خواهند بود

```
const Quat correctedOrientation =  
Quat(CCamera::CreateOrientationYPR(ypr));
```

تمام تبدیلات انجام شده دوباره باید برای دوربین پلیر ثبت شود

```
GetEntity()->SetPosRotScale(GetEntity()-
>GetWorldPos(), correctedOrientation, Vec3(۱,
۱, ۱));
```

همچنین در پایان باید مقادیر مربوط به موقعیت، چرخش و مقیاس با استفاده از تابع `SetPosRotScale` برای پلیر با تابع `GetWorldPos` برای موقعیت، با مقدار متغیر `correctedOrientation` برای چرخش و با بردار یکانی واحد `(۱, ۱, ۱)` برای مقیاس محاسبه و در نظر گرفته شود

```
}
```

تابع `AttachOneObjectToPlayer` همانطور که از اسم آن پیدا است، برای اضافه کردن شی به پلیر استفاده می شود

```
void
CPlayerComponent::AttachOneObjectToPlayer()
{
```

```
IEntity* pChild = gEnv->pEntitySystem-
>FindEntityByName("W۱");
```

ابتدا در داخل مرحله باید یک `entity` با نام `W۱` وجود داشته باشد تا عمل پیدا کردن با تابع `FindEntityByName` انجام شود و در متغیر اشاره گری `pChild` قرار گیرد.

```
pChild->SetPos(Vec3(۰, ۲, ۱));
```


سپس تعیین می کنیم که شی W_1 براساس پلیر شی در چه موقعیتی از پلیر در فضای سه بعدی طبق تابع `SetPos` قرار گیرد، در این خط کد $x=0$ و $y=2$ و $z=1$ است

```
pChild->SetScale(Vec3(0.5, 0.5, 0.5));
```

اندازه شی W_1 را با توجه به تابع `SetScale` مقیاس گذاری می کنیم و به صورت یک ابعاد مربعی با $x=0.5$ و $y=0.5$ و $z=0.5$ در نظر می گیریم

```
m_pCameraComponent->GetEntity()-  
>AttachChild(pChild);
```

و در نهایت به وسیله تابع `AttachChild` از شی جاری `GetEntity` آرگومان را با نام متغیر `pChild` اختصاص می دهیم و شی W_1 به پلیر الصاق یا همان وصل خواهد شد و به پلیر اضافه می شود

```
}
```

تابع `UpdateCamera` برای به روزرسانی چرخش دوربین و نحوه چرخش دوربین براساس داده های ورودی ماوس است و با برش های زمانی براساس پارامتر `frameTime` درارتباط است

```
void CPlayerComponent::UpdateCamera(float  
frameTime)  
{
```

```
AttachOneObjectToPlayer();
```

تابع `AttachOneObjectToPlayer` همانطور که در صفحات قبل توضیح داده شد برای اضافه کردن شی به پلیر است

```
if (gEnv->IsEditor())
return;
```

بلوک `if` مربوطه برای جلوگیری از اجرای خطوط کد دیگر است در زمانی که بازی در سندباکس (`gEnv->IsEditor`) اجرا می شود.

```
Ang3 ypr =
CCamera::CreateAnglesYPR(Matrix33(m_lookOrienta
tion));
```

طبق آنچه که در آخرین داده های ورودی از دستگاه ماوس حاصل می شود، ایجاد زاویه درست برای دوربین و ماتریکس سه بعدی برای عمل چرخش دوربین محاسبه شود و سپس انتقال این ماتریکس در ادامه خطوط کد انجام شود.

```
ypr.x += m_mouseDeltaRotation.x *
m_rotationSpeed;
```

```
const float rotationLimitsMinPitch = -۰.۸۴f;
const float rotationLimitsMaxPitch = ۱.۵f;
```

```
ypr.y =CLAMP(ypr.y + m_mouseDeltaRotation.y *
m_rotationSpeed, rotationLimitsMinPitch,
rotationLimitsMaxPitch);
```

```
ypr.z = ۰;

m_lookOrientation =
Quat(CCamera::CreateOrientationYPR(ypr));

m_mouseDeltaRotation = ZERO;

ypr.x = ۰;
```

قبلا در رابطه با این خطوط کدها، توضیحات کافی ارائه داده ام و از توضیحات مجدد آن پرهیز می نمایم.

```
Matrix۳۴ localTransform = IDENTITY;
```

ماتریکس همانی ۴×۳ تشکیل می شود تا انتقال ماتریکس برای دوربین با درایه های پیش فرض یک واحدی انجام شود

```
localTransform.SetRotation۳۳(CCamera::CreateOrientationYPR(ypr));
```

در نهایت عمل چرخش دوربین با متغیر `ypr` انجام می شود و با تابع `SetRotation۳۳` انجام خواهد شد.

برای درک کامل ماتریکس ها لازم است، کاربرد ماتریکس ها در ساخت بازی های ویدئویی را مطالعه کنید

```
if (ICharacterInstance *pCharacter =
m_pAnimationComponent->GetCharacter())
{
```

طبق متغیر pCharacter تمامی انیمیشن های پلیر در این متغیر قرار میگیرد تا به سیستم اسکلت بندی پلیر نیز دسترسی داشته باشید

```
const QuatT &cameraOrientation = pCharacter-
>GetISkeletonPose()-
>GetAbsJointByID(m_cameraJointId);
```

طبق این خط کد، دوربین باید در سر پلیر تعبیه شود و نقطه اتصال دوربین براساس جهت درست پلیر در زمان ساخت پلیر در نرم افزارهای سه بعدی لحاظ می شود و نقطه اتصال با توجه به GetAbsJointByID جستجو و مدنظر قرار گرفته می شود

```
localTransform.SetTranslation(cameraOrientatio
n.t + Vec3(0, viewOffsetForward,
viewOffsetUp));
```

نسبت فاصله مدل سه بعدی پلیر از دوربین را فضایی با نام آفست تشکیل می دهد، این فضا باید با تابع SetTranslation برای دوربین انقال یابد و مختصات صفر دوربین cameraOrientation.t با یک بردار سه بعدی جمع می شود که پارامتر viewOffsetForward و y= و z= viewOffsetUp است و x=۰ است.

```
}
```

```
m_pCameraComponent-
>SetTransformMatrix(localTransform);
```

در نهایت فضای آفستی دوربین باید به متغیر کامبوتی دوربین با نام `m_pCameraComponent` بوسیله تابع `ماتریکس انتقال` `SetTransformMatrix` ثبت شود

```
m_pCameraComponent->GetCamera().GetViewdir();
```

ابتدا دوربین جاری را با استفاده از تابع `GetCamera` دریافت کرده و با استفاده از تابع `GetViewdir` جهت دوربین پلیر رو به عمق (رو به جلو) محاسبه می شود و در متغیر برداری `ahmadval` قرار می گیرد.

```
}
```

تابع `RayCast` جزء تابع اصلی بازی است که بیشترین کار را برای پلیر انجام می دهد، این تابع برای هدف گیری، نشانه گذاری بمب `C۴`، پرتاب نارنجک و ... استفاده می شود، در واقع بیشتر توابع و کلاس های ایجاد شده و استفاده شده در این کتاب در تابع `RayCast` به کار گیری می شوند

```
void CPlayerComponent::RayCast()
{
```

```
Vec3 pos = GetEntity()->GetWorldPos() + Vec3(۰,
۰, ۲);
```

متغیر برداری `pos` برای گرفتن مختصات جهانی با آفست ارتفاع `z=۲` در نظر گرفته شده است.

```
Vec3 dir = Vec3(ahmadval.x*۱۰۰۰۰,
ahmadval.y*۱۰۰۰۰, ahmadval.z * ۱۰۰۰۰);
```

متغیر برداری `dir` نیز برای محاسبه برد هدف گیری تا ۱۰ کیلومتر با استفاده از متغیر برداری `ahmadval` به کارگیری شده است

```
int ObjType= ent_all;
```

متغیر `ObjType` محلی است و همه اشیاء در بازی را با اشعه برخوردی آدرس دهی می کند، این اشیاء می توانند شامل

<code>ent_static</code>		
<code>ent_sleeping_rigid</code>		<code>ent_rigid</code>
<code>ent_living</code>		<code>ent_independent</code>

`ent_terrain` باشند

```
const unsigned int rayFlags =  
rwi_stop_at_pierceable | rwi_colltype_any;
```

متغیر `rayFlags` محلی و ثابت است و پرچم های اشعه برخوردی را فعال می کند

```
gEnv->pPhysicalWorld->  
RayWorldIntersection(pos, ahmadval, ObjType,  
rayFlags, &hit, ۱);
```

تابع `RayWorldIntersection` همان تابع اشعه برخوردی (`Raycast`) است که با توجه به پارامترهای زیر، نقاط برخورد و نوع شی یا همان `entity` را برای ما در متغیر `hit` می ریزد :

`pos` : با توجه به متغیر تعریف شده در بالا، موقعیت پلیر با ارتفاع ۲ متر، شروع نقطه برخوردی است

ahmadval : جهت رو به جلو یا همان عمقی است که دوربین پلیمر به آن نگاه می کند و انتهای نقطه برخوردی است

ObjectType : با توجه به تعریف متغیر بالا، نوع شناسایی entity ها در اثر اشعه برخوردی حاصل می شود

rayFlags : با توجه به تعریف متغیر بالا، نوع پرچم به کار رفته را در اشعه برخوردی محاسبه می کند

hit : مجموعه اطلاعاتی که هنگام اشعه برخوردی حاصل می شود و در داخل متغیر hit ذخیره می شود، مانند مسافت پلیمر از نقطه برخوردی، نوع entity برخوردی چیست، نقطه برخوردی در فضای سه بعدی، بردار سه بعدی نرمال حاصل شده از نقطه برخوردی و...

۱ : عدد یک همیشه ثابت است و نباید تغییر داده شود، حاصل عدد یک، حاصل متراژ نقطه برخوردی تا آخرین نقطه از دوردست ها را شامل می شود

```
pHitEntity = hit.pCollider;
```

متغیر pHitEntity دارای محتوایی است که منشاء آن اشعه برخوردی است که آن اشعه با استفاده از متغیر hit، شی ای که فیزیک collider و rigidbody را دارند در خود حمل می کند

```
pEntity = gEnv->pEntitySystem-  
>GetEntityFromPhysics(pHitEntity);
```

با این خط کد با استفاده از تابع GetEntityFromPhysics، شی ای را دریافت می کند که حتما باید فیزیک collider و rigidbody را داشته باشد، سپس شی در داخل متغیر pEntity قرار می گیرد

چند صفحه قبل مباحث Raycast توضیح داده شد اما مرور مجدد و توضیحات آن ضروری بود، زیرا که درک تابع RayCast در کرای انجین بسیار مهم و حیاتی است

```
if(pEntity)
{
    اگر شی pEntity دارای فیزیک کامپوننت collider و
    rigidbody باشد، بلوک if و دستورالعمل های زیر اجرا خواهند شد
    string sss= pEntity->GetName();
```

با توجه به تعریف متغیر رشته ای sss، اسم شی pEntity را ذخیره می کند

```
if (hit.dist < ۲,۵f && hit.dist >= ۰,۳f &&
GKeyAmmo>=۱ && strcmp(sss,"Player")!=۰)
{
```

بلوک if بعدی بررسی می شود و دارای ۴ شروط مهم است که آن شروط را بررسی میکنم :

hit.dist < ۲,۵f : اگر پلیر نسبت به یک شی یا اشیاء با توجه به متغیر ObjType در فاصله کمتر از ۲,۵ متر قرار گیرد
hit.dist >= ۰,۳f : اگر پلیر نسبت به یک شی یا اشیاء با توجه به متغیر ObjType در فاصله بزرگتر از ۰,۳ متر قرار گیرد
GKeyAmmo>=۱ : اگر خشاب بمب C۴ بزرگتر یا مساوی یک قرار گیرد
strcmp(sss,"Player")!=۰ : اگر اشعه افتاده شده با توجه به متغیر ObjType خود فیزیک پلیر نباشد و مخالف پلیر باشد

آنگاه براساس این شروط چهارگانه که همگی درست باشند، خطوط کد زیر اجرا می شوند

```
IPersistentDebug *iPD = gEnv->pGameFramework-
>GetIPersistentDebug();
```

ابتدا باید یک یک متغیر اشاره گر را با نام `iPD` تعریف کنم و فضای از حافظه RAM را برای آن اختصاص دهم و این فضا نهایتاً در صفحه نمایش بازی نمایش داده شود

```
iPD->Begin("EnshaAllah", false);
```

متغیر `iPD` که در بالا برای اختصاص فضایی برای رسم اطلاعات بر روی صفحه نمایش تعریف شده است، حالا با استفاده از تابع `Begin` شروع به کار می کند، رشته ای دلخواه به آن اختصاص دهید و مقدار `false` را به آن اختصاص دهید

```
iPD->AddText3D(hit.pt, ۲,۱۵f, ColorF(۰, ۱, ۱),
۰,۰۵f, "Attach C۴ Boom here" /*+sss*/);
```

تابع `AddText3D`، متنی سه بعدی را رسم می کند که در مختصات بردار سه بعدی اشعه برخوردی `hit.pt` قرار می گیرد که اندازه فونت متن ۲,۱۵ واحد و رنگ متن به صورت آبی کم رنگ و رشته متنی `Attach C۴ Boom here` نمایش داده می شود



```

}
}
else if (hit.dist < ۲.۵f && hit.dist >= ۰.۳f &&
GKeyAmmo >= ۱ )
{
    IPersistentDebug *iPD = gEnv->pGameFramework-
    >GetIPersistentDebug();
    iPD->Begin("EnshaAllah", false);
    iPD->AddText3D(hit.pt, ۲.۱۵f, ColorF(۰, ۱, ۱),
    ۰.۰۵f, "Attach C Boom here" /*+sss*/);
}

```



حالا توضیحات بلوک `if` های زیر را براساس توضیحات داده شده قبلی را در نظر بگیرید، درک این بلوک `if` ها بسیار ساده است

```
if (hit.dist<=۰.۵f && hit.dist>=۲.۵)
{
    IPersistentDebug *iPD=gEnv->pGameFramework-
    >GetIPersistentDebug();
    iPD->Begin("EnshaAllah",false);
    iPD->AddSphere(hit.pt, ۰.۱f, ColorF(۱, ۰, ۰),
    ۰.۰۵f);
}
```

فقط یک توضیح کوتاه در رابطه با تابع `AddSphere`: این تابع یک کره را رسم می کند و نقطه مختصات برداری رسم کره در متغیر با محتوای `hit.pt` بر اساس اشعه برخوردی خواهد بود و کره با شعاع `۰.۱` متر و رنگ قرمز و `۰.۰۵` ثانیه زمان رسم خواهد داشت و سپس بعد از `۰.۰۵` ثانیه کره سه بعدی از بین خواهد رفت، این کره به صورت یک نقطه قرمز رنگ نمایش داده می شود

```
iPD->AddText3D(hit.pt, ۲,۱۵f, ColorF(۱, ۰, ۰),
۰,۰۲f, "Distance : " +ToString((int)
hit.dist)+" meter");
```

در این خط کد نیز فاصله بین پلیر و اشعه برخوردی با واحد متر نشان داده می شود و رنگ متن distance: / meter با اندازه فونت ۲,۱۵ واحد با زمان ۰,۰۲ ثانیه قرمز خواهد بود،بقیه خطوط کدها نیز قابل حدس و ساده هستند

```
}
```



```
if (hit.dist <=۱۰.f && hit.dist >۵.f)
{
IPersistentDebug *iPD = gEnv->pGameFramework-
>GetIPersistentDebug();
iPD->Begin("EnshaAllah", false);
iPD->AddSphere(hit.pt, ۰,۱۵f, ColorF(۰, ۱, ۰),
۰,۰۵f);
iPD->AddText3D(hit.pt, ۲,۱۵f, ColorF(۰, ۱, ۰),
۰,۰۲f, "Distance : " + ToString((int)hit.dist)
+ " meter");
}
```



حالا می بینید که کره سبز رنگ به صورت یک نقطه سبز رنگ نمایش داده شده و متن distance: / meter نیز به رنگ سبز است

```
if (hit.dist < ۱۵.f && hit.dist > ۱۰.f)
{
    IPersistentDebug *iPD = gEnv->pGameFramework-
    >GetIPersistentDebug();
    iPD->Begin("EnshaAllah", false);
    iPD->AddSphere(hit.pt, ۰.۲f, ColorF(۰, ۰, ۱),
    ۰.۱f);
    iPD->AddText3D(hit.pt, ۲.۱۵f, ColorF(۰, ۰, ۱),
    ۰.۰۲f, "Distance : " + ToString((int)hit.dist)
    + " meter");
}
```



حالا می بینید که کره آبی رنگ به صورت یک نقطه آبی رنگ نمایش داده شده و متن distance: / meter نیز به رنگ آبی است

```

if (hit.dist >= ۱۵.f && hit.dist < ۶۰.f)
{
    IPersistantDebug *iPD = gEnv->pGameFramework-
    >GetIPersistantDebug();
    iPD->Begin("EnshaAllah۳", false);
    iPD->AddSphere(hit.pt, ۰.۲۵f, ColorF(۰, ۰, ۰),
    ۰.۱f);
    iPD->AddTextrD(hit.pt, ۲.۱۵f, ColorF(۰, ۰, ۰),
    ۰.۰۲f, "Distance : " + ToString((int)hit.dist)
    + " meter");
}

```



حالا می بینید که کره سیاه رنگ به صورت یک نقطه سیاه رنگ نمایش داده شده و متن distance: / meter نیز به رنگ سیاه است

```

if (hit.dist >= ۶۰.f)
{
    IPersistantDebug *iPD = gEnv->pGameFramework-
    >GetIPersistantDebug();
    iPD->Begin("EnshaAllah۶", false);
    iPD->AddSphere(hit.pt, ۲.۲۵f, ColorF(۰, ۰, ۰),
    ۰.۱f);
}

```

```

iPD->AddTextRD(hit.pt, ۲,۱۵f, ColorF(۱, ۱, ۰),
۰,۰۲f, "Distance : " + ToString((int)hit.dist)
+ " meter");

}

}

```



حالا می بینید که کره زرد رنگ به صورت یک نقطه زرد رنگ نمایش داده شده و متن distance: / meter نیز به رنگ زرد است

```

void CPlayerComponent::GetDirCamera()
{
ahmadval= m_pCameraComponent-
>GetCamera().GetViewdir()*۱۰۰۰۰;

```

تابع GetDirCamera جهت دوربین را براساس برد ۱۰ کیلومتر بدست می آورد و به داخل متغیر ahmadval می ریزد

```

}

```

```

void CPlayerComponent::Revive()
{

```

```

SpawnAtSpawnPoint();

```

تابع `SpawnAtSpawnPoint` وظیفه اش تعیین نقطه شروع بازی برای پلیر طبق وجود کلاس `CSpawnPointComponent` است

```
GetEntity()->Hide(false);
```

با استفاده از تابع `Hide` مشخص می کنیم که پلیر مخفی شود یا نه؟ از آنجایی که آرگومان این تابع با مقدار `false` در نظر گرفته شده است، پلیر مخفی نخواهد شد

```
GetEntity()-  
>SetWorldTM(Matrix3f::Create(Vec3(۱, ۱, ۱),  
IDENTITY, GetEntity()->GetWorldPos()));
```

طبق ماتریکس های انتقال مکان در واحد های همانی، مکان واقعی پلیر در فضای سه بعدی برای ایجاد شدن پلیر در آن مکان محاسبه و اجرا خواهد شد

```
m_pAnimationComponent->ResetCharacter();
```

مقادیر پخش شدن انیمیشن ها در صف مانکن ادیتور صفر شده و مجددا آماده پخش شدن انیمیشن ها برای پلیر خواهد بود

```
m_pCharacterController->Physicalize();
```

فیزیک جاذبه و حرکت به کامبونت `CharacterController` ریست یعنی صفر شده و مجددا محاسبه، اجرا و به کارگیری خواهد شد

```
m_inputFlags = ۰;
```


مقدار صفر برای پرچم ورودی دستگاه های صفحه کلید و ماوس برای پلیر در نظر گرفته می شود.

```
m_mouseDeltaRotation = ZERO;
```

برش های جزئی چرخش ماوس برای دوربین پلیر صفر خواهد شد

```
m_lookOrientation = IDENTITY;
```

جهت نگاه پلیر و جهت دید پلیر مقدار اولیه خواهد شد

```
m_mouseDeltaSmoothingFilter.Reset();
```

برش های جزئی ماوس برای حرکت نرم و صاف دوربین پلیر با تابع Reset مقدار اولیه خواهد شد

```
m_activeFragmentId = FRAGMENT_ID_INVALID;
```

تگ های کلیپ انیمیشنی برای پلیر نیز با مقدار نامعتبر مقداردهی می شود تا در هنگام استفاده از مانکن ادیتور مقدار درست به پلیر داده شود و کلیپ انیمیشنی براساس صفحه کلید تگ گذاری و اجرا شود

```
m_horizontalAngularVelocity = ۰,۰f;
```

شتاب زوایه افقی ماوس برای چرخش صفر می شود

```
m_averagedHorizontalAngularVelocity.Reset();
```

میانگین شتاب زوایه افقی ماوس برای چرخش با تابع Reset مقداردهی اولیه می شود

```
if (ICharacterInstance *pCharacter =  
m_pAnimationComponent->GetCharacter())  
{
```

با استفاده از دستور `if` جاری، کاراکتر مربوطه را برای دریافت انیمیشن‌ها گرفته و براساس ساختمان `bone` کاراکتر نقطه یا `join` مربوط به سر یا همان `head` را پیدا می‌کنیم و به عنوان نقطه چشم پلیر درنظر می‌گیریم تا دوربین در آن نقطه تعبیه شود

```
m_cameraJointId = pCharacter-
>GetIDefaultSkeleton().GetJointIDByName("head"
);
}
}
```

تابع `SpawnAtSpawnPoint` نقطه شروع بازی را برای پلیر پیدا می‌کند

```
void CPlayerComponent::SpawnAtSpawnPoint()
{

if (gEnv->IsEditor())
return;
```

این خط کد در بلوک `if` نمیگذارد دیگر کدها در پایین اجرا شود وقتی که در سندباکس هستیم

```
auto *pEntityIterator = gEnv->pEntitySystem-
>GetEntityIterator();
```

با استفاده از تابع `GetEntityIterator` به لیست تمامی `entity` ها در مرحله جاری از بازی دسترسی خواهیم داشت

```
pEntityIterator->MoveFirst();
```

اولین entity که در مرحله جاری وجود دارد، با استفاده از MoveFirst آدرس دهی و با استفاده از متغیر pEntityIterator اشاره گر بر روی آن قرار می گیرد

```
while (!pEntityIterator->IsEnd())
{
```

با استفاده از حلقه while بررسی می کنیم که تا زمانی که تمام entity های مرحله را پیمایش نکرده ایم، این حلقه اجرا شود

```
IEntity *pEntity = pEntityIterator->Next();
```

با استفاده از تابع Next در مرحله جاری به entity بعدی دسترسی خواهیم داشت و با استفاده از متغیر pEntity اشاره گر به آن اشاره خواهد کرد

```
if (auto* pSpawner = pEntity-
>GetComponent<CSpawnPointComponent>())
{
```

در بلوک if جاری یک متغیر با نام pSpawner ایجاد می شود که خالی است و با استفاده از تابع ژنریک GetComponent یک کلاس CSpawnPointComponent را دریافت می کند و در متغیر اشاره گری pSpawner می ریزد

```
pSpawner->SpawnEntity(m_pEntity);
```

با استفاده از تابع `SpawnEntity`، مکان (نقطه) شروع بازی (`SpawnPoint`) برای پلیر پیدا می شود و پلیر در آن مکان قرار میگیرد

```
break;
}
}
}
```

با استفاده از تابع `HandleInputFlagChange` رفتار کلیدهای ماوس و صفحه کلید را در گرفتن و یا رها کردن، مشخص می کنیم و پرچم های این کلیدها `set` یا `reset` یا معکوس خواهد شد

```
void
CPlayerComponent::HandleInputFlagChange(TInput
Flags flags, int activationMode,
EInputFlagType type)
{
    با استفاده از typedef و enum تعریف شده در TInputFlags و
    EInputFlagType درون ساختار switch می توان مشخص کرد
    که آیا کلید های صفحه کلید یا کلیک های ماوس Hold شده است یا
    Toggle است؟
}
```

```
switch (type)
{
    case EInputFlagType::Hold:
    {
        if (activationMode == eIS_Released)
        {
```

با استفاده از بلوک `if` جاری وضعیت پرچم کلیدها یا کلیک ها `set` یا معکوس می شود، زمانی که عمل گرفتن کلیدها یا کلیک ها اتفاق افتاده است

```
m_inputFlags &= ~flags;
```

```
}
```

```
else
```

```
{
```

```
m_inputFlags |= flags;
```

```
}
```

```
}
```

```
break;
```

```
case EInputFlagType::Toggle:
```

```
{
```

```
if (activationMode == eIS_Released)
```

```
{
```

با استفاده از بلوک `if` جاری وضعیت پرچم کلیدها یا کلیک ها `reset` می شود، زمانی که عمل رها کردن کلیدها یا کلیک ها اتفاق افتاده است

```
m_inputFlags ^= flags;
```

```
}
```

```
}
```

```
break;
```

```
}
```

```
}
```

با درک کامل این فصل، Template های دیگری که با زبان C++ در کرای انجین ۵,۴ تهیه شده، می توانید به خوبی درک کنید و این Template ها در مسیر زیر وجود دارند :

CRYENGINE_۵,۴\Templates\cpp

فصل نهم

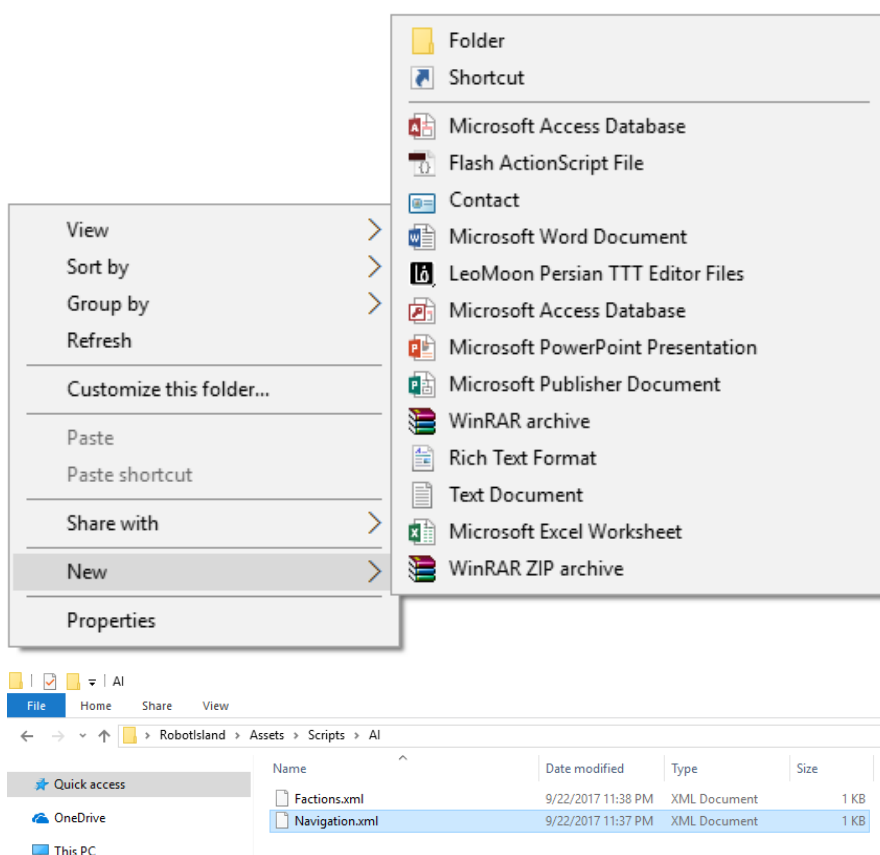
هوش مصنوعی

یکی از مهمترین موضوع در ساخت بازی های ویدئویی هوش مصنوعی است، این موضوع هر چه بهتر و با جزئیات بیشتر پیاده شود، کیفیت بازی بر اساس ساخت دشمنان مختلف، گیم‌های بیشتری را به خود جذب خواهد کرد، در این فصل به بررسی هوش مصنوعی در کرای انجین با زبان C++ می پردازم، اگر مطالعات کافی را بر روی الگوریتم های هوش مصنوعی انجام داده باشید، اولین مورد از هوش مصنوعی به پیدا کردن کوتاه ترین مسیر به هدف با الگوریتم های A* و Floyd می رسید، اگرچه شما می توانید این الگوریتم ها را در کرای انجین پیاده کنید و سیستم هوش مصنوعی خود را با زبان قدرتمند C++ پیاده کنید اما کرای انجین نیز دارای سیستم هوش مصنوعی بهینه سازی شده است و شما می توانید از ابزارهای هوش مصنوعی کرای انجین استفاده کنید، اولین ابزار هوش مصنوعی در کرای انجین Navigation Area است که در این فصل به آن خواهیم پرداخت، البته مباحث دیگر مانند Behavior Tree , Cover Points و غیره وجود دارند که به صورت Component Entity System برای C++ در کرای انجین ارائه نشده است و شما باید از پروژه GameSDK استفاده کنید و در حال حاضر تیم کرای انجین، مشغول جداسازی کدهای C++ ماژول های CryAction و GameSDK است و انتظار می رود که در نسخه کرای انجین ۵.۵ این جداسازی و ایجاد کامبونت های Cover Points ، Behavior Tree و دیگر کامبونت ها اضافه شوند

در تصویر زیر می بینید که فایل Navigation.xml در پروژه جزیره ربات ها با توجه به مسیر زیر به چشم می خورد:

RobotIslands\Assets\Scripts\AI

شما باید این فایل را درست در این مسیر ایجاد کنید و برای استفاده از هوش مصنوعی در پروژه دیگر پوشه های تودرتو را طبق مسیر داده شده ایجاد کنید و سپس در یک جای خالی در داخل پوشه AI راست کلیک کنید و گزینه New را انتخاب کنید و سپس برروی گزینه Text Document کلیک کنید تا فایلی متنی با پسوند *.txt* ایجاد شود، پسوند فایل را به xml و نام فایل را به Navigation تغییر دهید



حالا دستورات زیر را داخل فایل Navigation.xml تایپ کنید و سپس محتوای فایل را ذخیره کنید، بعدا متوجه خواهید شد که این دستورات در

سندباکس لود خواهد شد و باعث ایجاد شدن دو گزینه جدید در Navigation Area Entity خواهد شد

```
<Navigation version="۶" >
<AgentTypes>

<AgentType name="MediumSizedCharacters"
voxelSize="۰.۱۲۵, ۰.۱۲۵, ۰.۱۲۵" radius="۴"
height="۱۶" climbableHeight="۳"
maxWaterDepth="۸" >
<SmartObjectUserClasses>
<class name="Human" />
</SmartObjectUserClasses>
</AgentType>

<AgentType name="VehicleMedium"
voxelSize="۰.۲۰, ۰.۲۰, ۰.۲۰" radius="۱۴"
height="۱۰" climbableHeight="۴"
maxWaterDepth="۵" >
</AgentType>

</AgentTypes>
</Navigation>
```

فایل xml موجود براساس Navigation در نسخه ۶ کرای انجین تهیه شده که دو نوع عامل را تعریف می کند که برای دشمنان تان در بازی استفاده شوند، یک نوع عامل با نام MediumSizedCharacters وجود دارد که برای دشمنان به صورت انسان یا حیوان ارائه می شود و یک نوع عامل

دیگر با نام VehicleMedium که برای وسایل نقلیه دشمنان استفاده می شود

در تصویر زیر یک مثلث بنفش رنگ وجود دارد که به داخل Terrain بازی فرو رفته است، این مثلث در واقع محدوده حرکت دشمن را شامل می شود و دشمن تنها در داخل این مثلث می تواند حرکت کند و دشمن نمی تواند از این مثلث خارج شود، برای ایجاد یک ناحیه Navigation Area ابتدا به پنجره Create Object به بخش AI مراجعه کنید (به کادر زرد رنگ شماره ۱ دقت کنید) و سپس دکمه Navigation Area را انتخاب کنید (به کادر زرد رنگ شماره ۲ نگاه کنید) و محدوده ای را بر روی terrain برای پیمایش دشمن مانند آنچه که در تصویر می بینید با هر شکلی (مربع، مثلث، مستطیل و ... به رنگ بنفش است) رسم کنید و همانطور که می بینید خصوصیات Navigation Area در پنجره properties نمایش داده شده و طبق تصویر خصوصیات Height=۱۶ را در نظر بگیرید و در بخش navigation (کادر زرد رنگ شماره ۳)، دو گزینه با نام های زیر وجود دارند :

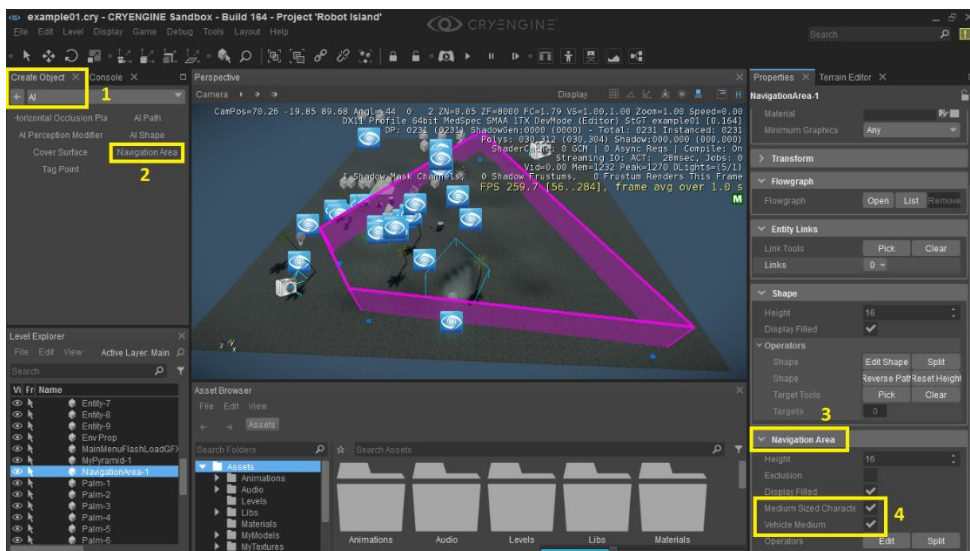
Medium Sized Characters : این گزینه برای پیاده سازی هوش مصنوعی حیوانات و انسان ها به کار برده می شود

Vehicle Medium : این گزینه برای پیاده سازی هوش مصنوعی ماشین ها، تانک ها، نفربرها، زره پوش ها، قایق و کشتی، زیردریایی به کار برده می شود

این دو گزینه در ارتباط با فایل Navigation.xml است که آن را ایجاد کردیم و محتوای آن را با تگ های مختلف در مسیر گفته شده ذخیره کردیم، دیگر تگ ها نیز در رابطه با ارتفاع و عمق Navigation است و حتما لازم است که این فایل xml در مسیر مورد نظر وجود داشته باشد (به کادر زرد رنگ شماره ۴ نگاه کنید)

RobotIslands\Assets\Scripts\AI

نکته بسیار مهم: برای لود شدن مپ (مرحله) در داخل خروجی بازی (هنگام play کردن بعد از build پروژه) در ویژوال استودیو، لازم است که قبل از build و play کردن بازی در ویژوال استودیو، به داخل ادیتور کرای انجین رفته، سپس به منوی فایل مراجعه کرده و گزینه Export to Engine را انتخاب کنید یا دکمه F7 را بر روی صفحه کلید بزنید، با این کار، کل مرحله برای ویژوال استودیو قابل لود شدن خواهد بود



حالا به کدنویسی یک دشمن ساده می پردازم که اگر کمتر از ۳ متر از پلیمر فاصله داشت، پلیمر و خودش را از بین ببرد، البته پلیمر به هر جایی برود، دشمن نیز او را دنبال خواهد کرد، سرعت دشمن نیز قابل تنظیم است، اجازه دهید که مباحث فصول قبلی را در رابطه با ساخت یک entity یا component جدید تشریح نمایم، این مسئله باعث می شود که مروری بر روی کدهای فصول قبل داشته باشم و کدهای جدید را نیز در این فصل بیان کنم

ابتدا هدر فایل AI_Follower.h را بررسی می‌کنم

```
#pragma once
```

برای جلوگیری از تعریف مجدد کلاس‌ها و برای جلوگیری از تعریف مجدد هدر فایل‌ها با پیش پردازنده include استفاده می‌شود، توصیه می‌شود برای ایجاد entity یا component در کرای انجین از این دستور استفاده کنید و فقط یکبار تعریف هدر فایل و فقط یکبار تعریف کلاس با این دستور انجام می‌شود

```
#include <CryEntitySystem\IEntitySystem.h>
```

برای ساخت entity یا component جدید باید از این هدر فایل استفاده کنیم

```
#include  
<DefaultComponents/AI/PathfindingComponent.h>
```

این هدر فایل برای ایجاد یک کامپوننت از نوع path finder در دشمن خواهد شد که باعث می‌شود، پلیمر را تعقیب کند

```
#include  
<DefaultComponents/Physics/CharacterController  
Component.h>
```

این هدر فایل برای ایجاد یک کامپونت Character Controller در دشمن خواهد شد، این کامپونت برای ساخت فیزیک حرکت دشمن استفاده می شود

```
class CAIFollowerComponent final : public
IEntityComponent
{
```

یک کلاس نهایی را می سازیم که از رابط IEntityComponent ارث بری کرده باشد

```
public:
virtual ~CAIFollowerComponent() {}
```

اگرچه در این کلاس، سازنده تعریف نشده است و به صورت اتوماتیک ایجاد می شود اما تخریب کننده کلاس با علامت ~ در کنار نام کلاس تعریف شده است اما هیچ کاری را انجام نمی دهد زیرا که بین دو علامت {} هیچ دستورالعملی وجود ندارد

این نکته جالب را بدانید که سازنده کلاس، تخصیص حافظه برای شی و تولید شی (entity یا component) را انجام می دهد و تخریب کننده کلاس، از بین بردن شی و پاک سازی حافظه را انجام می دهند

```
virtual void Initialize() override
{
```

تابع سیستمی Initialize یکبار اجرا می شود و برای ایجاد شی (entity یا component) در سندباکس (به صورت Drag & Drop) لازم است از این رویداد استفاده شود

```
const int geometrySlot = ۰;
```

یک ثابت با نام `geometrySlot` با مقدار صفر تعریف می کنیم و این ثابت برای استفاده از `slot۰` کاربرد دارد

```
m_pEntity->LoadGeometry(geometrySlot,
"Objects/Default/primitive_sphere.cgf");
```

با توجه به شی جاری (`m_pEntity` یا `GetEntity`) در داخل کلاس با تابع `LoadGeometry`، یک مدل هندسی سه بعدی لود می کنیم، آدرس مدل سه بعدی در مسیر `Assets Browser` است

```
auto *pAIFollowerMaterial = gEnv->p3DEngine-
>GetMaterialManager()-
>LoadMaterial("Materials/bullet");
```

یک متغیر اشاره گر (*) با نام `pAIFollowerMaterial` تعریف می کنیم و داخل این متغیر باید یک متریال لود شود که با تابع `LoadMaterial` این عمل را انجام می دهیم، آدرس متریال در مسیر `Assets Browser` است

```
m_pEntity->SetMaterial(pAIFollowerMaterial);
```

حالا با توجه به تابع `SetMaterial`، متغیر متریالی را به شی جاری نسبت می دهیم تا بر روی مدل سه بعدی طبق متغیر متریالی `pAIFollowerMaterial`، متریال مورد نظر نگاشت و کشیده شود

```
GetEntity()->SetViewDistRatio(۲۵۵);
```

شی جاری تا فاصله ۲۵۵ متری طبق تابع `SetViewDistRatio` دیده می شود و اگر فاصله بیشتر شود، شی جاری دیده نخواهد شد

```
m_pPathfindingComponent = m_pEntity-
>GetOrCreateComponent<Cry::DefaultComponents::
CPathfindingComponent>();
```

حالا باتوجه به شی جاری یک کلاس `CPathfindingComponent` را طبق تابع ژنریک (تابع جنریک نیز گفته می شود) `GetOrCreateComponent` گرفته یا ایجاد می کنیم و به داخل متغیر `m_pPathfindingComponent` می ریزیم

```
m_pCharacterController = m_pEntity-
>GetOrCreateComponent<Cry::DefaultComponents::
CCharacterControllerComponent>();
```

مجددا داخل شی جاری یک کلاس

`CCharacterControllerComponent` را طبق تابع ژنریک `GetOrCreateComponent` گرفته یا ایجاد می کنیم و به داخل متغیر `m_pCharacterController` می ریزیم.

```
m_pCharacterController-
>SetTransformMatrix(Matrix::Create(Vec3(۱.f),
IDENTITY, Vec3(۰, ۰, ۱.f)));
```

حالا با توجه به ماتریکس انتقال فیزیک بدنه دشمن در متغیر `m_pCharacterController`، یک متر براساس ارتفاع (Z) با بردار `Vec3(۰, ۰, ۱.f)` را لحاظ می کنیم، شما می توانید این بردار را

تغییر دهید اما به شدت توصیه می‌کنم که بقیه مقادیر پیش فرض تابع `SetTransformMatrix` را تغییر ندهید

```
}
```

```
static void
ReflectType(Schematyc::CTypeDesc<CAIFollowerCo
mponent>& desc)
```

```
{
```

تابع استاتیک `ReflectType` مستقل از کلاس عمل می‌کند و باعث ثبت و نمایش کلاس `CAIFollowerComponent` در پنجره `properties` می‌شود، در واقع این تابع عمل انعکاس کدهای کلاس `CAIFollowerComponent` را در ادیتور برای انجین نمایش می‌دهد.

```
desc.SetGUID("{D۹۸۵A۱۴۲-۶B۹۸-۴C۰۲-B۸۵D-
F۳۸۳D۳۲۵D۹۴۸}"_cry_guid);
```

در دفتر ثبت اسناد ویندوز یا همان `Registry Editor` کلید `D۹۸۵A۱۴۲-۶B۹۸-۴C۰۲-B۸۵D-F۳۸۳D۳۲۵D۹۴۸` ایجاد می‌شود، زیرا که هر `entity` یا `component` نیاز به یک کلید منحصر به فرد دارند که در ادیتور برای انجین ثبت شوند که در ویژوال استودیو از طریق منوی `Tools` گزینه `Create GUID` می‌توانید به این کلیدها دسترسی داشته باشید، در فصل ۶ به خوبی این مطلب توضیح داده شده است.

```
desc.SetEditorCategory("Game");
```

وقتی که می خواهیم یک entity یا component در سندباکس نمایش داده شود، یک کاتالوگ ایجاد می کنیم که اسم این کاتالوگ در این خط کد Game است.

```
desc.SetLabel("AIFollower");
```

اسم entity یا component در سندباکس (پنج‌گانه CreateObject->Components) با نام AIFollower نمایش داده می شود.

```
desc.SetDescription("This AI Follower can be  
used to spawn entities");
```

توضیحاتی برای entity یا component ی که ایجاد می شود، در نظر می گیریم، این بخش برای راهنمایی برنامه نویس و کاربران کرای انجین است.

```
desc.SetComponentFlags({  
    IEntityComponent::Eflags::Transform,  
    IEntityComponent::Eflags::Socket,  
    IEntityComponent::Eflags::Attach });
```

پرچم هایی برای ثبت entity یا component ایجاد شده در پنجره properties را باید در نظر بگیریم و این پرچم ها مختلف هستند که در اینجا پرچم Transform برای Rotation ، Position ، Scale استفاده می شوند، اگرچه کرایتک برای پرچم های دیگر توضیحاتی را منتشر نکرده است اما احتمالاً براساس کلمه پرچم مثل Socket باید entity یا component ایجاد شده را برای شبکه سازی بازی در نظر بگیریم و کلمه Attach برای این است که component های دیگر را به component یا entity ایجاد شده بتوانیم اضافه نماییم.

```
}
```

```
virtual uint۶۴ GetEventMask() const override {
return BIT۶۴(ENTITY_EVENT_UPDATE |
BIT۶۴(ENTITY_EVENT_DONE); }
```

تابع سیستمی `GetEventMask` برای ماسک گذاری رویدادها که می تواند حامل پیام های بیتی مختلفی باشد، ثابت های مختلف رویدادی را به تابع سیستمی `ProcessEvent` ارسال می کند، در واقع دو تابع سیستمی `ProcessEvent` و `GetEventMask` مکمل هم دیگر هستند و در اینجا تابع `GetEventMask` حامل ثابت های رویدادی `ENTITY_EVENT_UPDATE` و ثابت رویدادی `ENTITY_EVENT_DONE` است

```
virtual void ProcessEvent(SEntityEvent& event)
override
{
تابع سیستمی ProcessEvent، ثابت
ENTITY_EVENT_UPDATE را از طریق پارامتر event دریافت می
کند و از طریق دستور switch یا if بررسی می کند که چگونه بازی اجرا
شود و برای این کدها، ثابت های رویدادی مختلف بیتی دیگر را براساس نیاز
توسعه دهنده بازی تعریف و اجرا می کند.
if (event.event == ENTITY_EVENT_UPDATE)
{
از آنجایی که در داخل بازی همیشه محاسبات مختلف قوانین
ریاضی، فیزیک، رندرینگ و غیر انجام می شود، این بلوک if همیشه اجرا می
شوند که در فصل ۷ توضیحات خوبی برای آن ارائه شد
```

`SEntityUpdateContext*`

`pCtx=(SEntityUpdateContext*) event.nParam[.];`

متغیر `pCtx` بر اساس کستینگ گذاری با ساختار `SEntityUpdateContext` و با پارامتر `event` با دریافت اولین پارامتر (صفر داخل `[]`) از آرایه ای از پارامترها مقداردهی می شود

`OnUpdate(pCtx->fFrameTime);`

تابع غیر سیستمی `OnUpdate` آرگومانی با محتوای `pCtx->fFrameTime` را دریافت می کند که برش های زمانی یا همان `deltaTime` است، در واقع فریم های بازی دائما در حال اجرا هستند و این زمان است که نقش اصلی این محاسبات را انجام می دهد

}

حالا من میخوام تعداد دشمنانی که کشته می شود را در مرحله شمارش نمایم، پس باید از ثابت رویدادی `ENTITY_EVENT_DONE` استفاده کنیم و در داخل هدر فایل `StdAfx.h` یک متغیر استاتیک مانند زیر تعریف می کنم:

`static int KillEnemy=0;`

حالا دستور `if` زیر بررسی می کند که آیا دشمن از بین رفته است؟

`if (event.event == ENTITY_EVENT_DONE)`
{

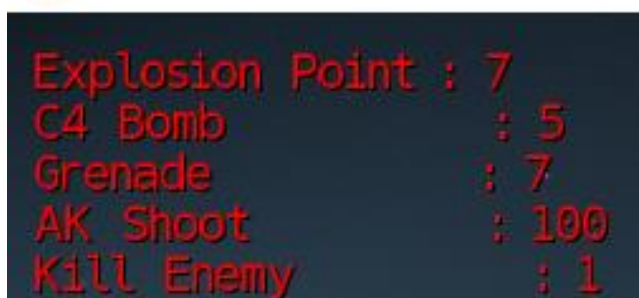
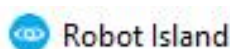
اگر دشمن را از بین ببریم یک واحد به شمارنده متغیر استاتیک `KillEnemy` اضافه می شود، همانطور که باید بدانید متغیرهای استاتیک مستقل از کلاس عمل می کنند.

`KillEnemy++;`

```
}
```

برای نمایش دادن متغیر استاتیک می توانید به فایل `Player.cpp` مراجعه کنید و در تابع `ShowWeaponAmmo` دستور زیر را وارد کنید :

```
gEnv->pAuxGeomRenderer->Draw2dLabel(۱۰, ۹۰, ۲,
ColorF(۱, ۰, ۰), false, "Kill Enemy      : "
+ ToString(KillEnemy));
```



با این کار برروی صفحه نمایش تعداد دشمنان کشته شده را نشان داده می شود و همچنین زمانی که تعداد دشمنان کشته شده برابر ۲ شد، مرحله بعدی یا هر مرحله ای که دوست دارید را می توانید لود کنید و این کد را در تابع `OnUpdate` در فایل `Player.cpp` درج کنید، شما می توانید یک کلاس ایجاد کنید که کنترل بازی و نمایش متغیرها و کنترل صدای های بازی، کنترل گرافیک بازی و ... را انجام دهد، این کار بسیار ساده است

```
if(KillEnemy==۲)
{
gEnv->pConsole->ExecuteString("map example۰۱",
false, true);
```

```

CryLog("LoadMap; -");
KillEnemy=۰;

}

}

```

```

public:
void SpawnEntityAIFollower(Ientity*
otherEntity);

```

تابعی با نام SpawnEntityAIFollower تعریف می کنیم و بعداً عملکرد آن را تشریح می کنیم

```

void OnUpdate(float deltaTime);

```

تابعی غیرسیستمی با نام OnUpdate تعریف می کنیم و بعداً عملکرد آن را تشریح می کنیم، این تابع پارامتر deltaTime یا برش های زمانی از نوع float را دریافت می کند

```

Cry::DefaultComponents::CPathfindingComponent*
m_pPathfindingComponent = nullptr;

```

یک متغیر با نام m_pPathfindingComponent را تعریف می کنیم، این متغیر براساس Navigation Area در سندباکس، نقش الگوریتم های مسیریابی را دارد.

```

Cry::DefaultComponents::CcharacterControllerCo
mponent* m_pCharacterController = nullptr;

```

یک متغیر با نام `m_pCharacterController` را تعریف می کنیم، این متغیر نقش بدنه فیزیک دشمن را دارد

```
private :  
float sum=۰;
```

یک متغیر با نام `sum` تعریف می کنیم که جمع برش های زمانی را در آن ذخیره می کنیم و مقدار اولیه آن صفر است

```
};
```

اکنون به بررسی فایل `AIFollower.cpp` می پردازم

```
#include "StdAfx.h"
```

متغیرها و توابع تعریف شده در `StdAfx.h` در داخل فایل `AIFollower.cpp` قابل دسترس خواهند بود

```
#include "AIFollower.h"
```

متغیرها و توابع تعریف شده در `AIFollower.h` در داخل فایل `AIFollower.cpp` قابل دسترس خواهند بود

```
#include <CrySchematyc/Env/IEnvRegistrar.h>
```

```
#include  
<CrySchematyc/Env/Elements/EnvComponent.h>
```

هدر فایل `EnvComponent.h` و `IEnvRegistrar.h` باعث ثبت `Component` جدید یا `Entity` جدید در سیماتیک و سندباکس خواهد شد

```
static void
RegisterAIFollowerComponent(Schematyc::IEnvRegistrar& registrar)
{
```

با توجه به تابع استاتیک `RegisterAIFollowerComponent` عمل ثبت (کامبونت جدید یا اینتیتی جدید) اجرا خواهد شد

```
Schematyc::CEnvRegistrationScope scope =
registrar.Scope(Ientity::GetEntityScopeGUID())
;
```

با توجه متغیر `scope`، برای اجرا شدن عمل ثبت، مقدار کلید `GUID` را دریافت می کند

```
Schematyc::CEnvRegistrationScope
componentScope =
scope.Register(SCHEMATYC_MAKE_ENV_COMPONENT(CAIFollowerComponent));
// Functions
{
```

متغیر `componentScope` نیز کلاس `CAIFollowerComponent` را برای ثبت (کامبونت جدید یا اینتیتی جدید) در سیماتیک و سندباکس در نظر میگیرد

```
}
```



```
}
}
```

```
CRY_STATIC_AUTO_REGISTER_FUNCTION(&RegisterAIFollowerComponent)
```

و در نهایت عمل ثبت اجرا شده و کامپونت جدید یا اینتیتی جدید در سندباکس و سیماتیک قابل نمایش خواهد بود

```
void
CAIFollowerComponent::SpawnEntityAIFollower(IEntity* otherEntity)
{
    otherEntity->SetWorldTM(m_pEntity->GetWorldTM());
}
```

تابع `SpawnEntityAIFollower` یک پارامتر با نام `otherEntity` دارد و اینتیتی که در فضای سه بعدی به این تابع ارسال می شود را در فضای سه بعدی بازی توسط تابع `SetWorldTM` قرار می دهد، اینتیتی که قرار است تغییر مکان دهد همان شی جاری (`m_pEntity`) در اینجا منظور دشمن است) با گرفتن موقعیت با تابع `GetWorldTM` است، با تابع `SpawnEntityAIFollower` می توان، تغییر موقعیت و انتقال به مکان دیگر را از طریق یک اینتیتی دیگر انجام داد چون این تابع از زیرمجموعه `public` است.

```
void CAIFollowerComponent::OnUpdate(float
deltaTime)
{
```

تابع `OnUpdate` که یک پارامتر با نام `deltaTime` را دریافت می کند، وظیفه اش به روزرسانی و پیدا کردن کوتاه ترین مسیر دشمن به طرف پلیر است.

```
if(!gEnv->pEntitySystem-
>FindEntityByName("Player"))
return;
```

برای جلوگیری از `crash` شدن بازی، باید از اینگونه خطوط از کدها مانند دستور `if` روبرو استفاده کنید، در اینجا دستور `if` بررسی می کند که زمانی ادامه کدها اجرا خواهند شد که اینتیتی `Player` وجود داشته باشد و اگر اینتیتی `Player` وجود نداشت، بقیه کدها با استفاده از خط `return;` اجرا نخواهند شد.

دقت کنید که کدنویسی درست و دستورات مستحکم (استحکام پذیری کدها) باعث جلوگیری از `bug` ها و جلوگیری از `crash` ها می شود و یکی از مهمترین مباحث در برنامه نویسی بازی ها محسوب می شود

```
m_pPathfindingComponent-
>SetMovementRecommendationCallback([this](const
Vec& recommendedVelocity)
{
تابع SetMovementRecommendationCallback در ارتباط
با سرعت حرکت دشمن به طرف پلیر است که یک پارامتر با نام
recommendedVelocity را به داخل تابع وارد می کند
```

```
m_pCharacterController-
>SetVelocity(recommendedVelocity);
```

تابع `SetVelocity` برای اختصاص دادن سرعت حرکت با توجه به بدنه فیزیکی دشمن استفاده می شود و از متغیر `m_pCharacterController` نشات می گیرد و پارامتر `recommendedVelocity` در ارتباط با شکل پایین (کادر زرد رنگی است که) با کامپونت `pathfinder` و مقدار `Maximum Acceleration` برابر ۶ در پنجره `properties` است

```
});
```

میخواهم یک تکنیک بسیار جالب با مبحثی جدید به شما آموزش دهم و آن حلقه `if-time` است

حالا حلقه `if-time` چیست؟

به دستورات زیر دقت کنید.

```
sum+=deltaTime;
```

متغیر `sum` با توجه به پارامتر `deltaTime` برش های زمانی را جمع می کند

```
if(sum>=۳)
```

```
{
```

اگر متغیر `sum` بزرگتر یا مساوی ۳ ثانیه شد، بلوکی از خط های زیر را اجرا می کند

```
if (m_pPathfindingComponent-
>IsProcessingRequest())
{
```

در دستور `if` بررسی می شود که آیا عملیات کوتاه ترین مسیر جستجو در حال انجام است؟ با توجه به تابع `IsProcessingRequest`، در صورتی که این فرآیند در حال اجرا باشد، خط بعدی اجرا می شود

```
m_pPathfindingComponent-
>CancelCurrentRequest();
```

این خط عملیات بعدی را در کوتاه ترین مسیر برای رسیدن به هدف را لغو می کند، دلیل استفاده از تابع `CancelCurrentRequest` این است که از آنجایی که پلیس می تواند به مسیرهای مختلفی رفته و از موانع مختلفی عبور کند، این مسئله باعث می شود که کوتاه ترین مسیر جستجو برای رسیدن به پلیس باید سریع مجدداً محاسبه شود و تابع `CancelCurrentRequest` به ما کمک می کند که عمل حرکت دشمن به طرف پلیس در مسیر دیگری سریع انجام شود و در واقع برای بافرینگ کوتاه ترین مسیر مربوط به پلیس در متغیر `m_pPathfindingComponent` سریع به روز رسانی شود

```
}
```

```
m_pPathfindingComponent->RequestMoveTo(gEnv-
>pEntitySystem->FindEntityByName("Player")-
>GetWorldPos());
```

با توجه به تابع `RequestMoveTo` از متغیر `m_pPathfindingComponent` عمل حرکت دشمن به طرف پلیس انجام می شود و با توجه به آرگومان `gEnv->pEntitySystem->FindEntityByName("Player")->GetWorldPos()`، پلیس پیدا می شود و مختصات جهانی آن به دشمن داده می شود و دشمن پلیس را پیدا می کند

```
sum=0;
```

متغیر `sum` که برش های زمانی را جمع می کند، صفر می شود تا مجدداً عمل مسیریابی دشمن تا سه ثانیه دیگر با مسیرهای دیگر به طرف پلیر انجام شود، دقت کنید این یک حلقه منطقی بی نهایت برپایه `if` است که آن را شبیه سازی کرده ام و این حلقه بر اساس زمان بوده و تا زمانی که دشمن وجود دارد و پلیر آن را از بین نبرده یا دشمن پلیر را از بین نبرده، حلقه `if-time` اجرا خواهد شد

```
}
```

```
Vec3 playerEntity= gEnv->pEntitySystem-
>FindEntityByName("Player")->GetWorldPos();
```

یک متغیر محلی با نام `playerEntity` را تعریف می کنیم که بر اساس تابع جستجو با نام `FindEntityByName`، مختصات پلیر را در جهان بازی ذخیره می کند و `GetEntity` نیز به اینتیتی دشمن اشاره دارد و براساس فرمول های ریاضی در محورهای `x, y, z` فاصله دشمن و پلیر در متغیرهای محلی `x, y, z` ریخته می شود.

```
float x = abs(playerEntity.x - GetEntity()-
>GetPos().x)*abs(playerEntity.x - GetEntity()-
>GetPos().x);
```

$$x = (|px - gx|) * (|px - gx|)$$

```
float y = abs(playerEntity.y - GetEntity()-
>GetPos().y)*abs(playerEntity.y - GetEntity()-
>GetPos().y);
```

$$y = (|py - gy|) * (|py - gy|)$$

```
float z = abs(playerEntity.z - GetEntity()-
>GetPos().z)*abs(playerEntity.z - GetEntity()-
>GetPos().z);
```

$$z = (|pz - gz|) * (|pz - gz|)$$

این فرمول فاصله بین دو مختصات $p(x, y, z)$ و $g(x, y, z)$ است که نقطه p منظور مختصاتی است که **Player** در آن وجود و نقطه g منظور مختصاتی است که **GetEntity** در آن موجود است، نقطه g به نقطه حضور دشمن اشاره دارد.

```
float d = sqrt(x + y + z);
```

$$d = \sqrt{x + y + z}$$

در دستور **if** بررسی می شود که اگر فاصله دشمن از پلیر (یا برعکس، قدر مطلق و رادیکال با فرجه زوج را در نظر بگیرید) یعنی متغیر محلی d کمتر از ۳ متر یا مساوی ۳ متر بود، خطوط کدهای بلوک **if** را اجرا کند

```
if (d <= ۳.f)
{
gEnv->pEntitySystem->RemoveEntity(gEnv-
>pEntitySystem->FindEntityByName("Player")-
>GetId());
```

با توجه به تابع `RemoveEntity` که عمل حذف اینتیتی را در بازی انجام می دهد، آرگومانی باید به آن اختصاص داد و با توجه به `id` پلیر که از تابع `GetId` توسط پیدا کننده اینتیتی نشات می گیرد، تابع `FindEntityByName`، پلیر را برای ما پیدا می کند و `id` آن را در اختیار تابع حذف اینتیتی قرار می دهد و پلیر حذف می شود

```
GetEntity()->RemoveAllComponents();
```

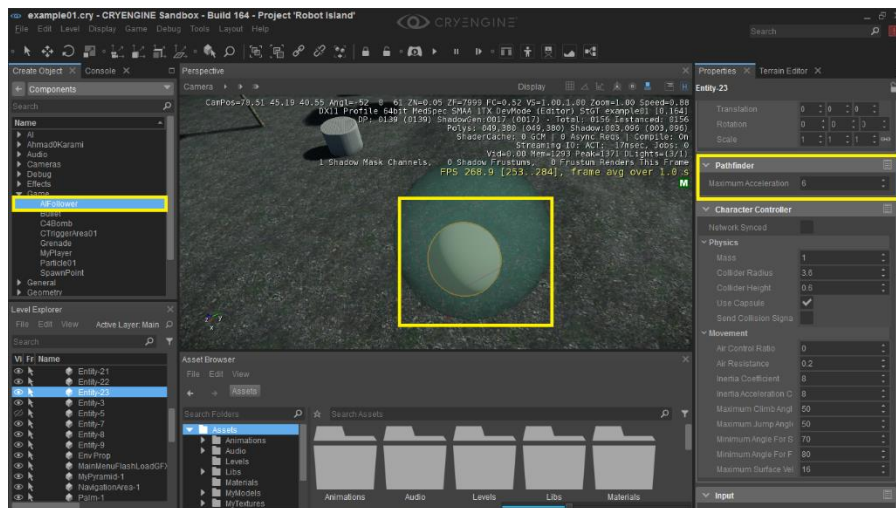
با توجه به شی جاری یعنی `GetEntity` تمامی کامبونت های دشمن حذف می شود

```
gEnv->pEntitySystem-  
>RemoveEntity(GetEntityId());
```

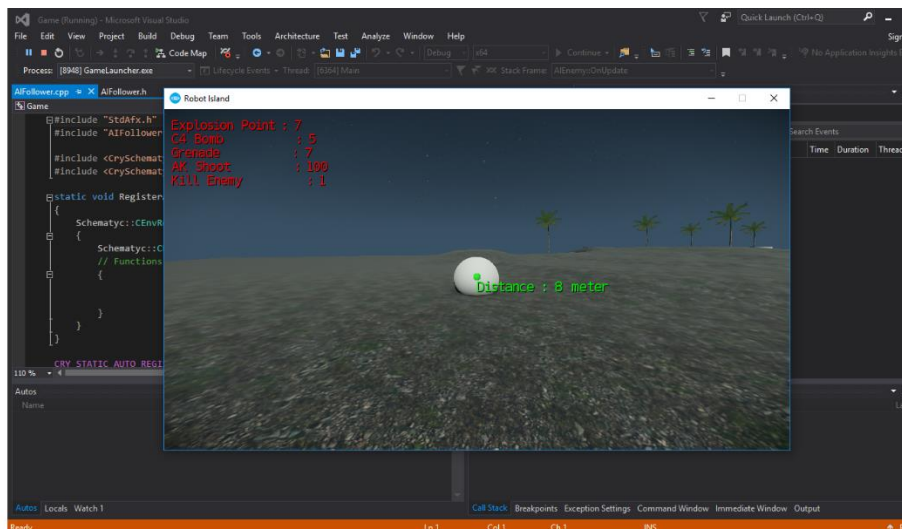
و سپس با توجه به تابع حذف اینتیتی `RemoveEntity`، دشمن نیز باید حذف شود و آرگومان آن تابع را با `GetEntityId` یعنی `id` دشمن در نظر میگیریم و دشمن نیز حذف خواهد شد

```
}  
}
```

فایل های `AI Follower.h` و `AI Follower.cpp` را در فایل `CMakeLists.txt` اضافه کنید و سپس پروژه را `Rebuild` کنید و یک پیغام نمایش داده می شود و دکمه `Reload All` را انتخاب کنید و سپس پروژه را `Play` کنید، این مبحث را در فصل ۶ کامل توضیح داده ام و برای هر هدر فایل `*.h` و سی پی پی فایل `*.cpp` باید انجام شود

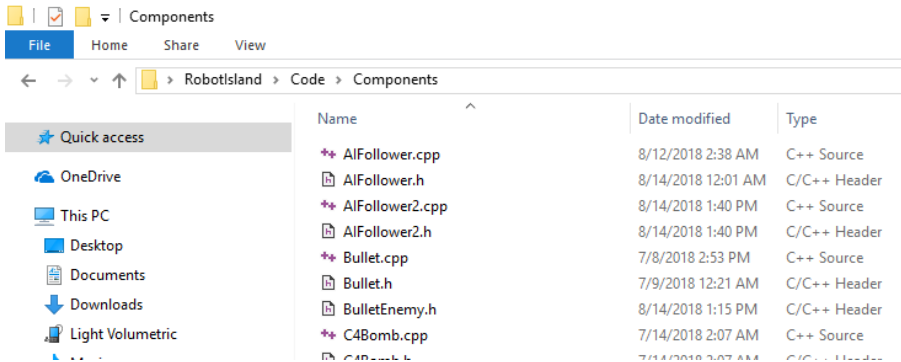


نتیجه نهایی در این تصویر نمایش داده شده است و در بخش components در پنجره Create Object یک کامپونت با اسم AIFollower وجود دارد و یک کامپونت با نام pathfinder با سرعت حرکت شتابی ۶ براساس پارامتر Maximum Acceleration در پنجره properties وجود دارد که می توانید سرعت دشمن را افزایش و یا کاهش دهید و یک کامپونت دیگر با نام Character Controller وجود دارد که جرم آن یعنی Mass برابر ۱ در بخش Physics بوده و پارامترهای دیگر در بخش Movement یعنی حرکت که پیشنهاد می دهیم آن را تغییر دهید تا نتایج بیشتری را از دشمن طراحی شده بدست آورید.



حالا بیاید یک مثال جدی را مورد بحث قرار دهیم و مطمئنا این مثال انگیزه شما را افزایش می دهد که زبان C++ را در کرای انجین کاربردی تر یاد بگیرید، باید یک entity (به صورت component) در پنجره Create Object در بخش Components با نام AIFollower۲ ثبت نمایم که دشمنی نسبتا پیشرفته است و پلیس را تعقیب می کند و در فاصله ۵ متری ایستاده و به طرف پلیس گلوله پرتاب می کند، البته این دشمن را با نود EntityFaceAt در فلوگراف مجهز کرده ام و دشمن دیگر در همان ساختار C++ وجود دارد که همان حملات را انجام می دهد اما وقتی که به پلیس نزدیک شد براساس دوربین پلیس جهت حمله به پلیس را بدون استفاده از نود EntityFaceAt مشخص و اجرا می کند، البته با اجرای پروژه Robot Island بهتر متوجه منظور من خواهید شد.

ابتدا هدر فایل AIFollower۲.h را بررسی می کنم، این هدر فایل باید در مسیر زیر ایجاد شود :



```
#pragma once
```

```
#include <ICryMannequin.h>
```

```
#include
```

```
<DefaultComponents/Cameras/CameraComponent.h>
```

```
#include
```

```
<DefaultComponents/Physics/CharacterController  
Component.h>
```

```
#include
```

```
<DefaultComponents/Geometry/AdvancedAnimationC  
omponent.h>
```

```
#include
```

```
<DefaultComponents/AI/PathfindingComponent.h>
```

چند هدر فایل بالا به ما کمک می کند که از توابع و متغیرهای مختلف در داخل این هدر فایل ها استفاده کنیم، آنها را فراخوانی کنیم و به کار ببندیم

```
class AIEntity final : public IEntityComponent  
{
```

یک کلاس با نام `AIEnemy` به صورت نهایی و بدون ارث بری بعدی تعریف می کنیم و از رابط `IEntityComponent` برای پیاده سازی اینتیتی ها و کامپونت ها استفاده می کنیم

```
public:
```

```
AIEnemy() = default;
```

سازنده کلاس را به صورت پیش فرض با کلمه `default` مقداردهی می کنیم

```
virtual ~AIEnemy() {}
```

و همینطور تخریب کننده کلاس را نیز به صورت مجازی تعریف می کنیم

```
virtual void Initialize() override;
```

رویداد سیستمی (یا تابع سیستمی نیز گفته می شود) با نام `Initialize` را به صورت مجازی با خاصیت تعریف مجدد اعلان می کنیم

```
virtual uint64 GetEventMask() const override;
```

رویداد سیستمی `GetEventMask` را به صورت مجازی با خاصیت تعریف مجدد اعلان می کنیم

```
virtual void ProcessEvent(SEntityEvent& event) override;
```

رویداد سیستمی `ProcessEvent` را به صورت مجازی با خاصیت تعریف مجدد اعلان می کنیم

```
static void
ReflectType(Schematyc::CTypeDesc<AIEnemy>&
desc)
{
```

رویداد سیستمی `ReflectType` ایستایی، باعث تعریف کلاس در سیماتیک ادیتور و در سیستم `Entity` `Component` در پنجره `Create Object` در بخش `Components` خواهد شد، کلاس `AIEnemy` را به تابع `ReflectType` اختصاص می دهیم

```
desc.SetGUID("{D۳۴D۸C۱C-۹۱۰C-۴DFC-B۹۲A-۱۱۸F۸۹۸AB۰۹۵}"_cry_guid);
```

به منوی `Tools` در ویژوال استودیو رفته بر روی گزینه `Create GUID` مراجعه می کنیم و یک `GUID` از نوع `Registry Format` انتخاب و کپی می کنیم در خط `desc.SetGUID` قرار می دهیم

```
desc.SetEditorCategory("Game");
```

در چه کاتالوگی می خواهید `entity` یا `component` تان ثبت شود، در اینجا کاتالوگ `Game` استفاده شده اما اگر نام دیگری برای کاتالوگ تان مانند `My AI` استفاده کنید، یک کاتالوگ با اسم `My AI` در پنجره `Create Object` در بخش `Components` در سندباکس ایجاد خواهد شد

```
desc.SetLabel("AIFollower۲");
```

اسم entity یا component تان چه باشد؟ در اینجا اسمی که در نظر گرفته ایم `AI Follower` است

```
desc.SetDescription("This AI Follower can be  
used to spawn entities");
```

توضیحاتی برای entity یا component که در حال ایجاد است، در نظر گرفته ایم، این توضیحات صرفاً جهت راهنمایی است

```
desc.SetComponentFlags({  
    IEntityComponent::EFlags::Transform,  
    IEntityComponent::EFlags::Socket,  
    IEntityComponent::EFlags::Attach });
```

در اینجا پرچم‌هایی برای entity یا component در حال ایجاد را در نظر می‌گیریم که باید به صورت خط بالا باشد

```
}
```

```
void Revive();
```

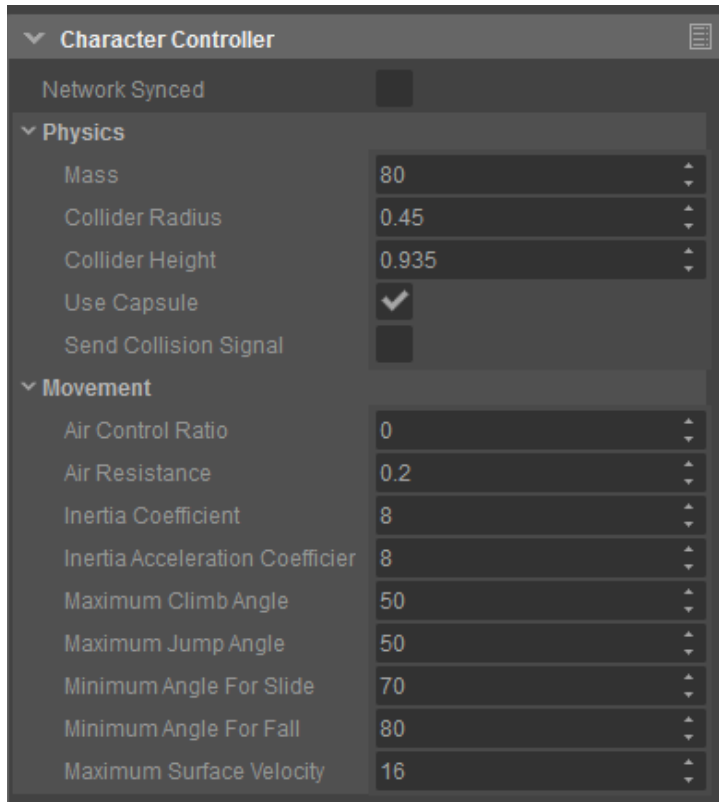
تابع `Revive` برای `reset` کردن و محاسبه برخی از اعمال کامپونت‌ها و متغیرها را شامل می‌شود، عملکرد این تابع را در ادامه این فصل تشریح می‌کنم

```
void OnUpdate(float deltaTime);
```

تابع غیر سیستمی `OnUpdate` برش‌های زمانی را دریافت می‌کند، عملکرد این تابع را در ادامه این فصل تشریح می‌کنم

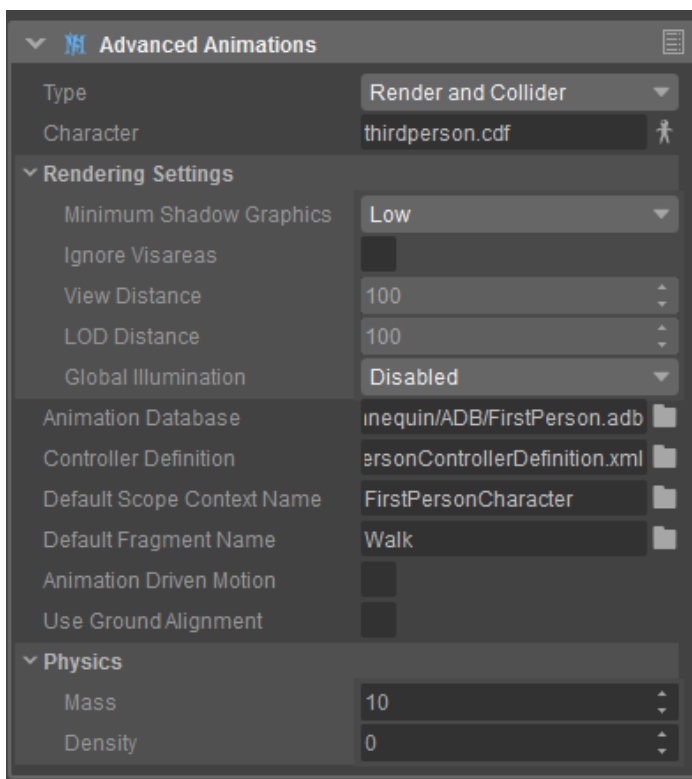
```
protected:
```

```
Cry::DefaultComponents::CCharacterControllerComponent* m_pCharacterController = nullptr;
```



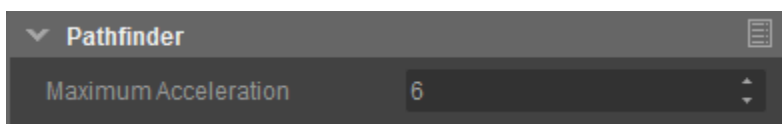
متغیری با نام `m_pCharacterController` برای ساخت یک کامپونت فیزیک از نوع `CharacterController` تعریف شده است

```
Cry::DefaultComponents::CAdvancedAnimationComponent* m_pAnimationComponent = nullptr;
```



متغیر با نام `m_pAnimationComponent` برای ساخت یک کامپونت انیمیشن از نوع انیمیشن پیشرفته تعریف شده است

```
Cry::DefaultComponents::CPathfindingComponent*
m_pPathfindingComponent = nullptr;
```



متغیر با نام `m_pPathfindingComponent` از نوع کوتاه ترین مسیر به طرف هدف (همان پلیر) تعریف شده است.

```
FragmentID m_walkTagId;
```

یک متغیر با نام `m_walkTagId` برای گرفتن کلیپ انیمیشنی `walk` تعریف شده است.

```
FragmentID m_IdleTagId;
```

یک متغیر با نام `m_IdleTagId` برای گرفتن کلیپ انیمیشنی `Idle` تعریف شده است.

```
FragmentID m_activeFragmentId;
```

یک متغیر با نام `m_activeFragmentId` برای گرفتن کلیپ انیمیشن جاری در حال پخش تعریف شده است

```
float sum=۰;
```

متغیر `sum` برای جمع برش های زمانی کاربرد دارد و از نوع داده اعشاری `float` است

```
bool isOnce=true;
```

متغیر `isOnce` از نوع داده `boolean` این تضمین را می دهد که فقط یکبار در هر شرط مجدد کدهای مشخصی اجرا شود

```
};
```


فایل AIFollower۲.cpp را بررسی می کنیم، این فایل باید در مسیر زیر ایجاد شود :

RobotIsland\Code\Components

```
#include "StdAfx.h"
```

اگر می خواهید از متغیرهای ایستا یا توابع تعریف و پیاده سازی شده ایستا در هدر فایل StdAfx.h استفاده کنید، باید آن را با کلمه کلیدی `include` به پروژه بازی تان الصاق کنید، این هدر فایل جزء هدر فایل ریشه در پروژه بازی است

```
#include "AIFollower۲.h"
```

متغیرها و توابع را در این هدر فایل اعلان کردیم و این هدر فایل در ارتباط با فایل AIFollower۲.cpp است و توابع را باید در فایل AIFollower۲.cpp پیاده سازی کنیم

```
#include "BulletEnemy.h"
```

متغیرها و توابع مربوط به گلوله ها را در هدر فایل BulletEnemy.h اعلان کردیم و باید توابع را نیز در هدر فایل BulletEnemy.h پیاده سازی می کنیم، در ادامه این فصل به تشریح BulletEnemy.h می پردازم.

```
#include "Particle۰۱.h"
```

برای استفاده از کلاس آتش، باید متغیرها و توابع هدر فایل Particle۰۱.h را به پروژه الصاق نماییم.

```
#include <CryRenderer/IRenderAuxGeom.h>
```

برای استفاده از توابع رندرینگ، نمایش متن و شکل های هندسی، هدرفایلی با نام `IRenderAuxGeom.h`، را به پروژه الصاق نماییم

```
#include <CryInput/IHardwareMouse.h>
```

برای استفاده از ماوس باید این هدرفایل را به پروژه الصاق کنیم

```
#include <CrySchematyc/Env/IEEnvRegistrar.h>
```

برای ثبت و استفاده از این فایل ها با نام `AIFollower۲.h` و `AIFollower۲.cpp` در سیماتیک و پنجره `Create Object` در بخش `Components`، باید از این هدرفایل استفاده کنیم.

```
#include  
<CrySchematyc/Env/Elements/EnvComponent.h>
```

برای ثبت و استفاده از این فایل ها با نام `AIFollower۲.h` و `AIFollower۲.cpp` در سیماتیک و پنجره `Create Object` در بخش `Components`، نیز باید از این هدرفایل استفاده کنیم.

```
static void  
RegisterAIFollower۲Component(Schematyc::IEEnvRe  
gistrar& registrar)  
{
```

برای ثبت `component` در محیط سیماتیک و سندباکس به صورت ثبت کننده محیطی باید از این تابع ایستا (همان استاتیک) استفاده نمود، این عمل در ارتباط با موازی کاری فایل هدر تعریف شده بالا با نام `AIFollower۲.h` است، داخل این تابع خطوط زیر وجود دارند:

```

Schematyc::CEnvRegistrationScope scope =
registrar.Scope(IEntity::GetEntityScopeGUID())
;
{
Schematyc::CEnvRegistrationScope
componentScope =
scope.Register(SCHEMATYC_MAKE_ENV_COMPONENT(AI
Enemy));
// Functions
{

```

اینجا محدوده ای با نام **scope** وجود دارد که تبدیلات زبان ماشین یا همان صفر و یک انجام می شود با استفاده از **guid** باید عمل ثبت کننده را انجام دهیم تا در دفتر ویندوز یا همان ریجستری ویندوز انجام شود، این عمل در ادامه با ثبت **component** و **entity** در سندباکس و در نهایت برای محیط سیماتیک نیز انجام می شود و شما با استفاده از پیش پردازنده ماکرویی **#define** یک کامپوننت جدید در سیماتیک (**SCHEMATYC_MAKE_ENV_COMPONENT**) با نام کلاس **AIEnemy** ایجاد می کنید و بدنه آن بدون تابع است (احتمالا این بخش در رابطه با ایجاد نودهای مختلف در سیماتیک است)

```

}
}
}

```

```

CRY_STATIC_AUTO_REGISTER_FUNCTION(&RegisterAIF
ollowerComponent)

```

تابع

```
CRY_STATIC_AUTO_REGISTER_FUNCTION(&RegisterAIFollowerComponent)
```

توسط ماکرو پیش پردازنده عمل ثبت entity و component را با موفقیت انجام می دهد (البته اگر کدها را درست تایپ کرده باشید)

```
void AIEnergy::Initialize()
{
```

```
m_pCharacterController = m_pEntity-
>GetOrCreateComponent<Cry::DefaultComponents::
CCharacterControllerComponent>();
```

با توجه به شی جاری و متغیر m_pCharacterController یک کامپونت Character Controller ایجاد می کنیم

```
m_pCharacterController-
>SetTransformMatrix(Matrix::Create(Vec(۱.f),
IDENTITY, Vec(۰, ۰, ۱.f)));
```

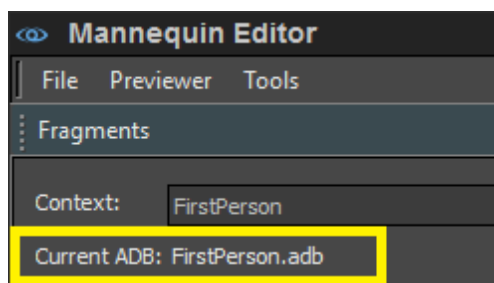
براساس ماتریکس انتقال با واحد همانی از متغیر m_pCharacterController با آرگومان IDENTITY (x=۱, y=۱, z=۱) و یک افسر برای کاراکتر کنترلر با مختصات (x=۰, y=۰, z=۱) در نظر می گیریم

```
m_pAnimationComponent = m_pEntity-
>GetOrCreateComponent<Cry::DefaultComponents::
CAdvancedAnimationComponent>();
```

با توجه به شی جاری و متغیر `m_pAnimationComponent` یک کامبونت انیمیشن پیشرفته ایجاد می کنیم تا انیمیشن ها را از طریق مانکن ادیتور برای دشمن به روزرسانی کنیم

```
m_pAnimationComponent->SetMannequinAnimationDatabaseFile("Animations/Mannequin/ADB/FirstPerson.adb");
```

طبق مسیر پوشه های تودرتو و ختم شده به فایل `FirstPerson.adb` در مانکن ادیتور، انیمیشن های مختلف برای سیستم پیشرفته در متغیر `m_pAnimationComponent` تعریف می شود، از آنجایی که فقط فایل `FirstPerson.adb` وجود دارد پس برای دشمن نیز از همین فایل استفاده می کنیم.

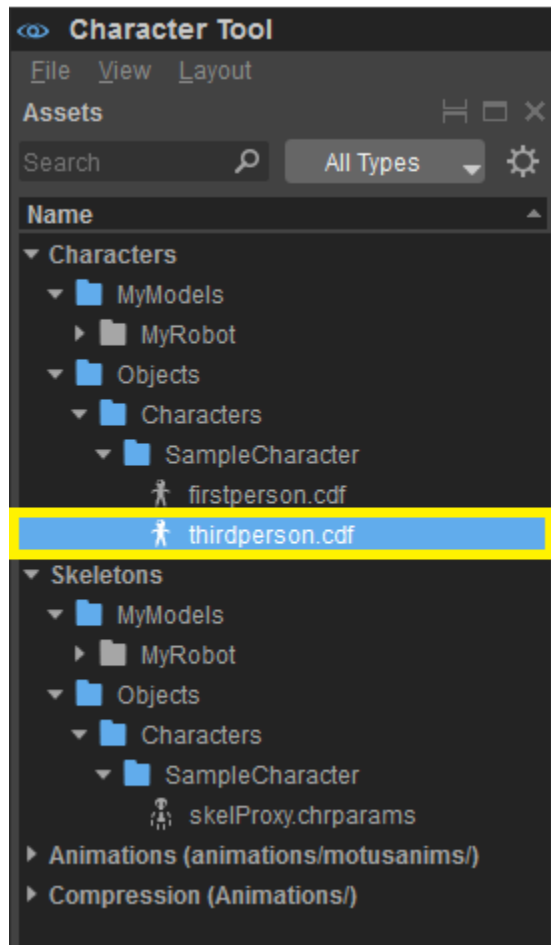


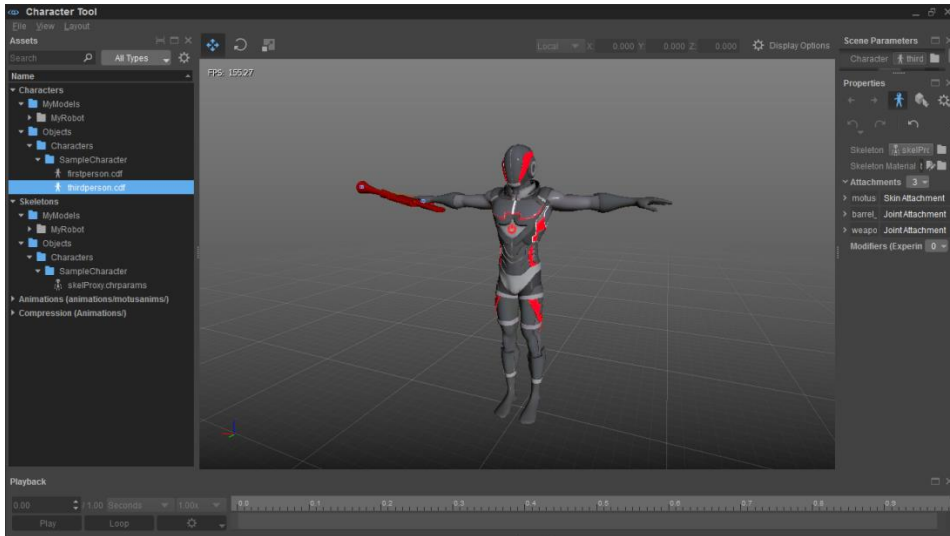
```
m_pAnimationComponent->SetCharacterFile("Objects/Characters/SampleCharacter/thirdperson.cdf");
```

با توجه به مسیر تودرتو و ختم شده به فایل `thirdperson.cdf`، مدل سه بعدی ریگ شده به متغیر `m_pAnimationComponent` اختصاص داده می شود و در پنجره `Character Tool` فایل های

firstperson.cdf و thirdperson.cdf وجود دارند، به

شکل زیر نگاه کنید



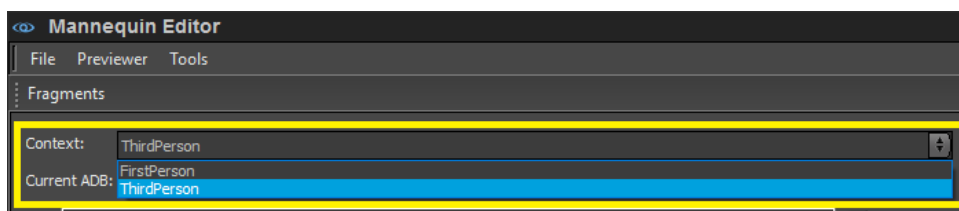


```
m_pAnimationComponent-
>SetControllerDefinitionFile("Animations/Mannequin/ADB/FirstPersonControllerDefinition.xml")
;
```

کلیه لایه های انیمیشنی و کلیپ های انیمیشنی Walk ، Idle در مانکن ادیتور در یک فایل با نام FirstPersonControllerDefinition.xml قرار داده شده است و این فایل توسط مانکن ادیتور ایجاد می شود، زیرا که تنها فایل FirstPersonControllerDefinition.xml در پروژه موجود است.

```
m_pAnimationComponent-
>SetDefaultScopeContextName("FirstPersonCharacter");
```

طبق تصویر زیر نوع context مدل که از نوع سه بعدی اول شخص یا سوم شخص خواهد بود با این خط کد در بالا نوع context باید از FirstPersonCharacter تبعیت کند



```
m_pAnimationComponent-  
>SetDefaultFragmentName("Walk");
```

با توجه به صف کلیپ های انیمیشنی باید یک انیمیشن را به حالت پیش فرض انتخاب کنیم که پلیر آن را اجرا نماید، با توجه به این خط انیمیشن Walk نمایش داده می شود و به صورت پیش فرض و همیشگی این انیمیشن اجرا خواهد شد، مگر اینکه کلیدهای دیگری برای پخش انیمیشن های دیگر تعریف شود، مثل انیمیشن معطلی (انتظار) که در خطوط بعدی به این مسئله می پردازم.

```
m_pAnimationComponent-  
>SetAnimationDrivenMotion(false);
```

این خط کنترل می کند که توسط فیزیک حرکت انجام شود، در اینجا افسست مربوط به نقاط اتصال ها غیرفعال شده است چون مقدار false برای تابع SetAnimationDrivenMotion فرستاده شده است.

```
m_pAnimationComponent->LoadFromDisk();
```


عمل بارگذاری یا لود کردن داده های کاراکتر و مانکن ادیتور برای دشمن با این خط انجام می شود

```
m_IdleTagId = m_pAnimationComponent-
>GetFragmentId("Idle");
```

این خط با توجه به متغیر m_IdleTagId آدرس دسترسی به کلیپ انیمیشن Idle را برای دشمن مهیا می کند

```
m_walkTagId = m_pAnimationComponent-
>GetFragmentId("Walk");
```

این خط با توجه به متغیر m_walkTagId آدرس دسترسی به کلیپ انیمیشن Walk را برای دشمن مهیا می کند

```
m_pPathfindingComponent = m_pEntity-
>GetOrCreateComponent<Cry::DefaultComponents::
CPathfindingComponent>();
```

با توجه به شی جاری و متغیر m_pPathfindingComponent یک کامبونت با هدف کوتاه ترین مسیر برای رسیدن به پلیس را ایجاد می کنیم

```
m_pPathfindingComponent-
>SetMovementRecommendationCallback([this](const
Vecr& recommendedVelocity)
{
m_pCharacterController-
>SetVelocity(recommendedVelocity);
```

```
});  
}
```

حالا با توجه به متغیر `m_pPathfindingComponent` از تابع `SetMovementRecommendationCallback` استفاده می‌کنیم تا محاسبه کوتاه‌ترین مسیر از دشمن به طرف پلیر انجام شود اما نحوه حرکت را بر اساس تابع فیزیک شتاب حرکتی با نام `SetVelocity` از متغیر کامپوننتی `m_pCharacterController` استفاده می‌کنیم

```
uint64 AIEnergy::GetEventMask() const  
{  
    return BIT64(ENTITY_EVENT_START_GAME) |  
    BIT64(ENTITY_EVENT_UPDATE);  
}
```

تابع `GetEventMask` دو مقدار را برای تابع `ProcessEvent` به وسیله ماکرو `BIT64` برمیگرداند، ثابت رویدادی `ENTITY_EVENT_START_GAME` و `ENTITY_EVENT_UPDATE`

```
void AIEnergy::ProcessEvent(SEntityEvent& event)  
{  
    switch (event.event)  
    {
```

پارامتر `event` را در داخل دستور `switch` بررسی می‌کنیم

```
case ENTITY_EVENT_START_GAME:  
{
```

ثابت رویدادی ENTITY_EVENT_START_GAME در هنگام شروع بازی اجرا می شود و تابع Revive را اجرا خواهد کرد

```
Revive();

}
break;

case ENTITY_EVENT_UPDATE:
```

تا زمانی که بازی در حال اجرا است، ثابت رویدادی ENTITY_EVENT_UPDATE نیز اجرا خواهد شد

```
SentityUpdateContext*          pCtx          =
(SentityUpdateContext*)event.nParam[.];
```

برای دریافت برش های زمانی یک متغیر با نام pCtx تعریف می کنیم که عمل cast شده برروی پارامتر event را براساس nParam[.] انجام دهد

```
OnUpdate(pCtx->fFrameTime);
```

آرگومان برش های زمانی یا همان pCtx->fFrameTime را به داخل تابع OnUpdate ارسال می کنیم

```
}
break;
}
}
```

```
void AIEnergy::Revive()
{
```

```
    GetEntity()->Hide(false);
```

آیا شی جاری مخفی شود؟ از آنجایی که شی جاری از تابع Hide استفاده می کند و آرگومان آن false است، شی جاری مخفی نخواهد شد

```
    GetEntity()-
    >SetWorldTM(Matrix34::Create(Vec3(۱, ۱, ۱),
    IDENTITY, GetEntity()->GetWorldPos()));
```

باید اطمینان داشته باشیم که شی جاری یا همان دشمن براساس تابع SetWorldTM در فضای سه بعدی با استفاده از تابع GetWorldPos قرار می گیرد و از ماتریکس همانی از آرگومان IDENTITY با واحدهای $x=1$ ، $y=1$ و $z=1$ استفاده می شود.

```
m_pAnimationComponent->ResetCharacter();
```

عمل صف بندی مجدد با reset شدن کاراکتر به وسیله متغیر m_pAnimationComponent انجام می شود

```
m_pCharacterController->Physicalize();
```

محاسبات فیزیکی بدنۀ دشمن به وسیله متغیر m_pAnimationComponent با تابع Physicalize برای حرکت کردن در فضای سه بعدی انجام می شود.

```
}
```

```
void AIEnergy::OnUpdate(float deltaTime)
{
```

دشمن همه اتفاقات را در تابع `OnUpdate` انجام می دهد، این تابع مجهز به دریافت پارامتر برش های زمانی با نام `deltaTime` است

```
sum += deltaTime;
```

متغیر `sum` برش های زمانی را جمع می کند، تا ثانیه ها تشکیل شود

```
if (sum >= ۱ && m_pPathfindingComponent &&
gEnv->pEntitySystem-
>FindEntityByName("Player"))
{
```

در بلوک `if`، می بینید که سه شرط وجود دارد که باید رعایت شوند زیرا که از عملگر منطقی `and` با علامت `&&` استفاده شده است و مقدار هر سه شرط باید `true` گردند تا کدهای داخل بلوک اجرا شوند :

`sum >= ۱` : زمانی که برش های زمانی در متغیر `sum` بزرگتر از یک ثانیه یا مساوی یک ثانیه شد، مقدار `true` برگشت داده می شود

`m_pPathfindingComponent` : از آنجایی که قبلا یک کامپوننت از نوع کوتاه ترین مسیر به هدف را ایجاد کردیم و این متغیر کامپوننتی خالی نیست و مقدار `nullptr` ندارد، مقدار `true` برگشت داده می شود، این شرط باید وجود داشته باشد و وجود این متغیر جزء استحکام پذیری کدها است و این متغیر باعث مقابله در برابر خطا در زمان اجرای بازی خواهد بود

```
gEnv->pEntitySystem-
>FindEntityByName("Player"))
```

که آیا `Player` در مرحله وجود دارد یا نه؟ اگر وجود داشته باشد مقدار

true برگشت داده می شود و این شرط باید وجود داشته باشد و وجود جستجو **Player** جزء استحکام پذیری کدها است و این شرط باعث مقابله در برابر خطا در زمان اجرای بازی خواهد بود.

کدهای استحکام پذیری دیگری نیز وجود دارند که شما می توانید با تحلیل درست به اینگونه کدها دست پیدا کنید

```
if (m_pPathfindingComponent-
>IsProcessingRequest())
{
```

در دستور **if** بررسی می شود که آیا عملیات کوتاه ترین مسیر جستجو در حال انجام است؟ با توجه به تابع **IsProcessingRequest**، در صورتی که این فرآیند در حال اجرا باشد، خط بعدی اجرا می شود

```
m_pPathfindingComponent-
>CancelCurrentRequest();
```

این خط عملیات بعدی را در کوتاه ترین مسیر برای رسیدن به هدف را لغو می کند، دلیل استفاده از تابع **CancelCurrentRequest** این است که از آنجایی که پلیر می تواند به مسیرهای مختلفی رفته و از موانع مختلفی عبور کند، این مسئله باعث می شود که کوتاه ترین مسیر جستجو برای رسیدن پلیر، سریع باید محاسبه مجدد شود و تابع **CancelCurrentRequest** به ما کمک می کند که عمل حرکت دشمن به طرف پلیر در مسیر دیگری سریع انجام شود و در واقع بافرینگ مربوط به مسیریهای مختلف در متغیر **m_pPathfindingComponent** به روز رسانی شود.

}

```
m_pPathfindingComponent->RequestMoveTo(gEnv-
>pEntitySystem->FindEntityByName("Player")-
>GetWorldPos());
```

با توجه به تابع RequestMoveTo از متغیر m_pPathfindingComponent، عمل حرکت دشمن به طرف پلیر انجام می شود و با توجه به آرگومان gEnv->pEntitySystem->FindEntityByName("Player")->GetWorldPos()، پلیر پیدا می شود و مختصات جهانی آن به دشمن داده می شود و دشمن پلیر را پیدا می کند

```
sum = ۰;
```

متغیر sum که برش های زمانی را جمع می کند، صفر می شود تا مجدداً عمل مسیریابی دشمن تا یک ثانیه دیگر با مسیرهای دیگر به طرف پلیر انجام شود، دقت کنید این یک حلقه منطقی بی نهایت برپایه if است که آن را شبیه سازی کرده ام و این حلقه بر اساس زمان بوده و تا زمانی که دشمن وجود دارد و پلیر آن را از بین نبرده یا دشمن پلیر را از بین نبرده، حلقه if-time اجرا خواهد شد.

```
const auto& desiredFragmentId =
m_pCharacterController->IsWalking() ?
m_walkTagId : m_IdleTagId;
```

یک ثابت با نام desiredFragmentId تعریف شده که بررسی می کند که آیا متغیر m_pCharacterController در حال حرکت (در حال قدم زدن) است؟ که اگر در حال حرکت باشد با توجه به متغیر

`m_walkTagId` و اگر در حال حرکت نباشد و در یک مکان ایستاده باشد با توجه به متغیر `m_IdleTagId`، انیمیشن با `FragmentId` مربوط به شرط در متغیر `desiredFragmentId` قرار خواهد گرفت

```
if (m_activeFragmentId != desiredFragmentId)
{
```

بلوک `if` بررسی می کند که اگر انیمیشن در حال پخش شدن در متغیر `desiredFragmentId` مخالف انیمیشن در حال پخش شدن `m_activeFragmentId` فعلی باشد، آنگاه دو خط بلوک `if` اجرا شوند

```
m_activeFragmentId = desiredFragmentId;
```

از آنجایی که متغیرهای انیمیشنی با `Id` مربوط `desiredFragmentId` و `m_activeFragmentId` مخالف هستند، پس انیمیشن جاری همان انیمیشن `desiredFragmentId` خواهد بود

```
m_pAnimationComponent->QueueFragmentWithId(m_activeFragmentId);
```

با توجه به متغیر کامپوننتی `m_pAnimationComponent`، انیمیشن جاری در متغیر `m_activeFragmentId` در صف پخش شدن انیمیشن قرار خواهد گرفت، قرارگیری پخش انیمیشن ها در صف با تابع `QueueFragmentWithId` انجام می شود.


```
}
```

```
if (IcharacterInstance *pCharacter =  
m_pAnimationComponent->GetCharacter())  
{
```

در دستور `if` با توجه به متغیر `m_pAnimationComponent` و تابع آن با نام `GetCharacter`، ساختار `bone` که نقطه اتصال اسلحه و دست دشمن را دریافت می کند و در داخل متغیر اشاره گری `pCharacter` می ریزد.

```
auto *pBarrelOutAttachment = pCharacter-  
>GetIAttachmentManager()-  
>GetInterfaceByName("barrel_out");
```

متغیر `pCharacter` نحوه دسترسی به نقطه اتصالی که باعث تولید گلوله می شود را آدرس دهی می کند، این آدرس دهی با توجه به تابع `GetInterfaceByName` با آرگومان `barrel_out` صورت می پذیرد، دستور `if` بررسی می کند که آیا اتصالی با نام `barrel_out` وجود دارد که این اتصال در متغیر `pBarrelOutAttachment` آدرس دهی شده است و در صورتی که این متغیر با مقدار خالی (`nullptr`) مخالف باشد، یعنی نقطه اتصال `barrel_out` وجود دارد.

```
if (pBarrelOutAttachment != nullptr)  
{
```

```
QuatTS bulletOrigin = pBarrelOutAttachment-  
>GetAttWorldAbsolute();
```

نقطه اتصال barrel_out طبق تابع GetAttWorldAbsolute در فضای سه بعدی در متغیر bulletOrigin ریخته می شود

```
SEntitySpawnParams spawnParams;
```

یک متغیر با نام spawnParams از نوع ساختمان داده SEntitySpawnParams تعریف می کنیم و وظیفه این متغیر آدرس دهی برای تولید گلوله های دشمن در نقطه اتصال barrel_out است

```
spawnParams.pClass = gEnv->pEntitySystem-
>GetClassRegistry()->GetDefaultClass();
```

طبق فیلد pClass از متغیر spawnParams باید کلاسی پیش فرض و ثبت شده به آن اختصاص دهیم تا بتوانیم تولید گلوله را انجام دهیم

```
spawnParams.vPosition = bulletOrigin.t;
```

متغیر bulletOrigin از نوع داده QuatTS بوده و باید فضای سه بعدی نقطه اتصال barrel_out را با استفاده از bulletOrigin.t در فیلد vPosition (موقعیت) در متغیر spawnParams بریزیم.

```
spawnParams.qRotation = bulletOrigin.q;
```

برای عمل چرخش نیز باید فضای سه بعدی نقطه اتصال barrel_out را با استفاده از bulletOrigin.q در فیلد qRotation در متغیر spawnParams بریزیم.

```
const float bulletScale = ۰.۰۵f;
```

اندازه گلوله با استفاده از ثابت `bulletScale` اندازه گیری می کنیم

```
spawnParams.vScale = Vec3(bulletScale);
```

سپس با توجه به بردار سه بعدی مربعی برای اندازه گلوله، فیلد `vScale` را در متغیر `spawnParams` می ریزیم، تا طول، عرض و ارتفاع گلوله به یک اندازه لحاظ شود

```
if (IEntity* pEntity = gEnv->pEntitySystem-
>SpawnEntity(spawnParams))
{
```

حالا با توجه به دستور `if` و شرط داخل آن، باید متغیر `spawnParams` را به وسیله تابع `SpawnEntity` تکثیر کنیم، با ایجاد و آدرس دهی `entity` ایجاد شده، باید این `entity` را در داخل متغیر `pEntity` بریزیم، اگر عمل تکثیر `entity` با موفقیت انجام شد، خط زیر اجرا می شود

```
pEntity-
>CreateComponentClass<CBulletEnemyComponent>()
;
```

حالا `entity` ما ایجاد شده است و به وسیله متغیر `pEntity` باید یک کلاس کامپوننتی توسط تابع جنریک `CreateComponentClass` برای تولید گلوله ایجاد کنیم و این کلاس کامپوننتی با نام `CBulletEnemyComponent` است و در داخل متغیر `pEntity` آدرس دهی خواهد شد.

```
}
}
}
}
```

در ادامه تابع `OnUpdate` به دستور `if` دیگری برخورد می کنیم که بررسی می کند که آیا `Player` در مرحله وجود دارد یا نه؟

```
if(gEnv->pEntitySystem-
>FindEntityByName("Player"))
{
```

در صورتی که `Player` وجود داشت، دستورات داخل بلوک `if` اجرا می شوند و اگر `Player` وجود نداشت، دستورات داخل بلوک `if` اجرا نخواهند شد، برای دانستن اینکه `Player` در مرحله وجود دارد یا نه از تابع `FindEntityByName` استفاده می کنیم

```
Vec3 playerEntity = gEnv->pEntitySystem-
>FindEntityByName("Player")->GetWorldPos();
```

متغیر برداری `playerEntity`، موقعیت پلیمر را در فضای سه بعدی به وسیله تابع `GetWorldPos` بدست می آورد

```
float x = abs(playerEntity.x - GetEntity()-
>GetPos().x)*abs(playerEntity.x - GetEntity()-
>GetPos().x);
float y = abs(playerEntity.y - GetEntity()-
>GetPos().y)*abs(playerEntity.y - GetEntity()-
>GetPos().y);
float z = abs(playerEntity.z - GetEntity()-
>GetPos().z)*abs(playerEntity.z - GetEntity()-
>GetPos().z);

float d = sqrt(x + y + z);
```

در همین فصل در رابطه با این فرمول ها برای محاسبه فاصله ی دو نقطه از فضای سه بعدی بحث شد و از توضیح مجدد آن صرف نظر می کنم و در نهایت متغیر `d` فاصله دشمن تا پلیمر (یا برعکس) را بدست می آورد

```
if (d <= ۵.f)
```

```
{
```

با توجه به متغیر `d` و دستور `if`، بررسی می شود که آیا فاصله دشمن از پلیمر کمتر از ۵ متر یا مساوی ۵ متر است؟ اگر شرط درست باشد، کدهای داخل بلوک `if` اجرا خواهد شد

در دستور `if` بررسی می شود که یکبار عمل `if` انجام شود به شرطی که `isOnce` درست باشد

```
if(isOnce)
```

```
{
```

مقدار صفر به تابع `SetMaxAcceleration` بر اساس متغیر `m_pPathfindingComponent` اختصاص می یابد و دشمن به طرف پلیمر حرکت نخواهد کرد و خواهد ایستاد

```
m_pPathfindingComponent->SetMaxAcceleration(۰);
```

یک ثابت با نام `desiredFragmentId` تعریف شده که بررسی می کند که آیا متغیر `m_pCharacterController` در حال حرکت (در حال قدم زدن) است؟ که اگر در حال حرکت باشد با توجه به متغیر `m_walkTagId` و اگر در حال حرکت نباشد و در یک مکان ایستاده باشد با توجه به متغیر `m_IdleTagId`، انیمیشن با `FragmentId` مربوط به شرط در متغیر `desiredFragmentId` قرار خواهد گرفت

```
const auto& desiredFragmentId =
m_pCharacterController->IsWalking() ?
m_walkTagId : m_IdleTagId;
```

از آنجایی که متغیرهای انیمیشنی با `Id` مربوط
`desiredFragmentId` و `m_activeFragmentId` مخالف
 هستند، پس انیمیشن جاری همان انیمیشن `desiredFragmentId`
 خواهد بود

```
m_pAnimationComponent-  
>QueueFragmentWithId(m_activeFragmentId);
```

با توجه به متغیر کامبونتی `m_pAnimationComponent`، انیمیشن
 جاری در متغیر `m_activeFragmentId` در صف پخش شدن
 انیمیشن قرار خواهد گرفت، قرارگیری پخش انیمیشن ها در صف با تابع
`QueueFragmentWithId` انجام می شود
 مقدار متغیر `isOnce` به `false` تغییر می کند

```
isOnce=false;  
}
```

دستور `if` زیر کدهای استحکام برای جلوگیری از خطا است و بررسی می
 کند که آیا کامبونت دوربین برای پلیر وجود دارد یا نه؟

```
if (gEnv->pEntitySystem-  
>FindEntityByName("Player")-  
>GetComponent<Cry::DefaultComponents::CCameraC  
omponent>())  
{
```

با توجه به متغیر `direnemy`، دوربین پلیمر براساس تابع جستجو `entity` با نام `FindEntityByName` برای گرفتن کامپونت دوربین با تابع جنریک `GetComponent` انجام می شود.

```
auto direnemy= gEnv->pEntitySystem-
>FindEntityByName("Player")-
>GetComponent<Cry::DefaultComponents::CCameraC
omponent>()->GetCamera();
```

متغیر سه بعدی برداری `v`، مقدارش را براساس متغیر `direnemy` با گرفتن جهت دوربین پلیمر با برد ۱۰۰۰ متر در نظر میگیرد

```
Vec3 v=direnemy.GetViewdir()*۱۰۰۰;
```

```
v.z=۰;
```

مقدار محور `Z` را برابر صفر میگیریم

```
Quat newRotation = Quat::CreateRotationVDir(-
v);
```

اگر به درستی دقت کنیم، تابع `CreateRotationVDir` از کلاس `Quat` همان عمل `LookAt` یا همان `FaceAt` را انجام می دهد، به این معنا که جهت دشمن به طرف پلیمر است تا زمانی که پلیمر به دشمن نگاه می کند، در غیر اینصورت این عمل لغو می شود، منفی شدن متغیر `v` همان نگاه دشمن به طرف پلیمر است تا زمانی که پلیمر به دشمن نگاه می کند، با استفاده از فلوگراف و نود `FaceAt` این مشکل را می توان به سادگی حل کرد، در اینجا مشخص می شود که دلایل محبوبیت فلوگراف در بین بعضی از برنامه نویسان کرای انجین بی دلیل نیست

```
m_pEntity->SetPosRotScale(m_pEntity-
>GetWorldPos(), newRotation, Vec3(۱, ۱, ۱));
```

وقتی که دشمن به فاصله کمتر یا مساوی ۵ متر از پلیر رسید، به پلیر نگاه می کند و به طرف پلیر نشانه می رود و شلیک می کند، موقعیت دشمن با تابع `GetWorldPos`، چرخش دشمن براساس متغیر `newRotation` و مقیاس مربعی به یک اندازه با تابع `Vec3` پیش نیاز است برای تابع `SetPosRotScale` تا عمل محاسبه و اجرا می شود

```
}
}
}
```

در اینجا بلوک `else` نشان می دهد که فاصله دشمن از پلیر بزرگتر از ۵ متر است، پس بلوک `else` بر این اساس اجرا خواهد شد

```
else
{
```

یک ثابت با نام `desiredFragmentId` تعریف شده که بررسی می کند که آیا متغیر `m_pCharacterController` در حال حرکت (در حال قدم زدن) است؟ که اگر در حال حرکت باشد با توجه به متغیر `m_walkTagId` و اگر در حال حرکت نباشد و در یک مکان ایستاده باشد با توجه به متغیر `m_IdleTagId`، انیمیشن با `FragmentId` مربوط به شرط در متغیر `desiredFragmentId` قرار خواهد گرفت

```
const auto& desiredFragmentId =
m_pCharacterController->IsWalking() ?
m_walkTagId : m_IdleTagId;
```



```
if (m_activeFragmentId != desiredFragmentId)
{
```

بلوک `if` بررسی می کند که اگر انیمیشن در حال پخش شدن در متغیر `desiredFragmentId` مخالف انیمیشن در حال پخش شدن `m_activeFragmentId` فعلی باشد، آنگاه خطوط زیر اجرا شوند

```
m_activeFragmentId = desiredFragmentId;
```

از آنجایی که متغیرهای انیمیشنی با `Id` مربوط `desiredFragmentId` و `m_activeFragmentId` مخالف هستند، پس انیمیشن جاری همان انیمیشن `desiredFragmentId` خواهد بود.

```
Quat newRotation =
Quat::CreateRotationVDir(m_pCharacterController-
->GetMoveDirection());
```

با توجه به متغیر `newRotation`، هنگامی که دشمن فاصله اش بزرگتر از ۵ متر است، عمل `LookAt` یا `FaceAt` به طرف پلیر انجام می دهد، به این معنا که دشمن به پلیر نگاه می کند و این عمل به وسیله تابع `CreateRotationVDir` با توجه به متغیر کامبونی `m_pCharacterController` از تابع `GetMoveDirection` انجام می شود، تابع `GetMoveDirection` همان تعیین جهت دشمن به طرف پلیر را محاسبه می کند و در داخل تابع `CreateRotationVDir` عمل تعیین جهت دشمن به طرف پلیر اجرا می شود

```
m_pEntity->SetPosRotScale(m_pEntity-
->GetWorldPos(), newRotation, Vec3(۱, ۱, ۱));
```

براساس تابع `SetPosRotScale`، موقعیت دشمن، چرخش دشمن به طرف پلیر و مقیاس دشمن مقدار دهی می شود

```
m_pAnimationComponent-
>QueueFragmentWithId(m_activeFragmentId);
```

با توجه به متغیر کامبوتی `m_pAnimationComponent`، انیمیشن جاری در متغیر `m_activeFragmentId` در صف پخش شدن انیمیشن قرار خواهد گرفت، قرارگیری پخش انیمیشن ها در صف با تابع `QueueFragmentWithId` انجام می شود

```
m_pAnimationComponent->ResetCharacter();
```

سپس تابع `ResetCharacter`، عمل صف بندی انیمیشن ها را برای کاراکتر مقدار دهی اولیه و مقدار دهی مجدد می کند و همه چیز از صفر برای پخش شدن انیمیشن ها آغاز می شود

```
}
```

```
m_pPathfindingComponent-
>SetMaxAcceleration(۶);
```

با توجه به تابع `SetMaxAcceleration` با مقدار ۶ از متغیر کامبوتی کوتاه ترین مسیر به طرف هدف، حرکت دشمن به طرف پلیر مجدداً آغاز می شود

```
isOnce = true;
```

متغیر `isOnce` مقدارش `true` می شود، این تضمین ایجاد می شود که انیمیشن های دشمن در هنگام متوقف شدن دشمن و حرکت مجدد دشمن به طرف پلیر درست پخش شود

```
}
}
}
```

حالا به پرتاب گلوله های دشمن به طرف پلیر، به صورت خلاصه و مفید در هدر فایل `BulletEnemy.h` می پردازم

```
#pragma once
```

```
class CBulletEnemyComponent final : public
IEntityComponent
{
```

کلاسی نهایی با نام `CBulletEnemyComponent` تعریف کردم که از رابط `IEntityComponent` ارث بری می کند

```
public:
virtual ~CBulletEnemyComponent() {}
```

یک تخریب کننده برای کلاس تعریف می کنیم که بعدا بتوانیم منابعی که از حافظه، `CPU` و `GPU` گرفته ایم، بازیس گیری کنیم

```
virtual void Initialize() override
{
```

تابع سیستمی `Initialize` کارهای زیر را انجام می دهد

```
const int geometrySlot = ۰;
```

یک ثابت با نام `geometrySlot` را با مقدار صفر در نظر می گیریم

```
m_pEntity->LoadGeometry(geometrySlot,
"Objects/Default/primitive_sphere.cgf");
```

شی جاری، که همان گلوله دشمن است، یک مدل هندسی را با استفاده از `LoadGeometry` بارگذاری می کند

```
auto *pBulletMaterial = gEnv->pEngine-
>GetMaterialManager()-
>LoadMaterial("Materials/bullet");
```

یک متغیر با نام `pBulletMaterial` تعریف می کنیم که یک متریال را با استفاده از تابع `LoadMaterial` لود می کند و اگر نمی دانید نوع `Data Type` متغیرتان چیست می توانید از کلمه `auto` استفاده کنید، سعی کنید که کمتر از این کلمه استفاده کنید و متغیرها را با نوع `Data Type` اعلان کنید، اما مواقعی پیش می آید که مجبورید از کلمه `auto` استفاده کنید

```
m_pEntity->SetMaterial(pBulletMaterial);
```

حالا با توجه به متغیر `pBulletMaterial` و وجود تابع `SetMaterial`، متریال لود شده برروی مدل گلوله نگاشت می شود و برروی مدل قرار می گیرد.

```
SEntityPhysicalizeParams physParams;
```

متغیر `physParams` از نوع دیتا تایپ `SEntityPhysicalizeParams` است که وظیفه آن به کارگیری پارامترها و فیلدهای فیزیک در گلوله های دشمن است

```
physParams.type = PE_RIGID;
```

فیلد `type` از متغیر `physParams`، ثابت `PE_RIGID` را میگیرد که به مفهوم `Rigidbody` اشاره دارد.

```
physParams.mass = ۲۰۰۰۰۰.f;
```

فیلد `mass` از متغیر `physParams` به جرم شی (گلوله) اشاره می کند و حامل ۲۰۰۰۰۰ گرم یا همان ۲۰۰ کیلوگرم است، البته این عدد اغراق آمیز است اما برای دیدن حرکت گلوله در بازی و در اینجا برای بحث آموزشی بسیار کاربرد دارد.

```
m_Entity->Physicalize(physParams);
```

با تابع `Physicalize` که عمل انجام فیزیک را براساس پارامترهای متغیر `physParams` برای گلوله لحاظ می کند

```
GetEntity()->SetViewDistRatio(۲۵۵);
```

تابع `SetViewDistRatio` برای شی جاری یا همان گلوله محدوده و دیدی را تا فاصله ۲۵۵ متری در نظر میگیرد تا گلوله در این محدوده دیده شود.

```
if (auto *pPhysics = GetEntity()->GetPhysics())
{
```

با توجه به بلوک `if` و متغیر اتوماتیک تخصیص حافظه اشاره گری با نام `pPhysics` از گلوله، قوانین فیزیک را دریافت می کند.

```
pe_action_impulse impulseAction;
```

متغیر `impulseAction` از نوع ساختار `pe_action_impulse` است که باید عمل ضربه زدن را طبق قوانین فیزیک انجام دهد

```
const float initialVelocity = ۱۰۰۰۰.f;
```

متغیر بدون تغییر (ثابت) `initialVelocity` با مقدار ۱۰ هزار نیوتن (۱۰ کیلونیوتن) مقدار دهی می شود

```
impulseAction.impulse = GetEntity()-
>GetWorldRotation().GetColumn\() *
initialVelocity;
```

حالا باید جهت حرکت گلوله را طبق فیلد `impulse` در متغیر `impulseAction` مقداردهی کنیم، با استفاده از تابع `GetWorldRotation` و `GetColumn\` جهت حرکت گلوله را بدست می آوریم که در اینجا تابع `GetColumn\` در واقع محور Y یا همان محور عمق است و رو به جلو است و مقدار نیروی نیوتنی که در متغیر `initialVelocity` است در تابع `GetColumn\` ضرب می شود و این باعث ضربه زدن و حرکت شتابی گلوله به جهت جلو خواهد بود اما هنوز ضربه وارد نشده است و گلوله در یک مکان از فضای سه بعدی (کنار دهانه اسلحه) تولید می شود.

```
pPhysics->Action(&impulseAction);
```

با تابع `Action` از متغیر `pPhysics` عمل ضربه با توجه به وجود متغیر مقداردهی شده `impulseAction` انجام می شود و گلوله به طرف رو به جلو و به طرف عمق حرکت می کند

```
}
}
```

```
static void
ReflectType(Schematyc::CtypeDesc<CBulletEnemyC
omponent>& desc)
{
desc.SetGUID("{B۵۳A۹A۵F-F۲۷A-۴۲CB-۸۲C۷-
B۱E۳۷۹C۴۱A۲A}"_cry_guid);
}
```

در اینجا تابع ReflectType کلاس CBulletEnemyComponent را تنها در بازی انعکاس می دهد و گلوله دشمن در ادیتور کرای انجین قابل دسترس نیست و تنها در بازی قابل استفاده است، در رابطه با تابع ایستا ReflectType توضیحات مفصلی ارائه شده است.

```
virtual uint۶۴ GetEventMask() const override {
return BIT۶۴(ENTITY_EVENT_COLLISION); }
```

تابع GetEventMask و تابع ProcessEvent چندین بار بررسی شده است اما به صورت خلاصه، این دو تابع مکمل عمل برخورد گلوله را آشکار می کنند و هنگامی که گلوله به سطح یا شی سفت و سختی برخورد می کند، دستور یا دستوراتی زیر اجرا می شوند.

```
virtual void ProcessEvent(SEntityEvent& event)
override
{

if (event.event == ENTITY_EVENT_COLLISION)
{
```

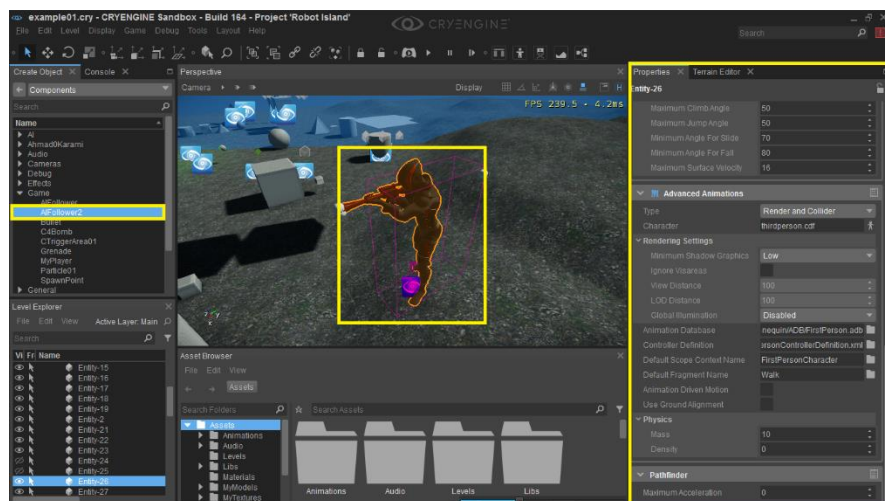
```
gEnv->pEntitySystem-
>RemoveEntity(GetEntityId());
```

در این بلوک **if**، بررسی می شود که اگر گلوله به شی سخت و سفتی برخورد کرد، گلوله با استفاده از تابع **RemoveEntity** حذف شود که این تابع نیز به آرگومان **GetEntityId** نیاز دارد که عمل حذف شدن را بر اساس **Id** گلوله انجام می شود.

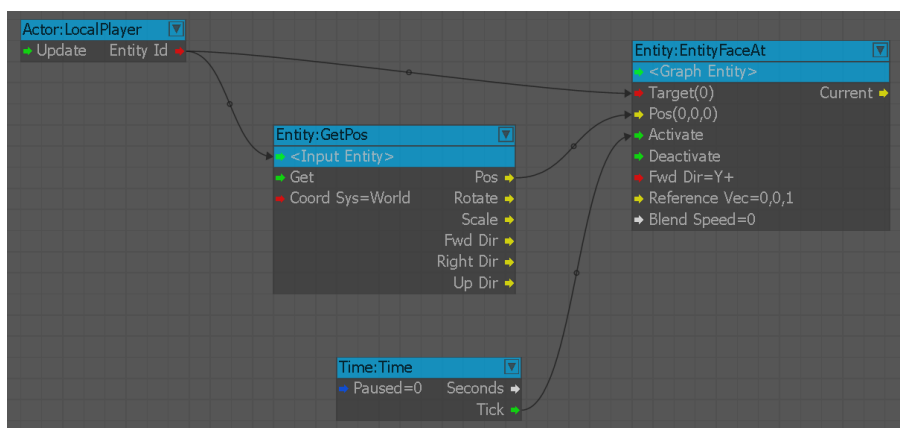
```
}
}
};
```

دقت کنید از آنجایی که ما از **BulletEnemy.cpp** استفاده نکردیم و تنها از هدر فایل **BulletEnemy.h** استفاده کردیم، پس کامپوننت **BulletEnemy** در سیستماتیک ادیتور و در پنجره **Create Object** در بخش **Components** نمایش داده نخواهد شد و تنها در بازی قابل دسترس خواهد بود و هدر فایل **BulletEnemy.h** را به **CMakeLists.txt** اضافه نمایید و حالا پروژه را در ویژوال استودیو **Build** کنید

نکته آخر، این است که برای افزایش و دقت ۱۰۰٪ نگاه دشمن به پلیر و قفل کردن این نگاه در هر فاصله ای و در هر مکانی از مرحله می توان از فلوگراف برای دشمن استفاده نمود (البته می توان محدوده ای را نیز برای نگاه دشمن به پلیر و قفل کردن این نگاه مشخص کرد) شما دشمن را در سندباکس و داخل مرحله **Drag & Drop** کرده اید.



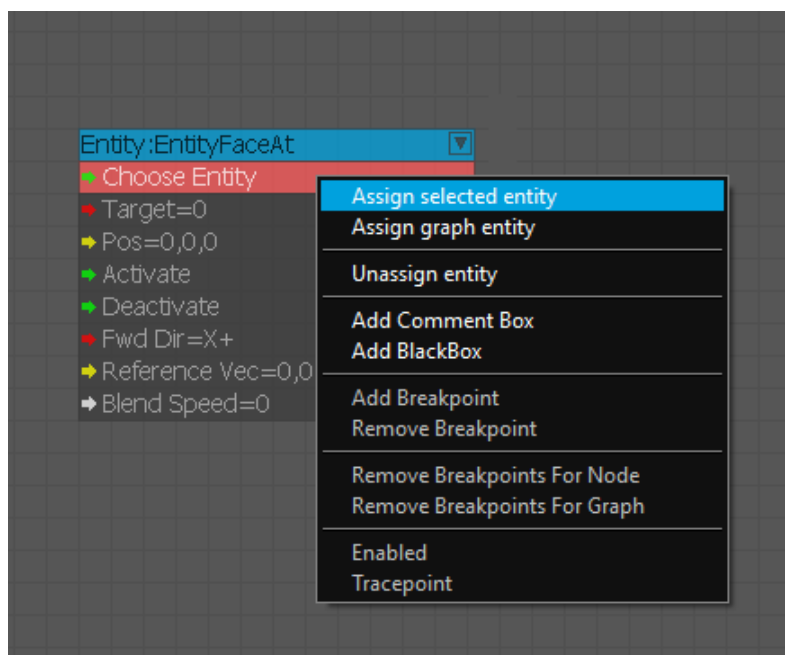
حالا دشمن را انتخاب کنید و به پنجره Properties مراجعه کنید و در بخش Flowgraph برروی دکمه Open کلیک کنید و یک کادر نمایش داده می شود و برروی دکمه OK کلیک کنید و طبق شکل زیر این نودها را اضافه کنید و این نودها را به هم وصل کنید.

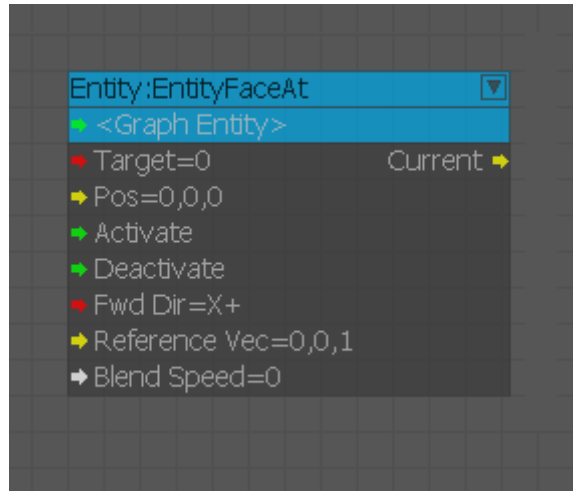


در واقع نود EntityFaceAt همان عملکرد LookAt را دارد و به شما کمک می کند براساس ID پلیر، دشمن به پلیر نگاه کند و می بینید که پارامتر ورودی Target به Local Player براساس خروجی Entity Id متصل شده است، پارامتر ورودی Pos همان موقعیت پلیر را براساس نود GetPos از

پارامتر خروجی Pos دریافت می کند و پارامتر ورودی Activate باید براساس نود Time با تیک های زمانی (برش های زمانی) از پارامتر خروجی Tick به روز رسانی شود.

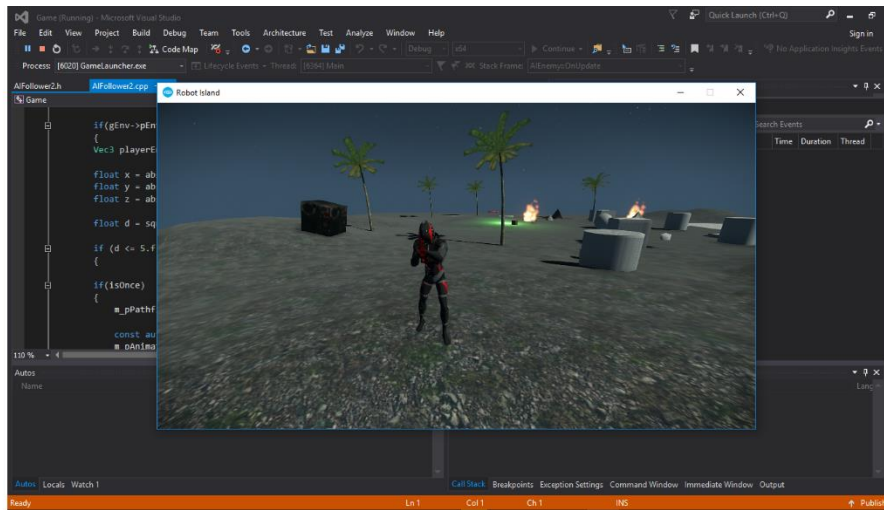
فراموش نکنید که در نود EntityFaceAt، اولین ورودی با نام Choose Entity وجود دارد و این ورودی را براساس محتوای Graph Entity لحاظ کنید، به این معنا که دشمن را در سندباکس انتخاب کنید، بر روی اولین ورودی از نود EntityFaceAt، راست کلیک کرده و گزینه Assign Select Entity را انتخاب کنید تا به <Graph Entity> تغییر نام داده شود، به دو شکل بعدی دقت کنید





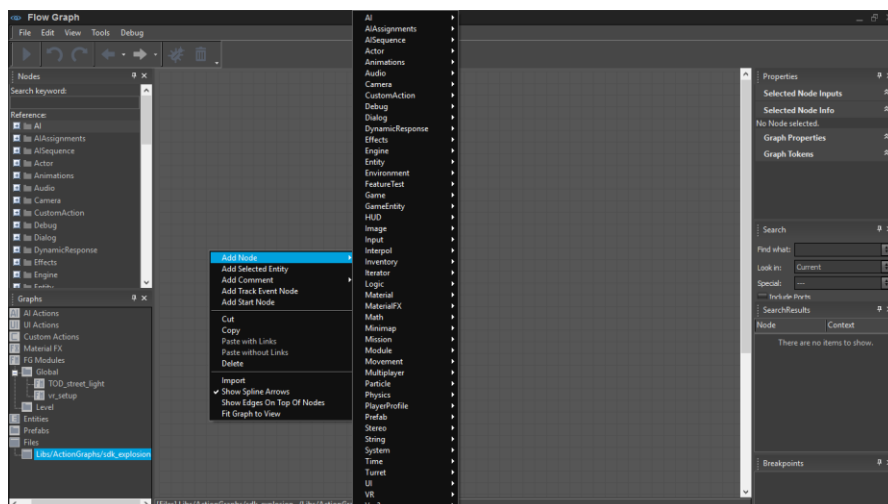
طبق تصویر می بینید که پارامتر ورودی اول نود EntityFaceAt از Choose Entity به <Graph Entity> تغییر کرده است

حالا فلوگراف را ذخیره کنید و همچنین مرحله را ذخیره کنید و حتما بعد از ذخیره مرحله به منوی File مراجعه کرده و گزینه Export to Engine را انتخاب کنید یا کلید F7 را برروی صفحه کلید بزنید تا مرحله و دیگر Assets ها برای ویزوال استودیو مهیا شود و لود مرحله در ویزوال استودیو انجام شود و در نهایت به ویزوال استودیو برگردید و پروژه را Rebuild کنید و بازی را اجرا کنید.



فصل دهم

کاربردهای Flowgraph در جلوه های
تصویری دوربین و ساخت منوهای بازی



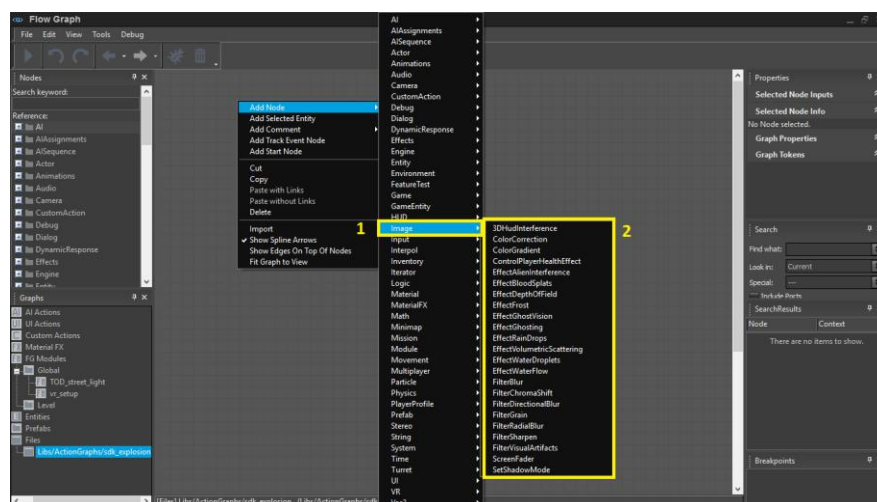
با ظهور سیماتیک فلوگراف کم کم جزء میراث کرای انجین ۳ حساب می شود، در تصویر بالا لیست طولانی از نودهای فلوگراف را می بینید، این نودها بیشترین وابستگی را به هسته انجین دارند و نمی توان برای ساخت یک بازی مستقل روی این نودها حساب ویژه ای را کرد، در واقع این نودها برای Entity هایی با ساختار مشخص مثل آنچه که در پروژه GameSDK دیدید کاربردی و قابل استفاده است و برای یادگیری بخشی از مفاهیم در کرای انجین مفید است اما وابستگی این نودها به شدت به زبان C++ بر میگردد و توصیه میکنم که کمتر به سراغ فلوگراف بروید و تمرکزتان بر روی C++ باشد، امیدوارم که در نسخه های بعدی کرای انجین فلوگراف حذف شود یا برای ساخت منطق در سطح Level ها مستقل از C++ شود، این امیدواری بیشترین نفع را برای کاربرانی دارد که دوست دارند بدون کدنویسی بازی هایشان را بسازند، سیماتیک و فلوگراف مستقل در آینده قدرتمندتر خواهند شد اما هنوز C++ حرف اول را می زند چرا که بازی هایی که با C++ نوشته می شوند سرعت اجرا بسیار بالاتر و مصرف کمتر حافظه را به دنبال دارند، اگر در پروژه GameSDK بر روی فلوگراف و سیماتیک در نسخه کرای انجین ۵،۴ کار کنید، به خوبی درک خواهید کرد که فلوگراف و سیماتیک برای ساخت بازی

نسبتاً کندتر از زبان C++ هستند، در اینجا تمجید از زبان C++ نیست، بلکه می‌خواهم این نکته را فراموش نکنید که موتور بازی سازی کرای انجین از کدهای C++ ایجاد شده است، اگرچه هنوز نیز از فلوگراف استفاده می‌شود و از فلوگراف برای دو بخش مهم و حیاتی بازی استفاده می‌شود، این دو بخش به شرح زیر است:

۱- Post Processing Effect

جلوه‌های ویژه در دوربین برای زیبا تر شدن بازی جزء عنصر لاینفک بازی محسوب می‌شود و بازی بدون پست پراسسینگ افکت یا امیج افکت (Image Effect) از جذابیت آن به شدت کاسته می‌شود و این مسئله در کرای انجین بیشتر تاثیر خود را نشان می‌دهد، برای کار کردن با جلوه‌های تصویری در این مبحث من به سراغ تک تک این جلوه‌ها رفته و آن جلوه‌ها را شرح خواهم داد، با اطمینان نمی‌توان گفت اما در آینده ممکن است این جلوه‌ها توسط سیماتیک مورد دسترسی واقع شود، فراموش نکنید که محور اصلی این کتاب زبان C++ است اما برای آشنایی شما این مبحث را تشریح کرده‌ام، در شکل زیر می‌بیند که دو کادر زرد رنگ با شماره ۱ و شماره ۲ مشخص کرده‌ام که در شماره ۱ کاتالوگ Image و در شماره ۲ تعداد نودهایی که در ارتباط با جلوه‌های تصویری است جدا سازی شده است، که شرح هر یک از این نودها و خروجی هر نود و تاثیر آن در بازی به شرح زیر است، تعداد پارامترها بسیار زیاد است و من هر نود را بر اساس پارامتر دلخواه خود تنظیم می‌کنم و خروجی را به نمایش می‌گذارم، **کل نودهای جلوه‌های تصویری ۲۳ نود با پارامترهای مختلف است** و نیاز شما را برای ساخت بازی‌های منحصر به فرد با جلوه‌های تصویری فوق العاده بر آورده می‌کند، مطمئناً ترکیب نمودن این جلوه‌ها زیبایی بازی را صدچندان خواهد کرد، این جلوه‌ها در پروژه GameSDK تست شده است و برای پروژه‌های

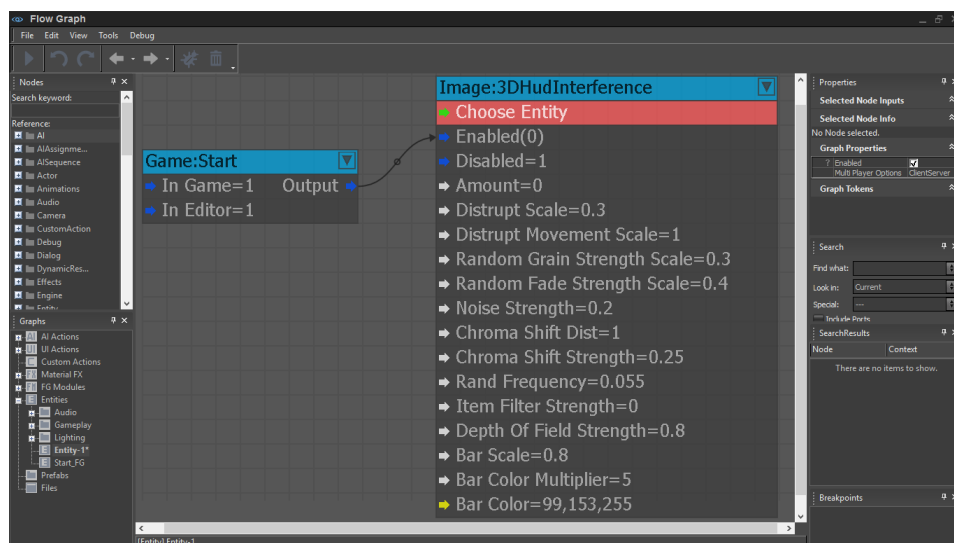
دیگر نیز عمل خواهد کرد، من مقادیر زیاد به پارامترها اختصاص داده ام تا نتایج به خوبی نشان داده شود اما شما در بازی تان نباید از مقادیر زیاد استفاده کنید زیرا که نتیجه عکس را خواهد داشت، شما با تمرین و تکرار در هر نود با توجه به توضیحاتم در این فصل می توانید به نتایج زیبا و ایده آل خودتان برسید



۳DHudInterference: یکی از کاربردهای این جلوه تصویری پارازیت انداختن روی صفحه رادار و میزان مهمات پلیمر است، مثلاً وقتی که به ناحیه ای از دستگاه های ناشناخته و سری دشمن می رسم رادار و میزان مهمات پلیمر دچار پارازیت شدید شود و قابل نمایش نباشد.



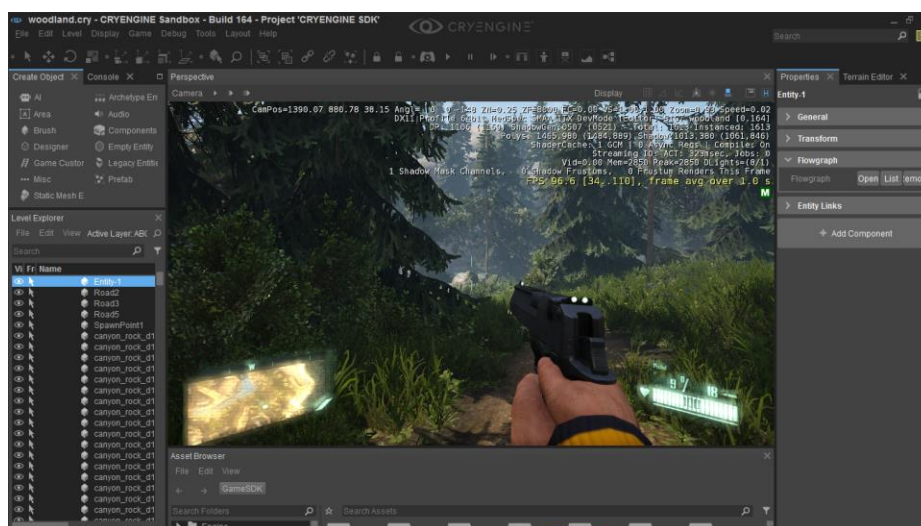
بعد از اینکه پارامتر Amount به مقدار ۱۰۰۰ افزایش می‌دهیم نتیجه به صورت زیر خواهد شد



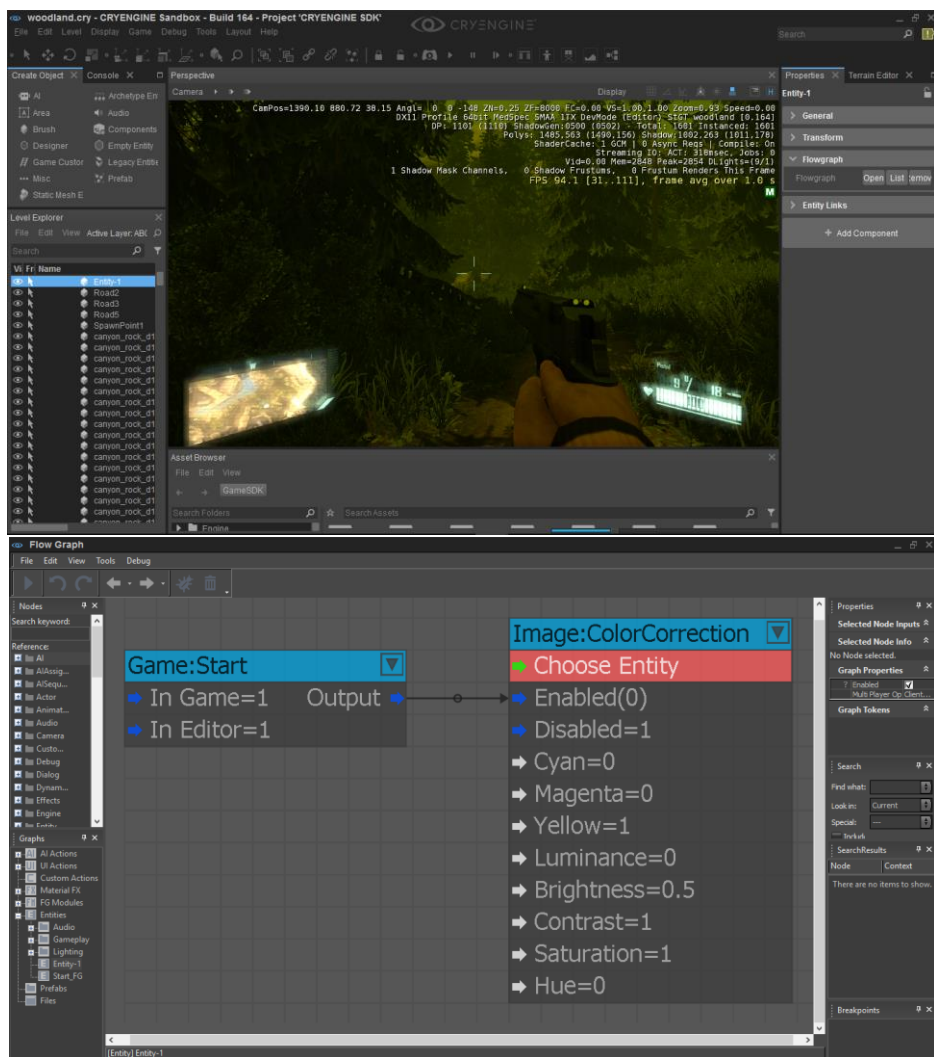
شما می‌توانید پارامترهای دیگر را نیز تغییر دهید و نتیجه را متفاوت تر مشاهده نمایید، تغییرات مختلف بر روی پارامترها اعمال نمایید تا بیشترین تلاش را برای کسب جلوه تصویری بهتر بدست آورید.

ColorCorrection: برای تنظیم رنگ‌های کل محیط استفاده می‌شود، معمولاً این رنگ‌ها براساس فضای بازی، نوع مرحله، میزان شدت خشونت و غیره تغییر می‌کند، شما باید تناسب استفاده از این نود را براساس سناریو

بازی انجام دهید، در محیط‌های مختلف باید جلوه‌های متفاوت را نشان دهید، تجربه بهترین راه برای کسب نتایج صحیح است.



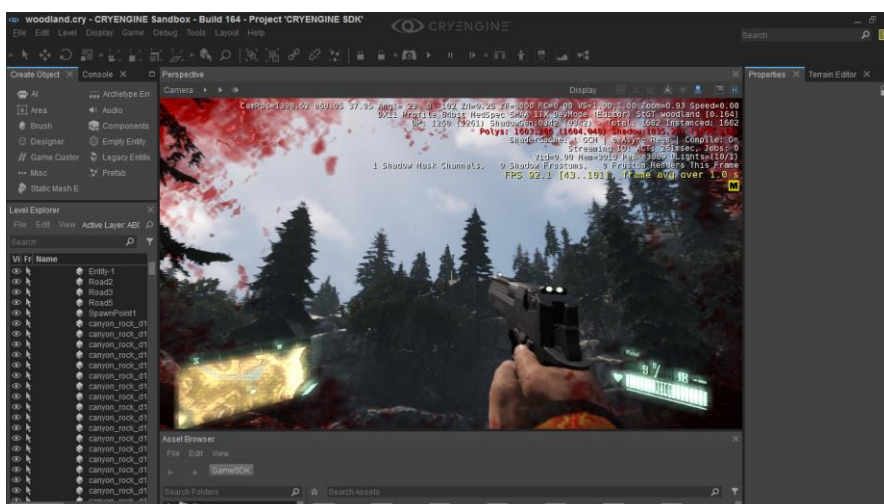
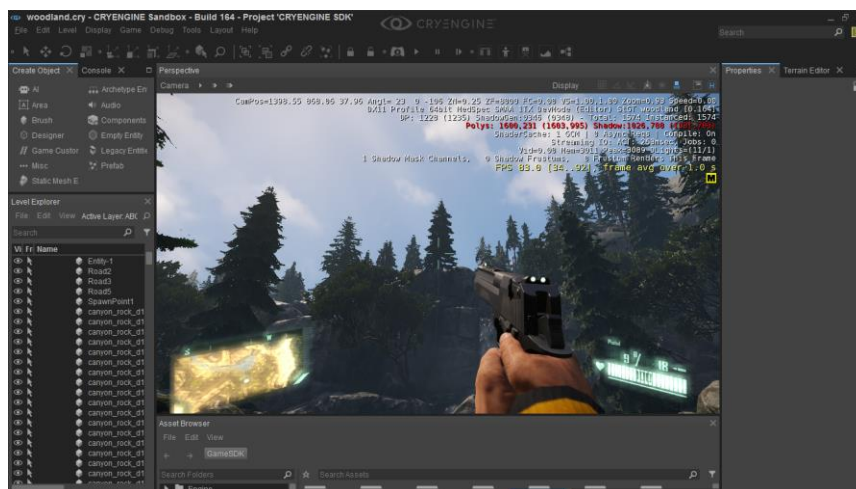
پارامترهای $Brightness=0.5$, $Yellow=1$ را مقدار دهی کنید و نتیجه به صورت زیر خواهد شد، رنگ زرد و میزان روشنایی صحنه بازی را به صورت کل تغییر خواهد داد، همچنین برای سیاه و سفید کردن بازی کفایت پارامتر Saturation را برابر صفر قرار دهید و برای بازگشت از تصاویر سیاه و سفید به رنگی همین پارامتر را به مقدار ۱ بازگردانید، سعی نکنید که مقدار پارامتر Saturation از ۱ بیشتر باشد چون رنگ‌ها شدت‌شان بیشتر می‌شود، حتما با پارامترهای مختلف بازی کنید تا نتیجه درست را کسب نمایید.

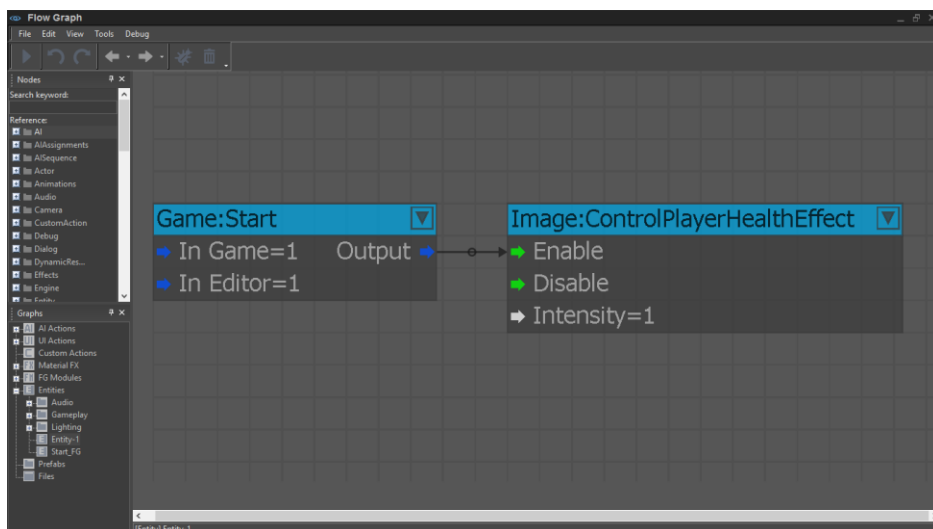


ColorGradient : برای نمایش تغییرات در محیط براساس طیف تصویر با استفاده از یک تکسچر عمل می کند، به نظر می رسد باگی در این نود وجود دارد و تغییرات قابل مشاهده نیست

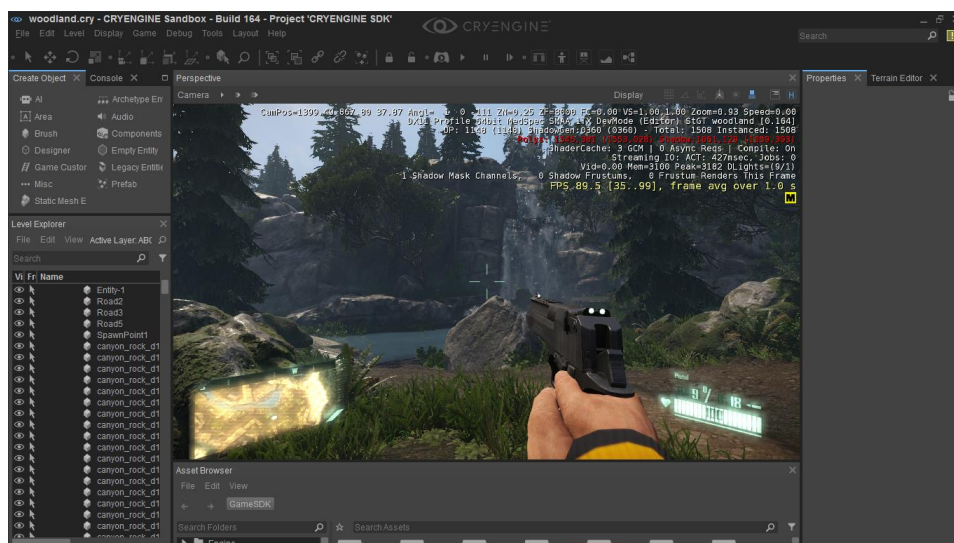
ControlPlayerHealthEffect : برای نمایش دادن این که پلیمر در حال از دست دادن خون است که اگر مقدار پارامتر Intensity برابر ۱ شود، این

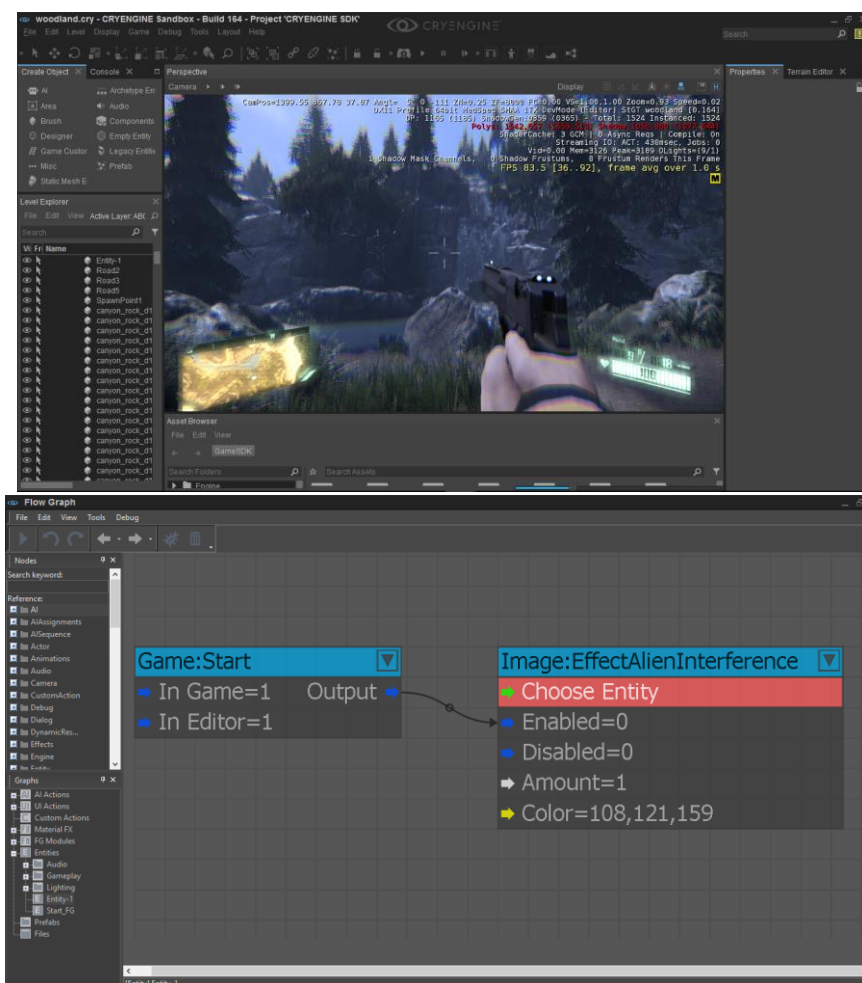
عمل اتفاق می افتد برای غیر فعال شدن از گیت disable استفاده کنید، به تصویر زیر و مقایسه آن در تصویر بعدی نگاهی بیندازید، این نود در پروژه GameSDK وجود دارد و در پروژه های دیگر قابل دسترس نیست.





EffectAlienInterference: برای ایجاد پارازیت بر روی دوربین پلیر و نمایشی ویژه از اختلال بر روی دوربین مقدار Amount را به یک تغییر دهید و تصویر زیر را مشاهده کنید

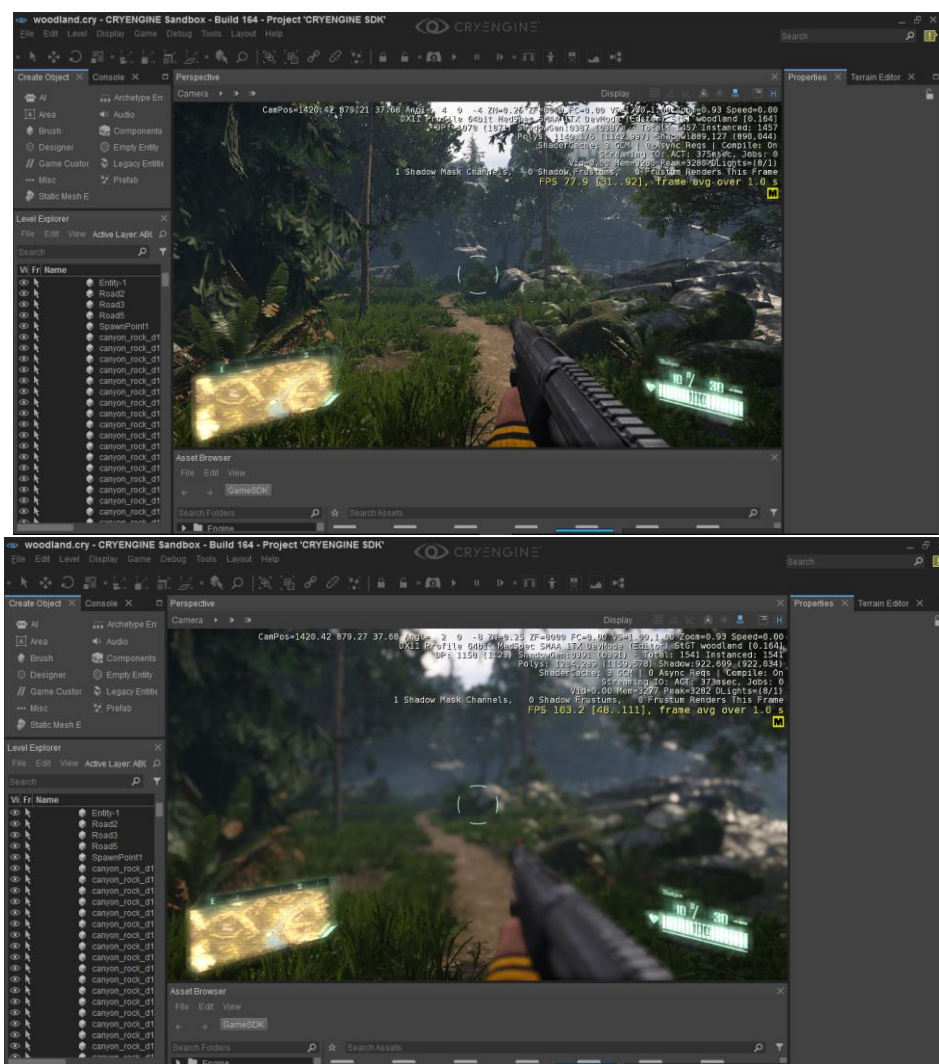


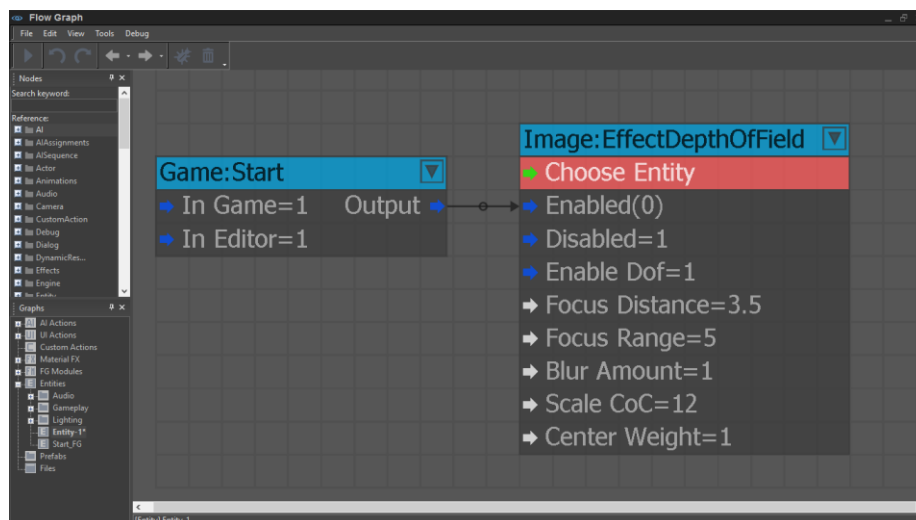


EffectBloodSplats : برای نشان دادن لکه های خون و پاشیدن آن
بر روی دوربین کاربرد دارد اما به نظر می رسد باگی در این نود وجود دارد و
تغییرات قابل مشاهده نیست

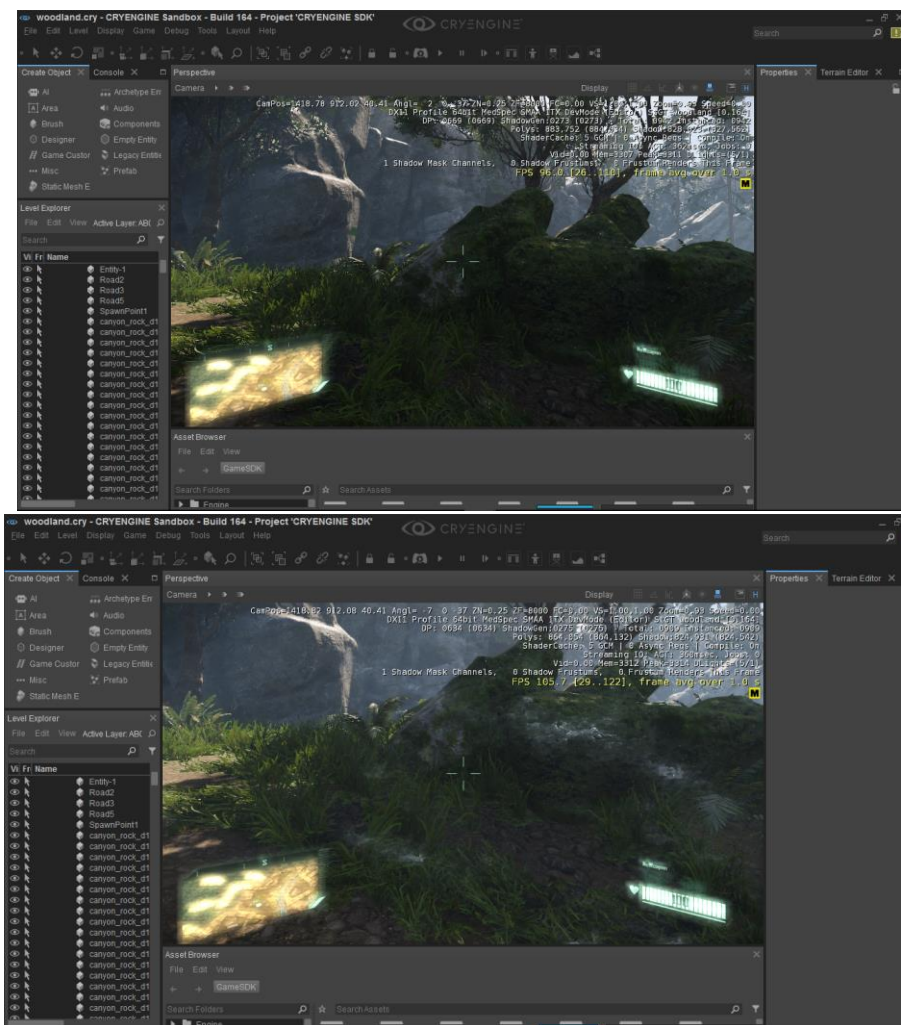
EffectDepthOfField : برای تاثیرگذاری بر عمق میدان دید دوربین در
پلیر و تار شدن دید دوربین از این نود استفاده می شود، شما می توانید
پارامترها را تغییر دهید اما در این مثال تنها پارامتر Enable Dof=true
است و نتیجه را در تصاویر زیر می بینید، به سمت رادار نگاهی بیندازید و

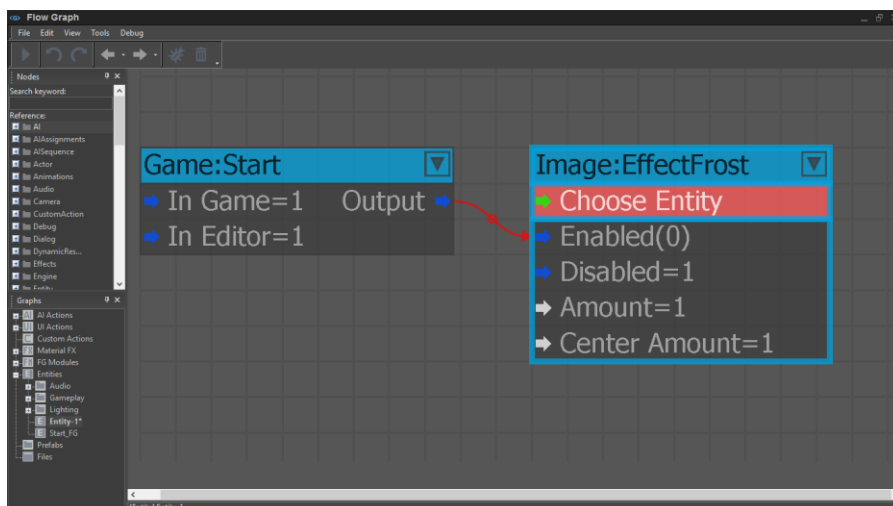
متوجه می شوید که تغییرات در آن قسمت لحاظ شده است، حتما پارامترها را در این نود تغییر دهید تا بهترین نتیجه را حاصل کنید





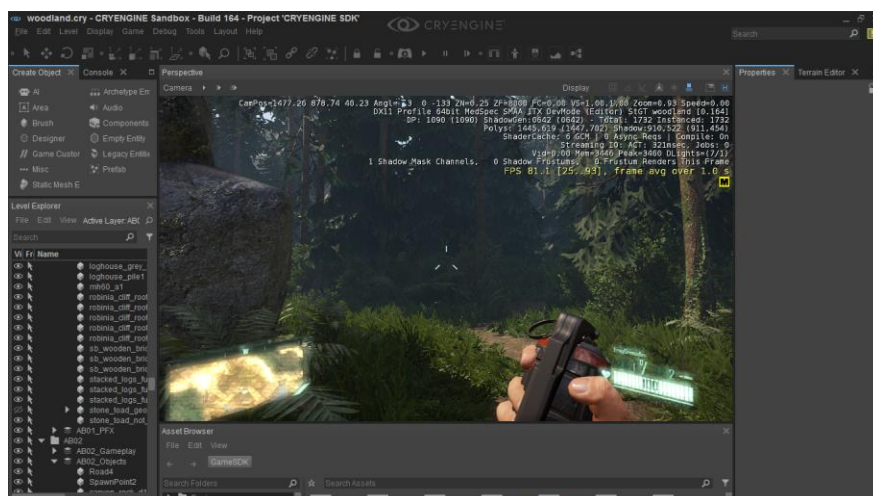
EffectFrost: برای نمایش لکه‌هایی سفید و نسبتاً شفاف بر روی دوربین پلیر استفاده می‌شود و مقدار **Amount** برابر ۱ شده است، مثلاً این جلوه می‌تواند برای موجودات ارگانیکی فضایی که به طرف شما ماده‌ای را انتشار و پخش می‌کنند می‌تواند مفید باشد.

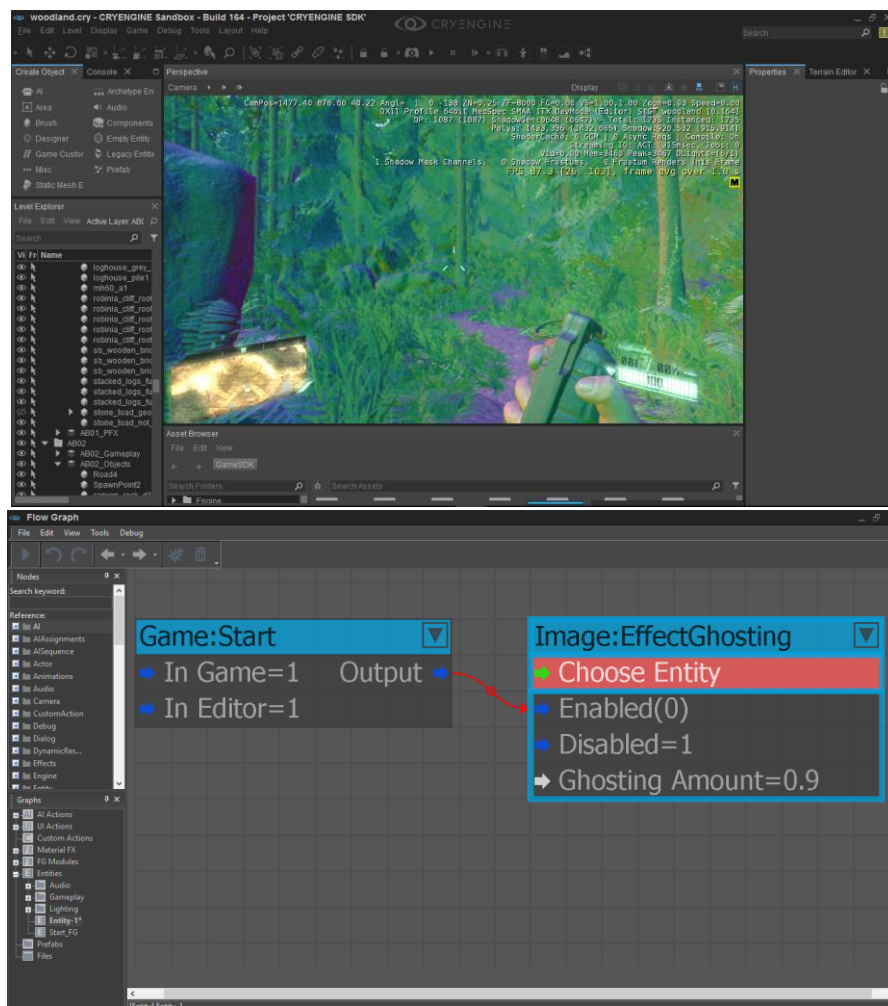




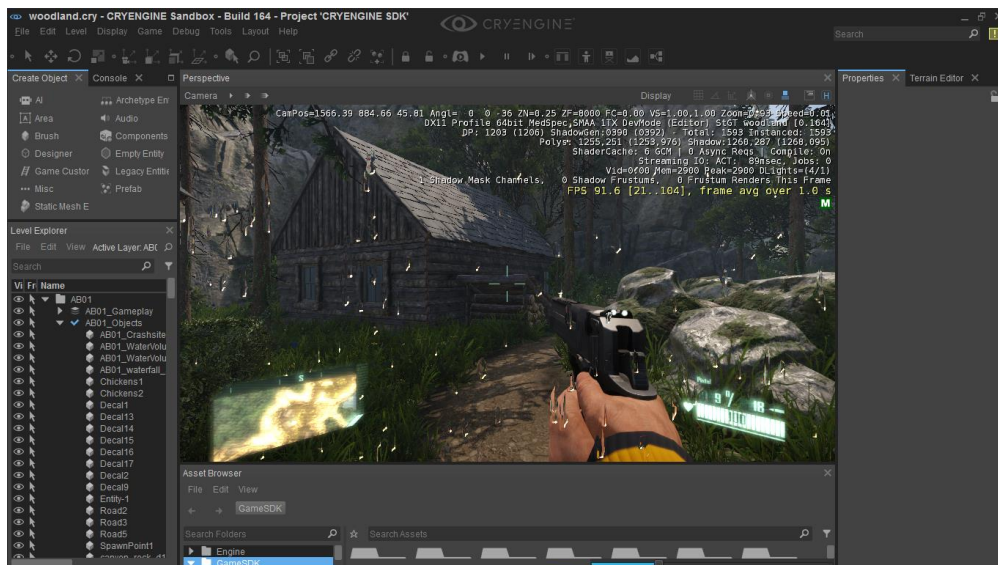
EffectGhostVision : به نظر می‌رسد باگی در این نود وجود دارد و عمل مورد نظر در این نود انجام نمی‌شود

EffectGhosting : برای نمایش دادن اینکه پلیمر مانند روح به محیط اطراف بازی نگاه کند، از این نود استفاده می‌شود، مقدار پارامتر Amount را برابر ۰٫۹ قرار دهید تا تغییرات را مثل تصاویر زیر ببینید.





EffectRainDrops : برای شبیه سازی ریزش باران یا عرق ریختن و خستگی پلیر در گرمای هوا استفاده می شود، طبق تصاویر زیر عمل کنید و پارامتر $Amount=1$ و پارامتر $Spawn\ Time\ Distance=0$ قرار دهید، هرچه مقدار پارامتر $Spawn\ Time\ Distance$ افزایش یابد عمل ریزش با تاخیر بیشتری انجام خواهد شد و قطرات کمتری بر روی دوربین جاری خواهد شد



EffectVolumetricScattering : تحقیقات بر روی این نود ادامه دارد و با

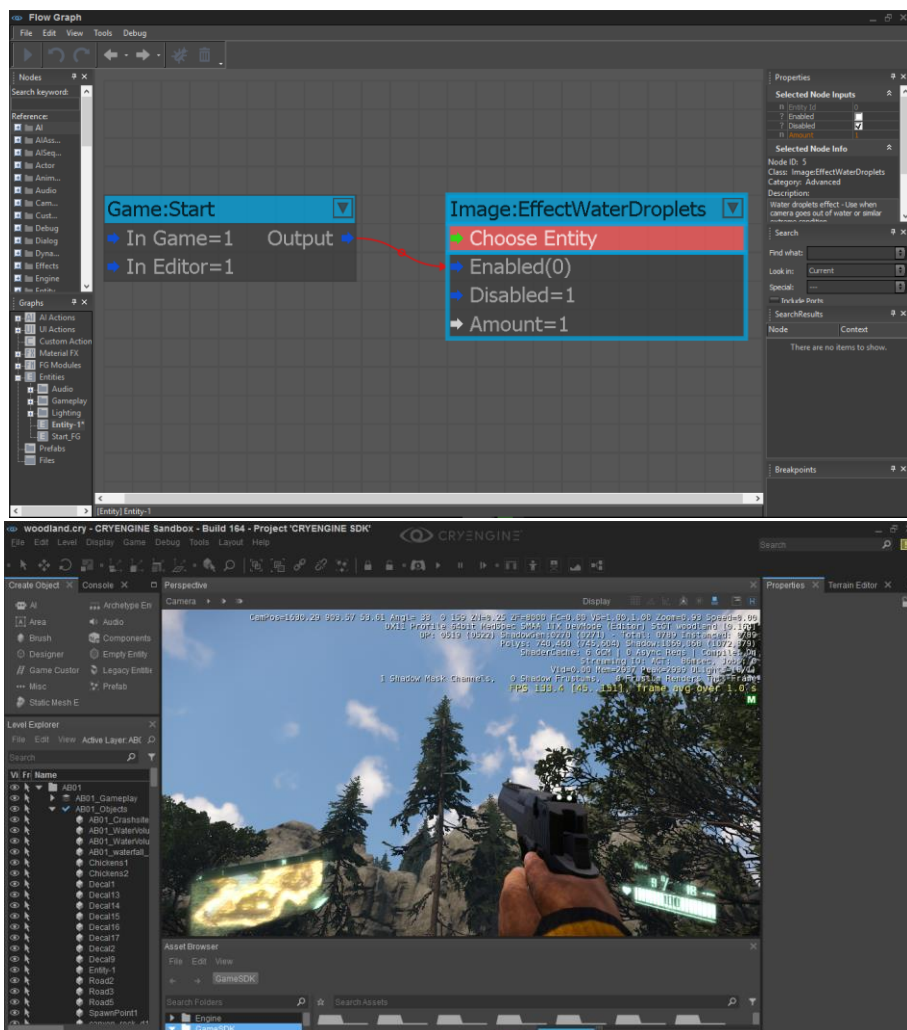
اعمال آن هیچ تغییری بر روی محیط انجام نمی شود

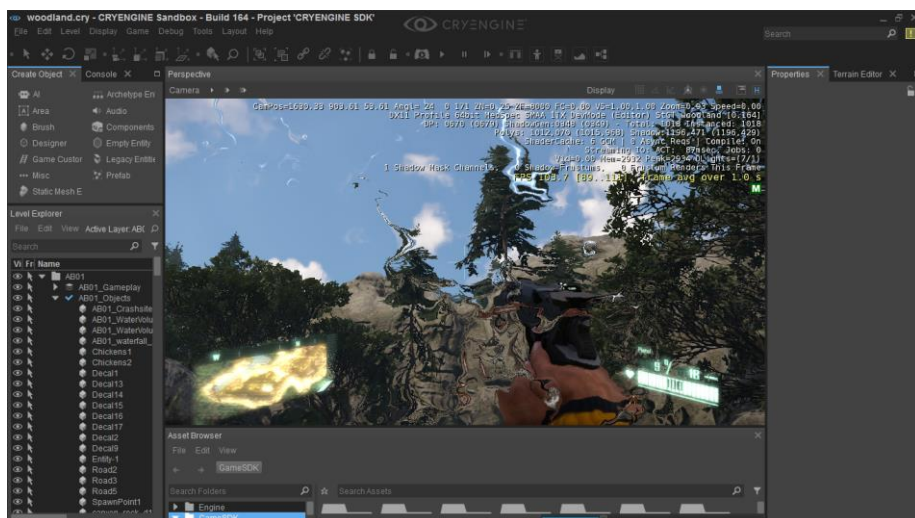
EffectWaterDroplets : وقتی به یک آبشار نزدیک می شویم یا آبی که

از بالا به پایین ریخته می شود دوربین پلیر را خیس می کند و بخشی از آب

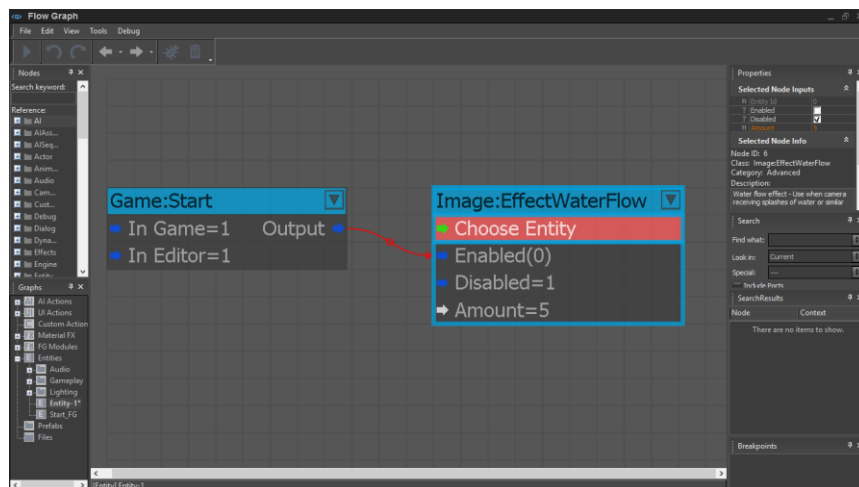
بر روی دوربین جاری می شود از این جلوه استفاده می کنیم، پارامتر

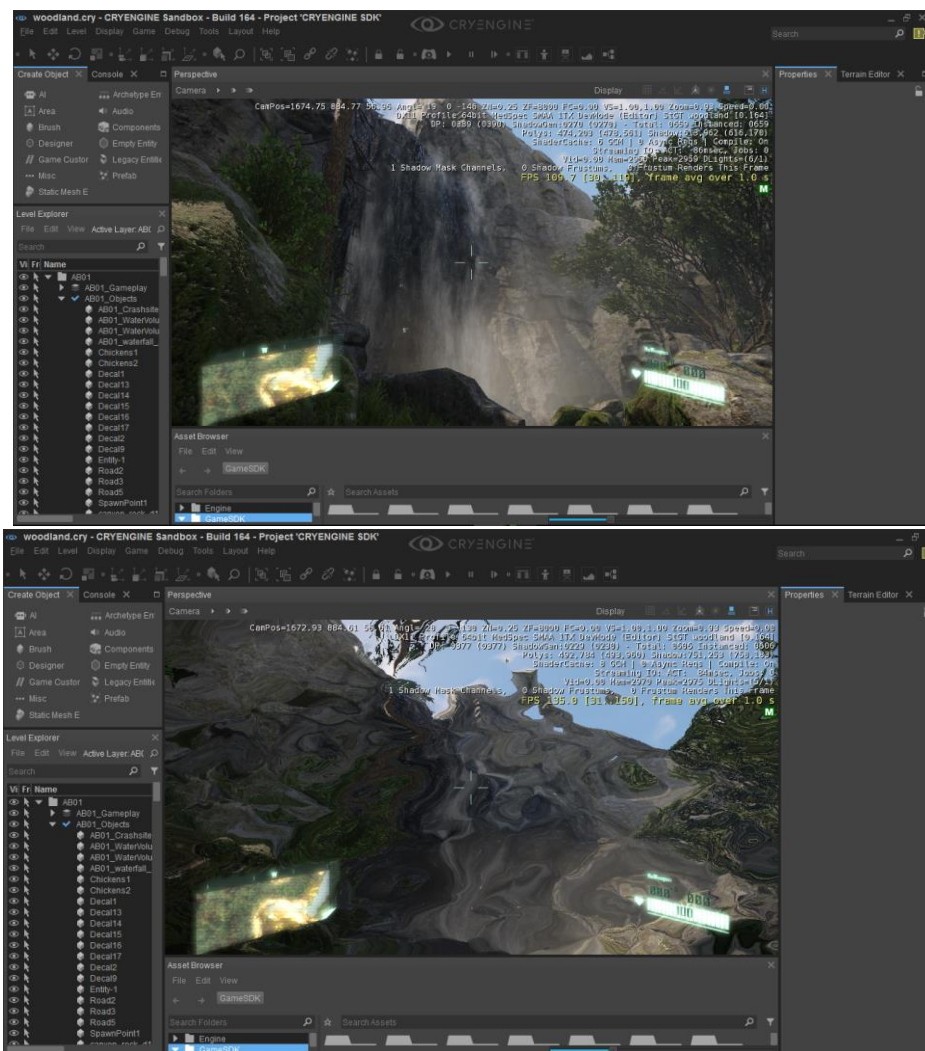
Amount=۱ باشد.



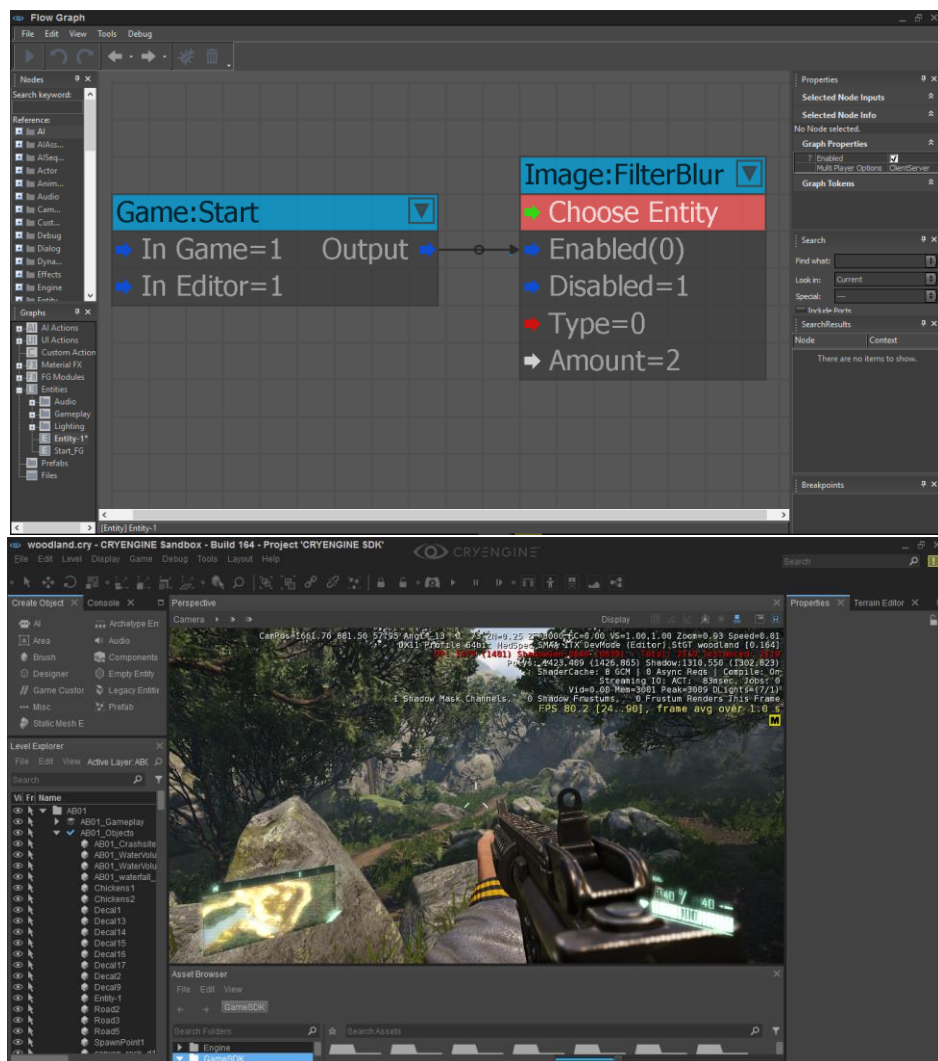


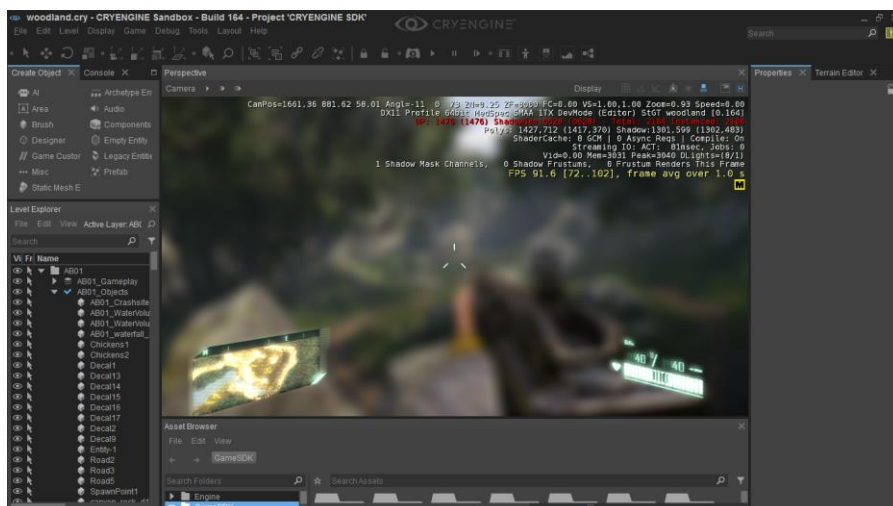
EffectWaterFlow: وقتی که می‌خواهیم پلیمر به داخل آب فرو رود و دوربین پلیمر از داخل آب به دنیای بازی نگاه کند، از این نود استفاده می‌کنیم، هر چه مقدار پارامتر **Amount** افزایش یابد شدت موج آب و شکستگی زاویه دید بیشتر خواهد بود، مقدار **Amount** را برابر ۵ قرار دهید و این پارامتر را برای اعداد ۰ تا ۵ این نود را امتحان کنید



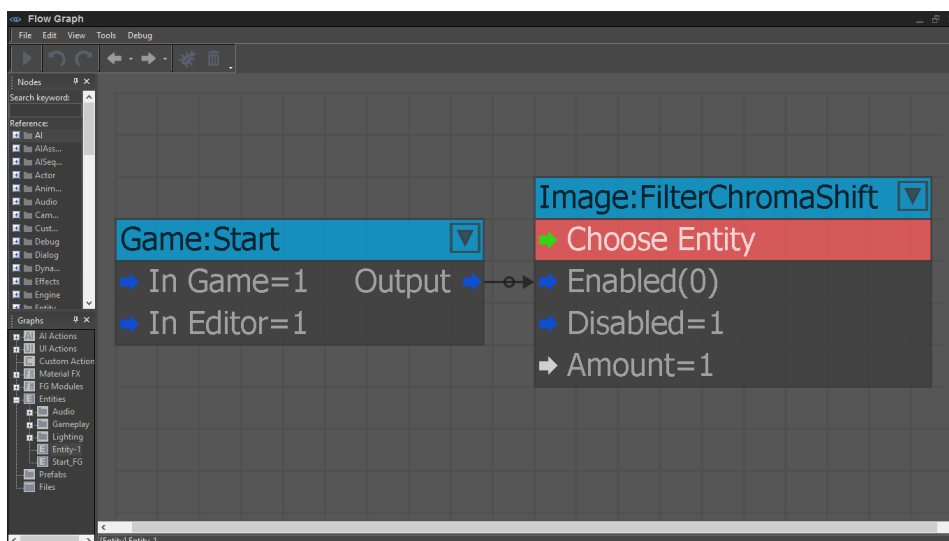


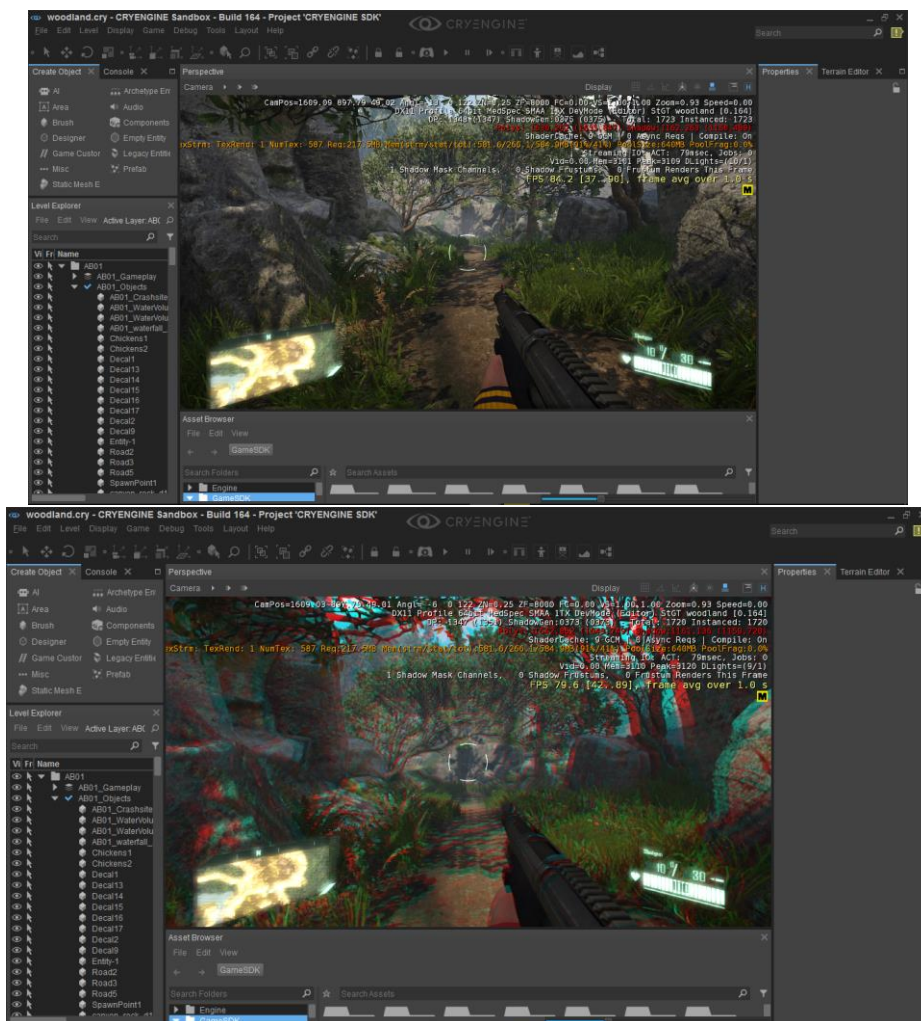
FilterBlur: این نود باعث تار شدن و نامشخص شدن دوربین در دنیای بازی خواهد شد، اشتباه نکنید این نود با نود DepthOfField متفاوت است، این نود کل مرحله را تار می کند در حالی که نود DepthOfField بر اساس مسافت و محدود دید پلیمر محیط بازی را تار خواهد کرد، پارامتر Amount = ۲ باشد و مقادیر دیگری از این پارامتر برای اعداد از فاصله ۰ تا ۲ را به این نود به صورت اعشاری نیز می توانید اختصاص دهید



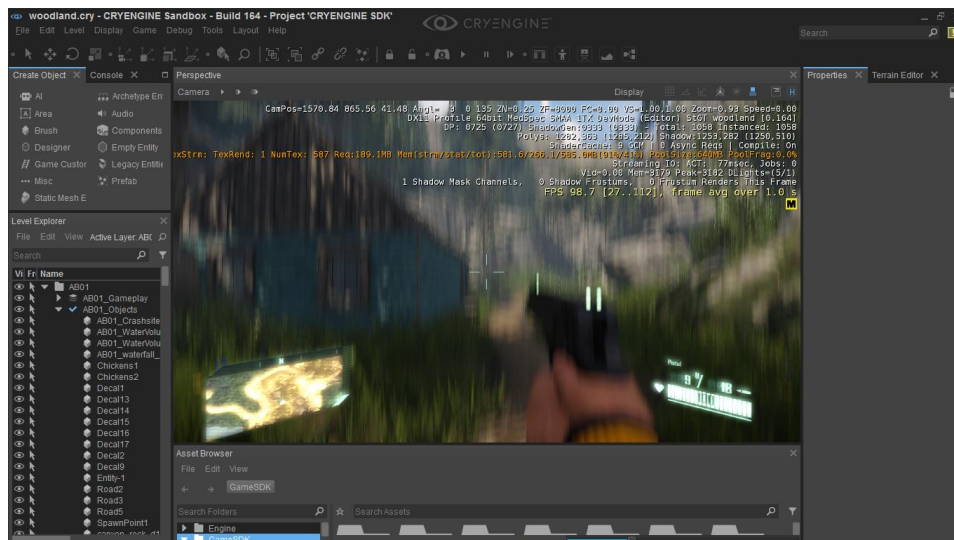


FilterChromaShift : برای شبیه سازی تجزیه نور استفاده می شود، مقدار پارامتر Amount=۱ قرار دهید اگرچه می توانید از مقادیر اعشاری نیز استفاده کنید

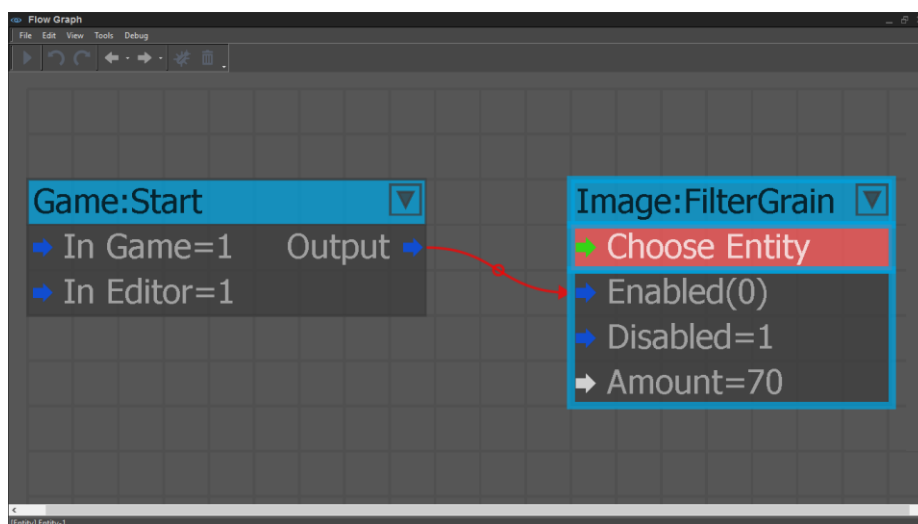


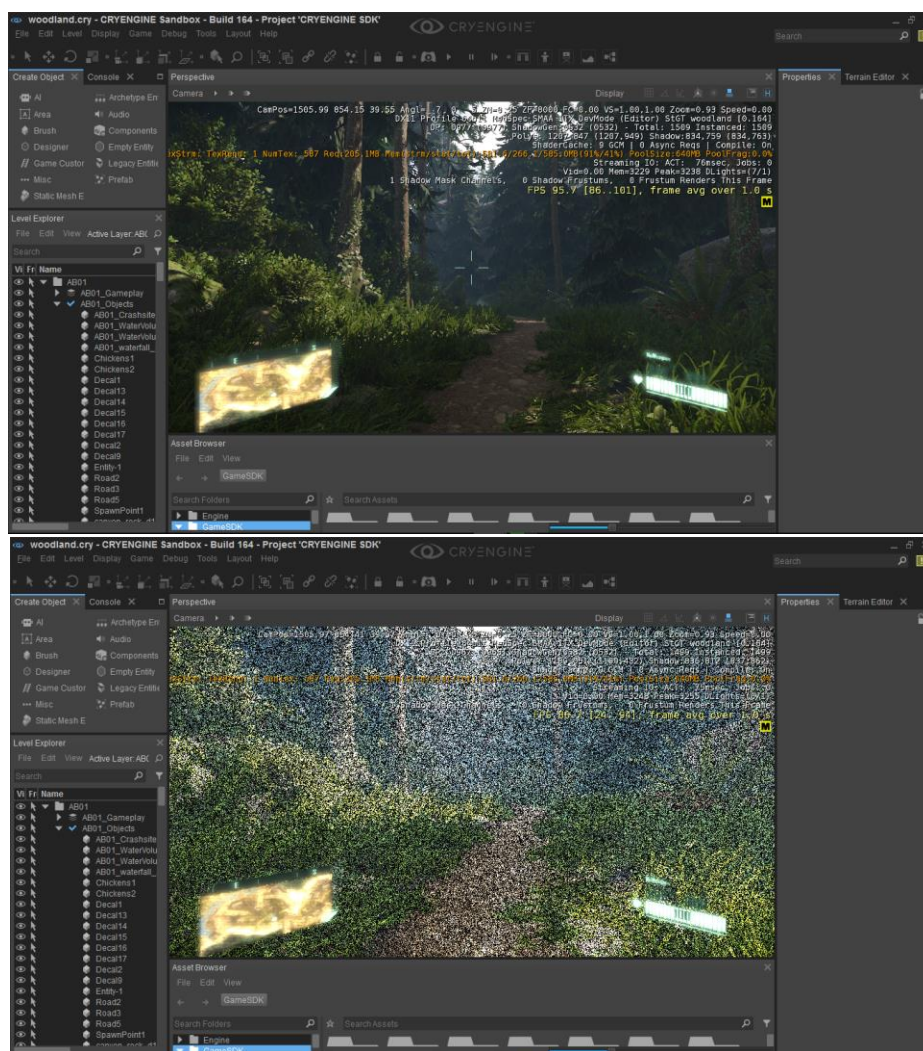


FilterDirectionalBlur : برای تار شدن براساس جهت استفاده می شود من بردار Direction را با مقادیر $x=0$, $y=1$, $z=0$ تنظیم کردم، شما می توانید از مقدار اعشاری و یا مقادیر صحیح بزرگتر یا کوچکتری استفاده کنید و نتیجه را به صورت دلخواه در آورید

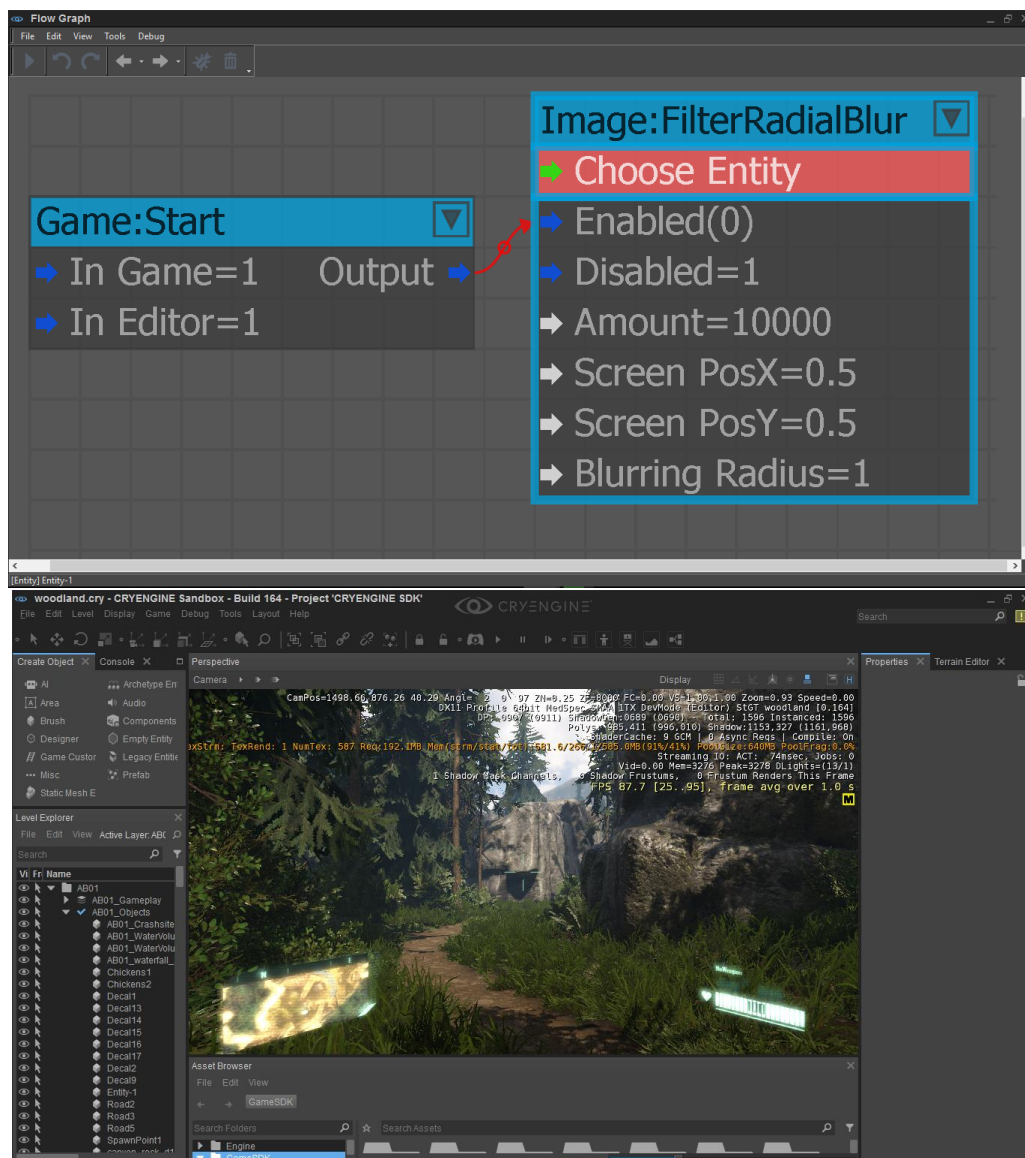


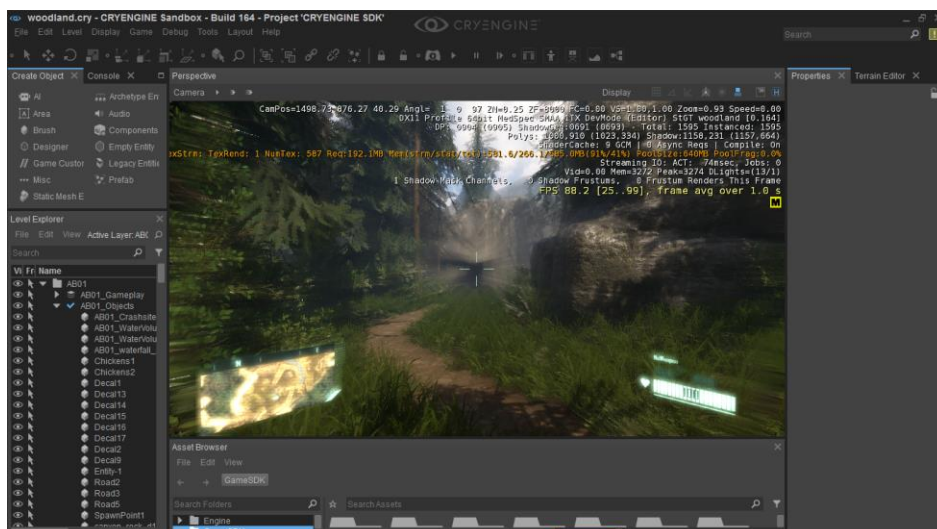
FilterGrain: برای ایجاد نویز و اختلالات تصویری حاصل از جریان برق استفاده می شود، نویزها و اختلالات تصویری را می توانید کاهش یا افزایش یا حتی شدید کنید، پارامتر **Amount** را برای مقادیر ۰ و ۵۰ و ۱۰۰ و ۱۵۰ و ۲۰۰ امتحان کنید، هر مقدار عددی دلخواه را می توانید به آن اختصاص دهید.



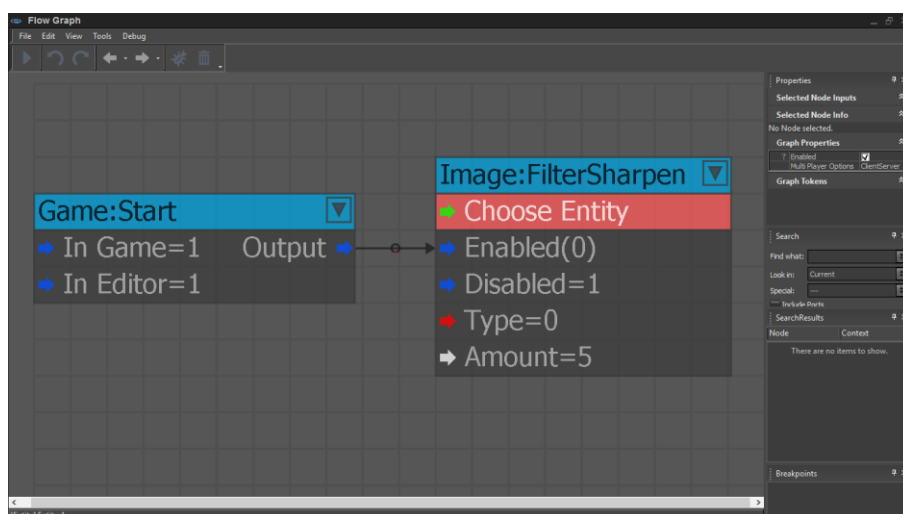


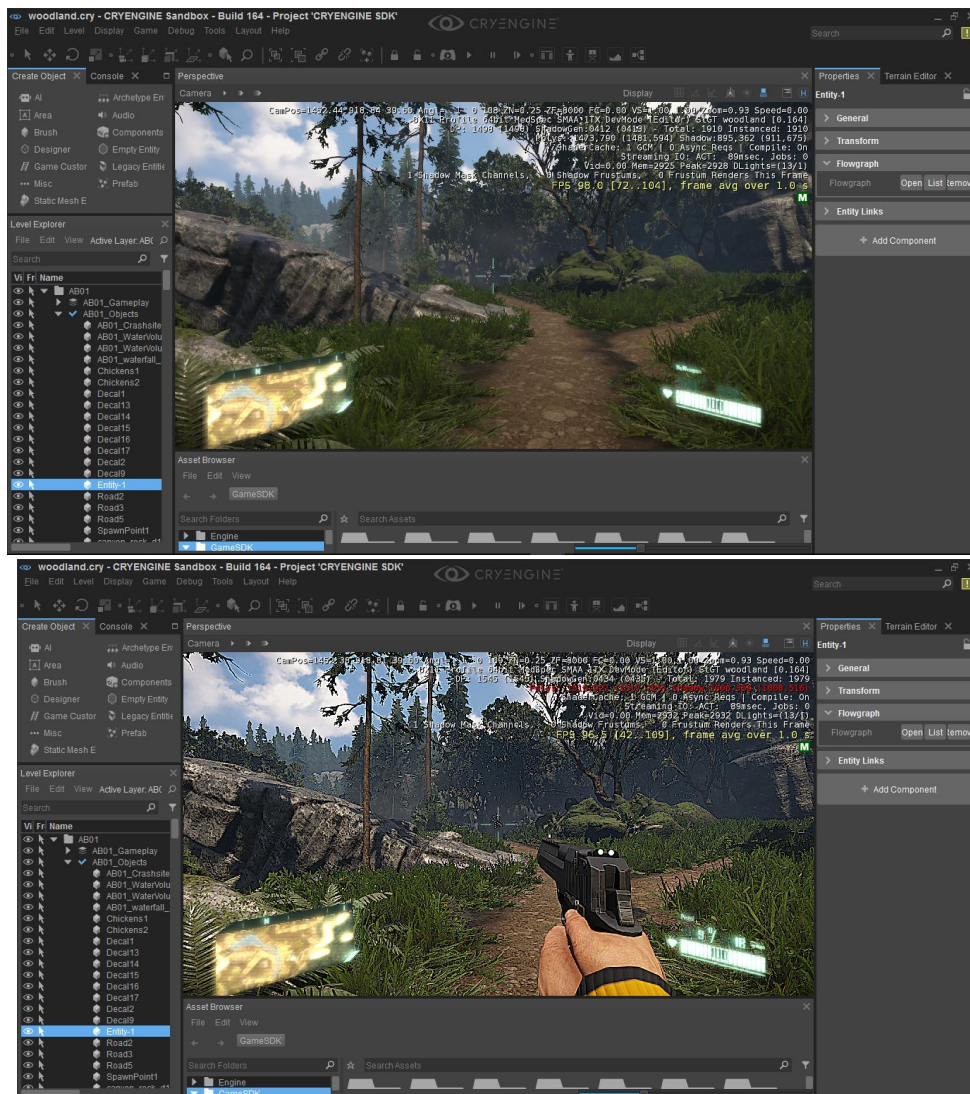
FilterRadialBlur : برای ایجاد تارشدگی تصویر بر اساس شعاع و مرکز تارشدگی تصویر استفاده می شود، مقدار Amount را به ۱۰۰۰۰ تغییر دهید و می بینید که مرکز تصویر به $x=0,5$, $y=0,5$ اختصاص داده شده است و شعاع تارشدگی با پارامتر $\text{Blurring Radius}=1$ است



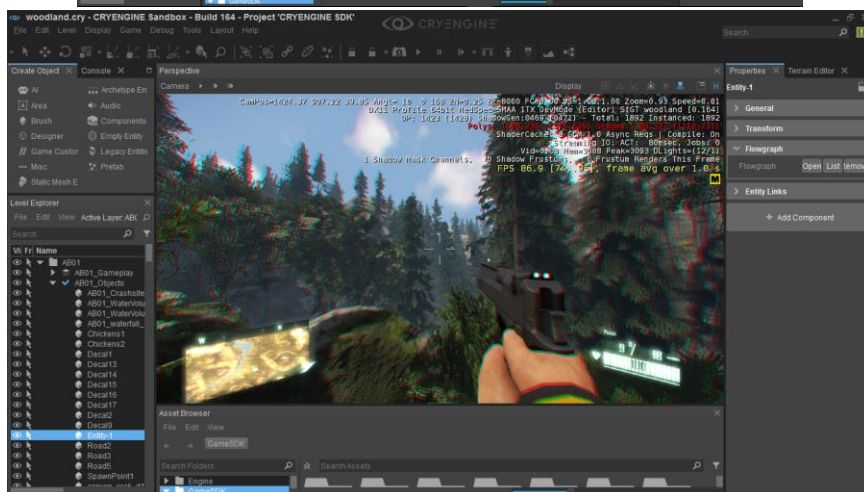
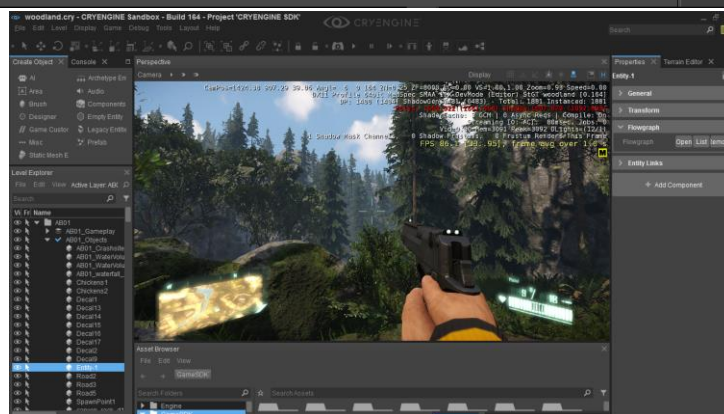
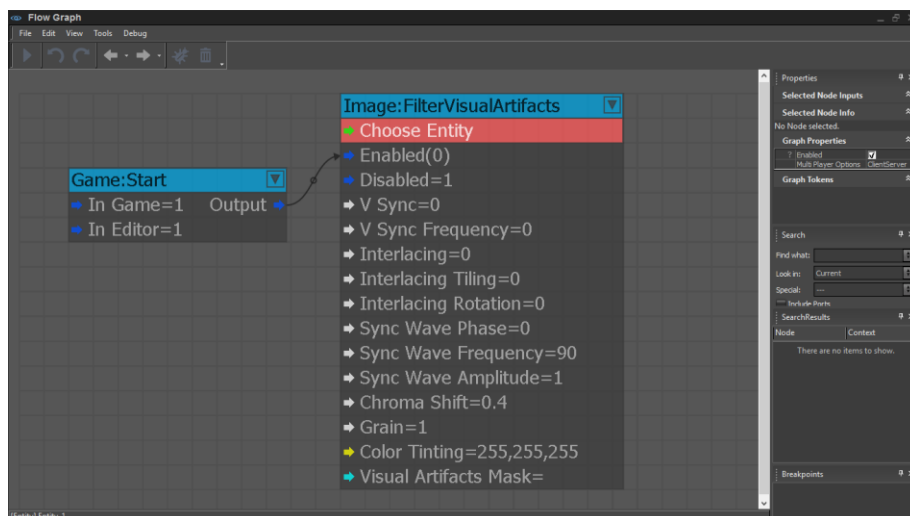


FilterSharpen : برای ایجاد تصاویر واضح و تیز استفاده می شود، این نود به شما کمک می کند که پیکسل های تصویری مقادیر واقعی را در بازی نشان دهند و در پروژه های معماری و ویژوالیزیشن یا همان پروژه های تعامل-بصری واقع گرایی را به بهترین شیوه نشان داده شوند، مقدار Amount=۵ قرار دهید

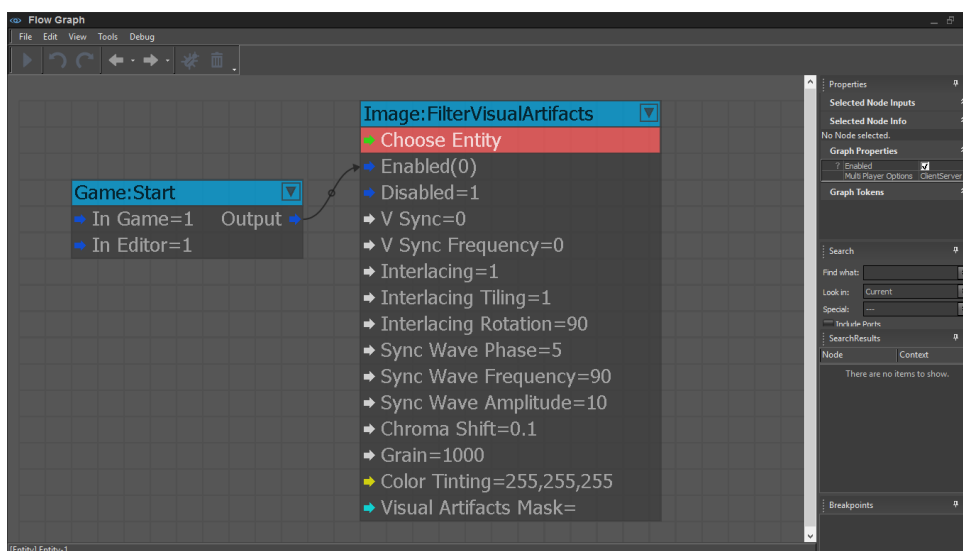


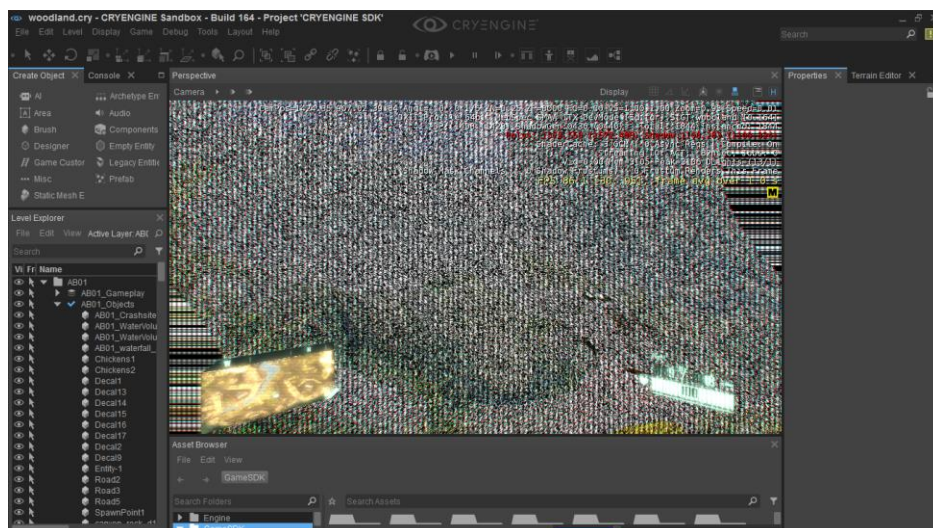


FilterVisualArtifacts : این نود کاربردهای فراوانی دارد و یکی از این کاربرد ها ایجاد شوک تصویر به دوربین پلیر در هنگام شلیک راکت، ایجاد و شبیه سازی زلزله و تغییرات شدید تصویری دارد، در اینجا می بینید که یک زلزله شدید شبیه سازی شده است، با این نود می توانید تجزیه نور را نیز شبیه سازی کنید

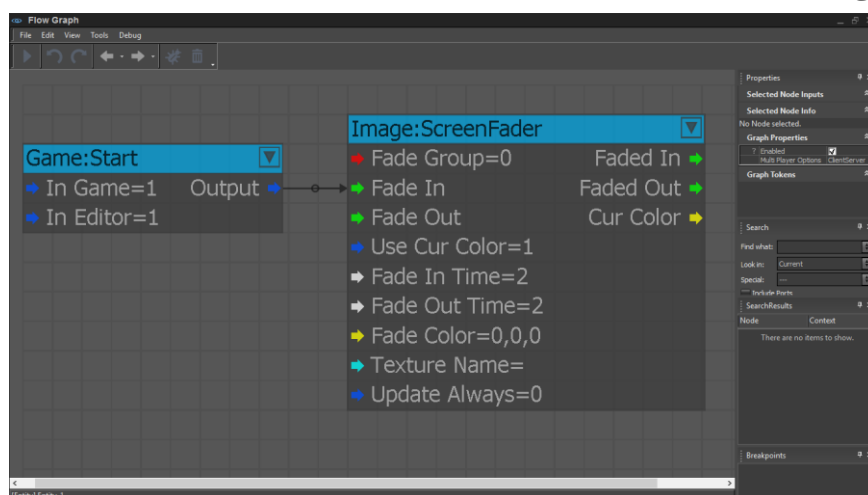


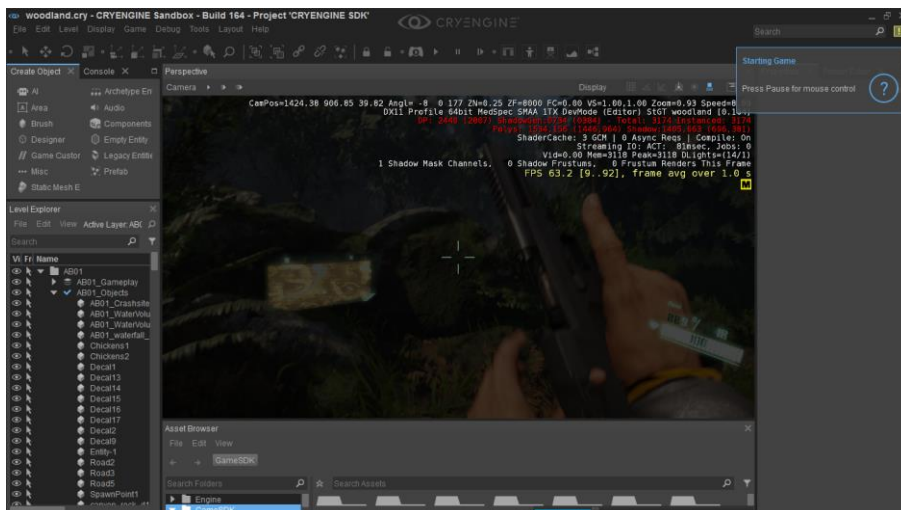
و در این مثال نیز اختلالات شدید تصویری ایجاد شده است و می‌توان بر اساس سناریو بازی این اختلالات را به دستگاه‌های موجودات فضایی در هنگام نزدیک شدن به پایگاه‌های نظامی آنها نسبت داد، البته تصاویر به صورت پویا و انیمیشنی نشان داده می‌شود اما در اینجا تنها به یک تصویر بسنده می‌کنم، با این نود می‌توان میزان پارازیت‌ها را براساس جریان برق نیز شبیه‌سازی کرد



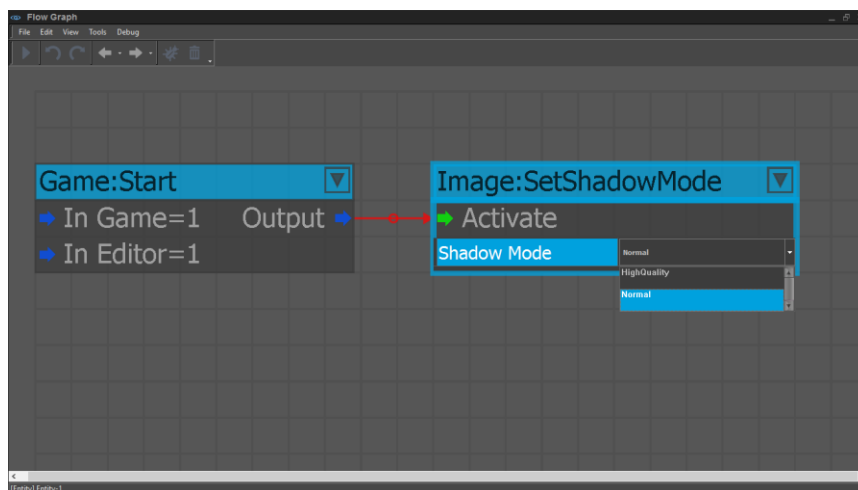


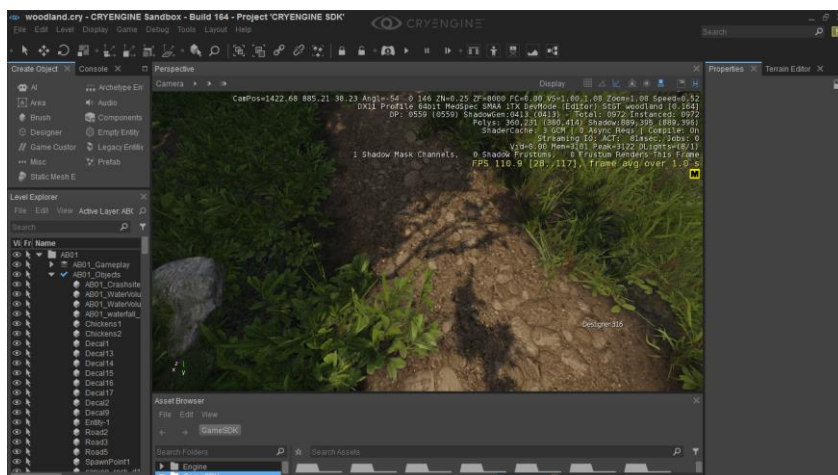
ScreenFader : این نود بسیار کاربردی است و هنگامی که بازی شروع می شود از یک صفحه تاریک شروع می شود و سپس صفحه تاریک کم کم محو شده و بازی شروع می شود، شما با استفاده از پارامتر Color می توانید رنگ محو شدگی را از سیاه و تاریک به رنگ دیگری تغییر دهید، زمان تغییر و محو شدگی نیز قابل تغییر است





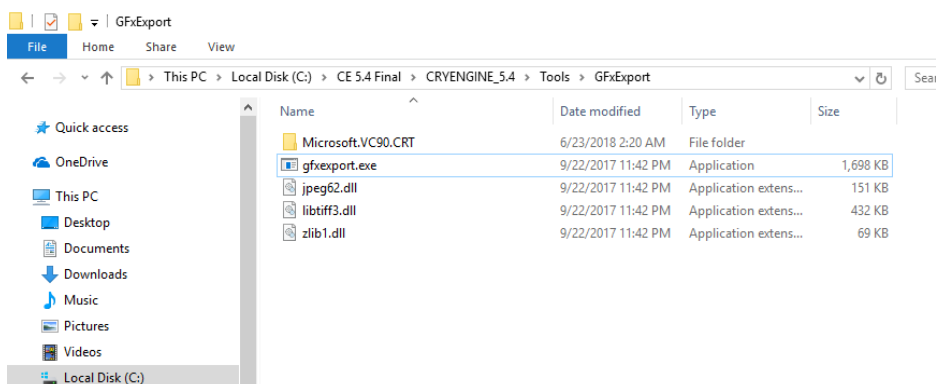
SetShadowMode: آیا کیفیت سایه ها در بازی برای شما مهم است؟ شما می توانید کیفیت سایه ها در بازی با پارامتر **ShadowMode** می تواند **Normal** (معمولی) و یا **High Quality** (بالا) باشد، کیفیت **Normal** (معمولی) جوابگوی بازی شما خواهد بود، زیرا که پردازش سایه ها و نورها سنگین است.



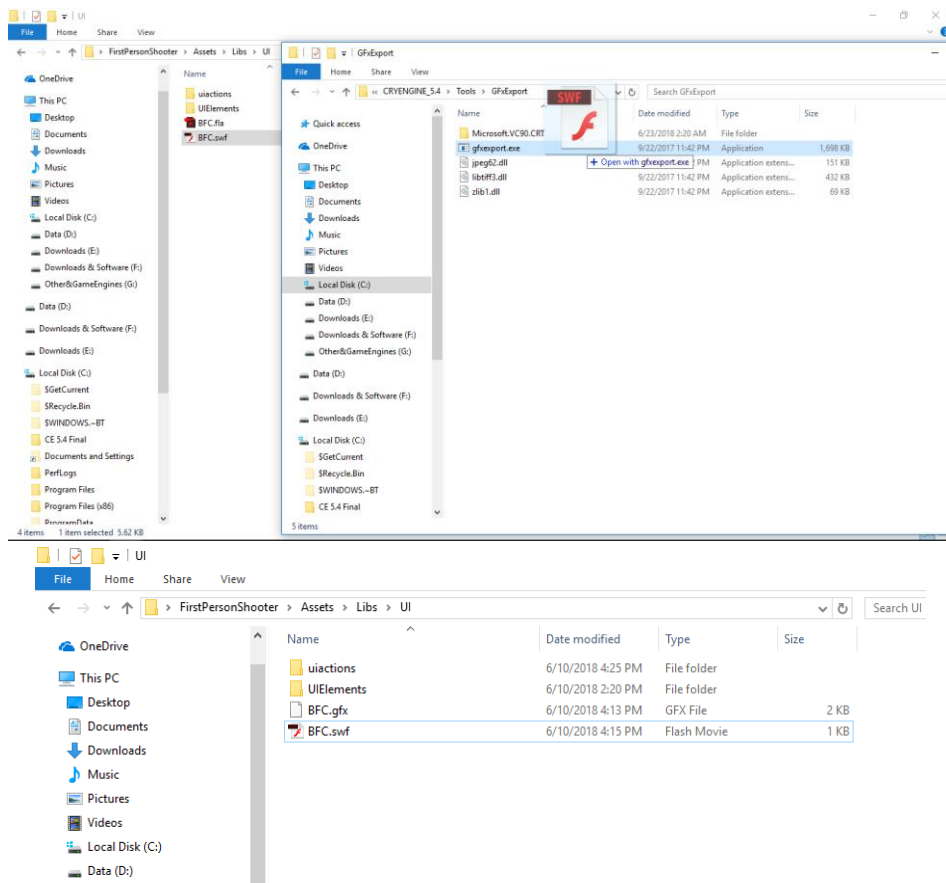


User Interface in Game-۲

برای نمایش دادن منوهای که با تکنولوژی Adobe Flash ساخته می شوند، به نودهای فلوگراف نیازمند هستید، بگذارید اول از ساخت فایل XML در مسیر مشخصی شروع کنید که به شما کمک می کند، مسیر درست را بپیماید و فایل های فلش تان را با تبدیل پسوند swf به پسوند gfx در بازی لود و به نمایش بگذارید، برای ایجاد فایل های gfx لازم است که فایل فلش با پسوند swf را به داخل مسیر زیر برروی فایل gfxexport.exe بکشید و رها کنید.

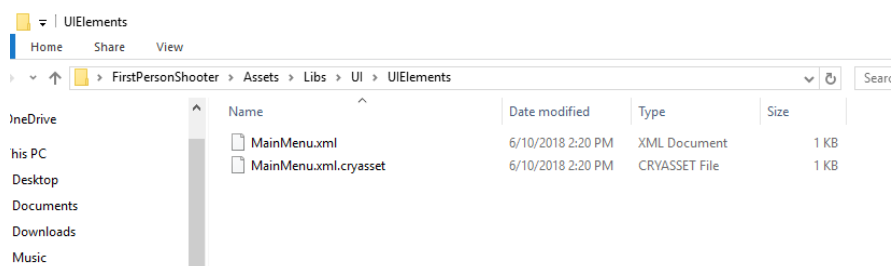


همانطور که می بینید مبدا فایل BFC.swf در مسیر و پوشه تودرتو Libs\UI داخل پوشه Assets باید باشد و فایل BFC.swf را به مسیر فایل gfxexport.exe و برروی فایل gfxexport.exe کشیده و رها می کنیم در مسیر Libs\UI فایل BFC.gfx ساخته می شود



اگر مسیر Assets\Libs\UI وجود نداشته باشد، باید این پوشه های تودرتو را ایجاد کنید و سپس فایل یا فایل های swf را در این مسیر کپی کنید، برای ساخت فایل xml مانند MainMenu.xml باید مسیر تودرتو با پوشه های Assets\Libs\UI\UIElements را بسازید و سپس داخل این مسیر فایل منوی اصلی (MainMenu.xml) یا هر سیستم رابط کاربری دیگری در

بازی را بسازید، فایل یا فایل های XML که شما می سازید داخل فلوگراف شناسایی شده و با نودهای فلوگراف می توانید با فایل های gfx کار کنید، فایل های gfx را لود کرده و سپس به نمایش بگذارید یا با دکمه های منو در بازی ارتباط برقرار کنید، مقادیر مهمات را کاهش دهید یا افزایش دهید و داخل منو یا رابط بازی نمایش دهید، اگرچه UI Editor در کرای انجین موجود نیست و به Adobe Flash وابسته است، انتظار می رود که در نسخه های بعدی UI Editor به کرای انجین اضافه شده و دیگر از فایل های فلش با نرم افزار Adobe Flash استفاده نشود.



در شکل زیر ساختار فایل xml را می بینید که از سه ساختار تشکیل شده است :

۱- UIElements : این ساختار کسه دارای نام AhmadKaramiMainMenu است می تواند هر اسم دلخواهی داشته باشد و این ساختار از یک زیر ساختار یا تگ UIElement با نام MainMenuAK ایجاد شده است که در داخل فلوگراف با همین اسم آن را در تصاویر پایین خواهید دید، اینجا پارامترهای controller_input مدیریت ورودی فایل فلش، cursor برای فعال شدن ماوس فلش یا ماوس کرای انجین در بازی، keyevents برای تاثیرگذاری کدهای صفحه کلید و mouseevents برای تاثیرگذاری کلیک های ماوس وجود دارد، در زیر ساختار بعدی با نام GFX فایل BFC.gfx با لایه گذاری عدد ۳ را می بینید و

این فایل همانطور که اشاره شد در مسیر `Libs\UI` قرار دارد، زیرساختار یا تگ `GFX` نیز یک زیرساختار دیگر دارد که با تگ `Constraints` اسم گذاری شده است و شامل قرار گیری فایل فلش بر روی صفحه بازی است و موقعیت فایل در کجای صفحه بازی لود شود، در گوشه بالا (`top`)، در گوشه پایین (`down`)، در مرکز (`center`)، در سمت راست (`right`) یا چپ (`left`) و مقیاس آن با `scale` در یک واحد است و حالت پویانمایی (`dynamic`) نیز در آن فعال شده است

۲- `Functions` : در این ساختار یا در این تگ هر چند تابع را که می خواهید در داخل فایل `xml` می توانید درج کنید و در تصویر پایین نیز یک مثال از زیرساختار تابع با تگ `function` وجود دارد که اسم تابع با نام `Test۰۱` برای فراخوانی و نام تابع با `func۰۱` وجود دارد، در اینجا پارامترهای `funcname` , `desc` می توانند مقدار دلخواهی داشته باشند و مقادیر این دو پارامتر اهمیت ندارد و همچنین در زیرساختار بعدی `param name` و `desc` نیز مقادیر دلخواه بوده و اهمیت ندارند، اینجا بیشتر به جنبه توضیح و پارامترهای کمکی اشاره می شود و در اینجا به پارامترهای کمکی نمی پردازم

۳- `Events` : این ساختار یا تگ مربوطه برای دکمه ها طراحی شده است و مختص به کرای انجین نیست، من زمانی که در حال ساخت پروژه های مختلف در محیط `VB۶ (Visual Basic ۶)` تا سال های `۲۰۰۵-۲۰۰۶` با استفاده `Windows API` و `Windows Registers` بودم از دستورات `FSCommand` استفاده می کردم، دستوراتی که به `FSCommand` ختم می شود به این صورت است که در داخل فایل فلش یک `fscommand` تعریف می کنید و با استفاده از آن یک اسم برای خارج از فایل فلش ارسال می شود، این اسم می تواند هر چیزی باشد اما شما می دانید دستور `fscommand` که داخل دکمه ها است و هر دکمه چه اسمی را به خارج از

فایل فلش ارسال می‌کند و با دستورات `if` یا `switch` تعیین می‌کنید که با کلیک کردن بر روی دکمه اول در فایل فلش دستور `fscommand` کلمه مثلاً `btn۰۱` را به فایل خارج از فلش ارسال می‌کند، من در `VB۶` نیز از `ActiveX` ی استفاده می‌کردم که می‌تونست فایل فلش را با پسوند `swf` در خود لود کند و با استفاده از این `ActiveX` در `VB۶` باید بررسی می‌کردم که آیا این کلمه ارسال شده برابر با رشته `"btn۰۱"` است یا نه، اگر برابر بود نشان می‌داد که من بر روی دکمه فلش کلیک کرده‌ام و اگر برابر نبود نشان می‌داد که بر روی دکمه فلش کلیک نکرده‌ام یا باید شرط‌های دیگر را در دیگر دکمه‌ها بررسی می‌کردم، در کرای انجین نیز همین فرمول برای فلوگراف وجود دارد و در داخل فلوگراف باید نود یا نودهای مربوط به دستورات `fscommand` را فراخوانی کنید، قضیه بسیار ساده است، همه کارها را فلوگراف برای شما انجام می‌دهد، کافیت شما این رابطه‌ها را بدانید.

در این ساختار `event name` که همان نام `event` است با `FS` شناخته می‌شود و پارامتر `desc` می‌تواند خالی باشد یا مقدار یا رشته دلخواهی داشته باشد و پارامتر `fscommand` با نام `okay۰۲` وجود دارد، این پارامتر به فلوگراف می‌گوید که نود `FS` را بساز و اگر پارامتر داخلی آن با نام `okay۰۲` بود نودهای مشتق شده و تکثیر شده و جدا شده از نود `FS` را اجرا کن، در واقع ساختار `xml` زیر برای ساخت نودها و همچنین شناسایی فایل فلش در داخل فلوگراف است.

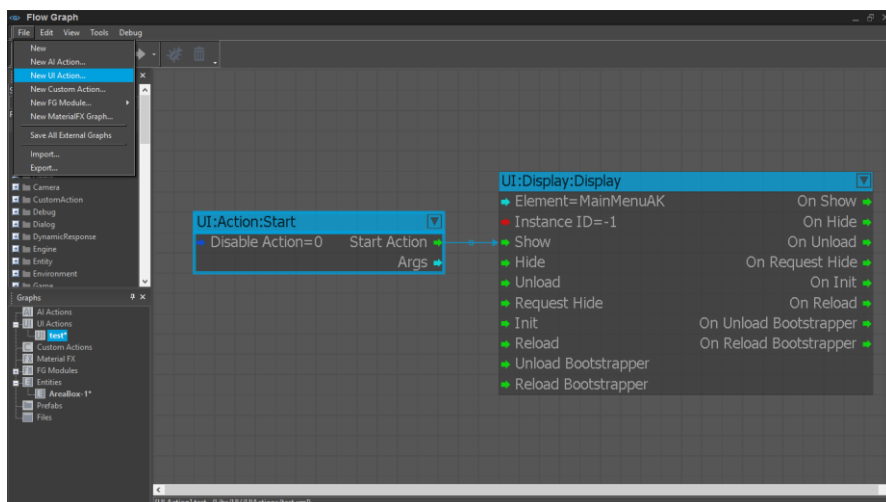
```

<?xml version="1.0"?>
- <UIElements name="AhmadKaramiMainMenu">
- <UIElement name="MainMenuAK" controller_input="1" cursor="1" keyevents="1" mouseevents="1">
- <GFx layer="3" file="BFC.gfx">
- <Constraints>
- <Align max="1" scale="1" valign="top" halign="left" mode="dynamic"/>
- </Constraints>
- </GFx>
- <functions>
- <function name="Test01" desc="my func is great" funcname="func01">
- <param name="MyFunc" desc="my function is great"/>
- </function>
- </functions>
- <events>
- <event name="FS" desc="" fscommand="okay02"/>
- </events>
- </UIElement>
</UIElements>

```

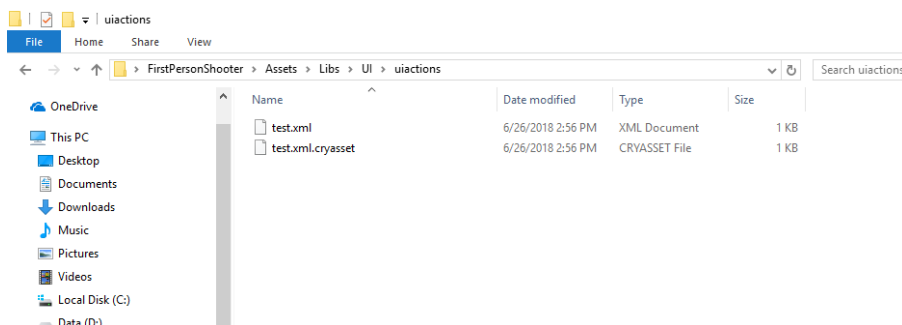
به فلوگراف برگردید و یک entity بسازید و داخل آن فلوگراف را فعال کنید و به منوی File مراجعه کرده و گزینه New UI Action... را انتخاب کنید، یک اسم با نام test برای آن اختصاص دهید و در مسیر Libs\UI\uiactions فایل test را ذخیره کنید، به صورت اتوماتیک کرای انجین متافایلی برای test.xml می‌سازد، فقط شما عمل ذخیره را انجام دهید، بقیه کارها را کرای انجین انجام می‌دهد، حالا نوبت آن است که دو نود به مشخصات زیر در فلوگراف فراخوانی کنید:

نود اول UI:Action:Start با راست کلیک کردن برروی فلوگراف و یک لیست طولانی از کاتالوگ‌ها و لیست‌های کشوی باز می‌شود، شما گزینه UI را انتخاب کرده و سپس زیر گزینه Action و دوباره یک لیست برای شما نشان می‌دهد، شما گزینه Start را باید انتخاب کنید، نود اول با اسمی که دارد، مسیر انجام کار را نشان می‌دهد، همان چیزی که من توضیح دادم. نود دوم UI:Display:Display که از روی اسم آن مشخص است، راست کلیک کرده و سپس مسیر گزینه UI و زیر گزینه Display و دوباره زیرگزینه Display را انتخاب کنید، طبق تصویر زیر باید این دو نود را به هم وصل کنید



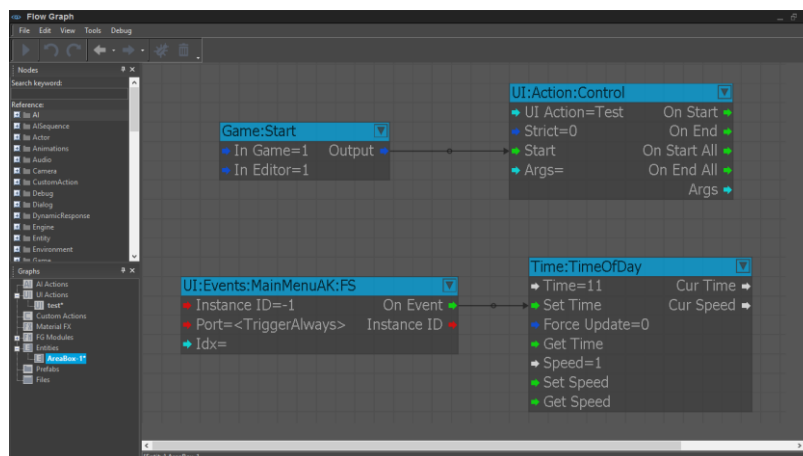
اگر دقت کنید ساختار Element آنچه که در فایل MainMenu.xml بود با نام Element=MainMenuAK در نود دوم را می بیند، فراموش نکنید که این دو نود را باید در یک فایل UI Action پیاده کنید و قدم بعدی برای انجام کار اضافه کردن نودها در داخل یک Entity خالی است که در صفحات قبلی به آن اشاره کردم.

باید قادر باشید که تصویر بالا را پیاده کنید، در تصویر زیر می بیند که ساختار فایل test.xml که در منوی File گزینه New UI Action... را انتخاب کردید، در این مسیر ذخیره شده است و متافایل آن با نام test.xml.cryasset ذخیره شده است.

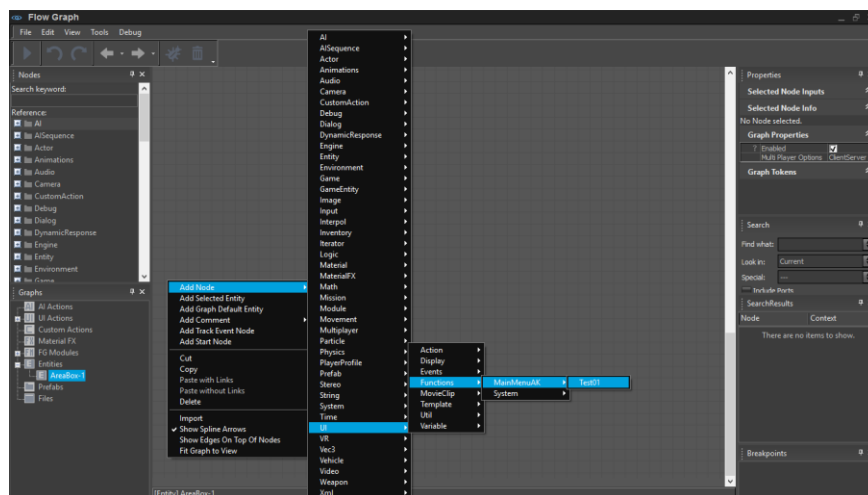


طبق تصویر زیر در داخل یک entity خالی یک فایل فلوگراف را ایجاد می کنید و نودها را طبق مسیری که در داخل اسم نود ها وجود دارد، اضافه کنید، این ساده ترین کاری است که در این کتاب وجود دارد، طبق نود `UI:Action:Control` شما باید قادر باشید که پارامتر `UI Action` را با اسم `test` لود کنید، این پارامتر به فایل `UI Action` اشاره دارد که در چند صفحه قبل به منوی `File` و گزینه `New UI Action...` مراجعه کردیم و فایل `test.xml` را ایجاد کردیم، فایل `test.xml` دارای دو نود بود که باعث می شد که ساختار یا تگ `Element` در `MainMenuAK` لود کند، این فرمول ساده ای است

به چند صفحه قبل برگردید به فایل `MainMenu.xml` برگردید و ساختار `Events` را با دستور `fscommand` که برابر `okay۰۲` و `event name` برابر با `FS` بود را به یاد بیاورید، طبق تصویر زیر و اسم نودها به مسیر اضافه شدن نودها مراجعه کنید، در اینجا یک نود `fscommand` به مسیر `UI:Events:MainMenuAK:FS` وجود دارد، همان `MainMenuAK` `UIElement` بود که در داخل `MainMenu.xml` با نام `Element` وجود داشت و در اینجا این نود بررسی می کند که اگر دکمه فلش لود شده در بازی کلیک شود زمان بازی به ساعت ۱۱ صبح تغییر خواهد کرد.



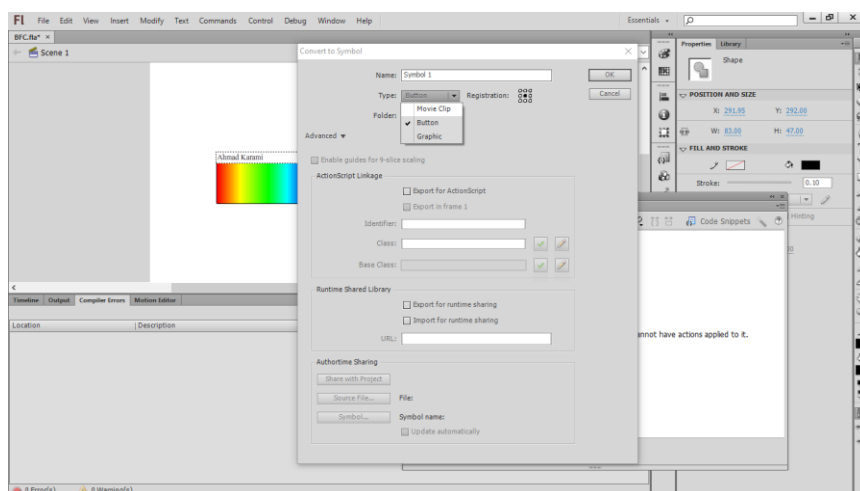
در تصویر زیر می بینید که طبق مسیر مراجعه شده تابعی با نام نود ۰۱ Test در فلوگراف وجود دارد، اگر فایل MainMenu.xml را بررسی کنید متوجه خواهید شد که در ساختار Functions شما یک تگ function اضافه کرده اید و اسم تابع ۰۱ Test است، طبق آنچه که در اسم نودها پیداست، به مسیر نودها در تصاویر بالا رفته و نودها را اضافه کنید، این ساده ترین مثال برای فهمیدن ارتباط دو تکنولوژی CryEngine و Adobe Flash است.



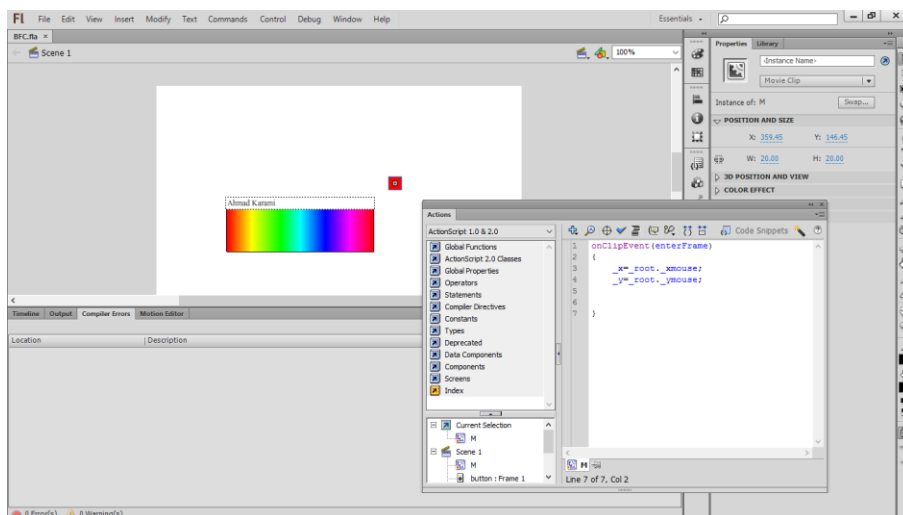
به Adobe Flash Professional CS6 خوش آمدید، در اینجا می‌توانید انواع و اقسام رابط کاربری را به بهترین شیوه از لحاظ گرافیک و پویانمایی طراحی کنید، از دید بسیاری از کاربران کرایتک که معتقدند کرای انجین نیاز به UI Editor ندارد، چون کرای انجین می‌تواند با Adobe Flash کار کند و از این تکنولوژی استفاده نماید.

یک فایل با نام BFC.flc با انتخاب زبان اسکریپت نویسی ۲،۰ ActionScript ایجاد کنید، فراموش نکنید که حتماً زبان اسکریپت نویسی نسخه ۲ یعنی Action Script ۲ باشد، چون زبان اسکریپت نویسی ۳ ActionScript در کرای انجین پشتیبانی نمی‌شود، در Stage (محیط کاربری سفید صحنه) می‌بینید که دو Object وجود دارد:

۱- یک مربع قرمز رنگ که نقش ماوس را برعهده دارد، اگر از ماوس کرای انجین در بازی نمی‌خواهید استفاده کنید، می‌توانید یک ماوس زیبا را در Flash ایجاد کنید، من از یک مربع قرمز ساده استفاده می‌کنم، مربع را باید از حالت Object یا Shape به حالت MovieClip تغییر دهید، بر روی مربع راست کلیک کنید و گزینه Convert to Symbol را انتخاب کنید و یک پنجره باز شده و از شما می‌خواهد که نوع Symbol تان را انتخاب کنید، در تصویر زیر مشخص است

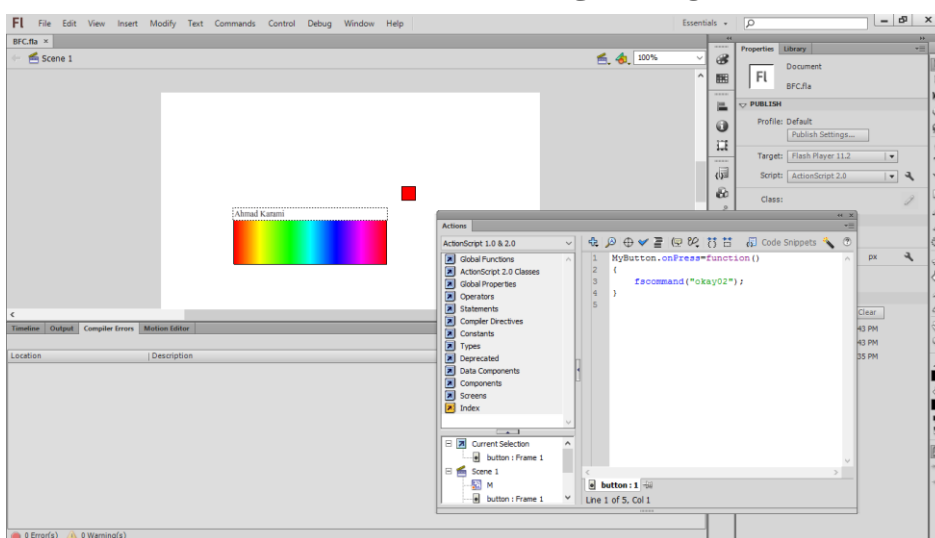


بعد از تبدیل و انتخاب مربع قرمز رنگ، دکمه F9 بر روی صفحه کلید را فشار دهید یا به منوی Window مراجعه کرده و گزینه Actions را انتخاب کنید تا پنجره Actions برای شما باز شود و فراموش نکنید که حتما مربع قرمز را انتخاب کرده باشید و سپس در این پنجره اسکریپت مربوط به رفتار ماوس را اضافه کنید و باعث حرکت مربع قرمز رنگ در دو جهت X, Y در محور مختصات ماوس خواهد شد

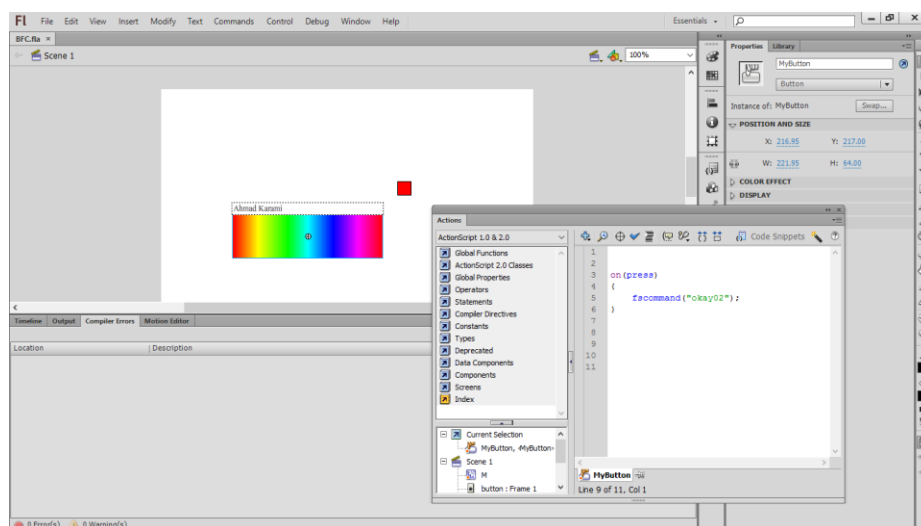


شما هر تعداد دکمه دلخواهی را که می خواهید بسازید و در اینجا برای مثال یک دکمه ساده می تواند به این صورت ساخته شود، یک Shape (شکل)

بکشید و سپس برروی شکل راست کلیک کرده و گزینه `convert to Symbol` را انتخاب کنید، سپس در پنجره ظاهر شده گزینه `Button` را انتخاب کرده و هر اسمی که دوست دارید به زبان انگلیسی (نمی توانید اسمی دکمه ها با حروف کردی یا فارسی باشد) به آن اختصاص دهید، من اسم `MyButton` را به آن اختصاص می دهم اسم خودم را با یک `Text` نیز اضافه میکنم و در حالی که برروی جایی خالی از صحنه کلیک می کنید، در پنجره `Actions` کد زیر را به آن اختصاص می دید، همانطور که می بیند دستور `fscommand` با نام `okay۰۲` ثبت شده است و کلمه `okay۰۲` به خارج از فایل فلش ارسال شده و در داخل فلوگراف گرفته می شود و توسط `MainMenu.xml` در فلوگراف به کار گیری می شود با نام نود `FS` زمان بازی به ساعت ۱۱ صبح تغییر می کند.

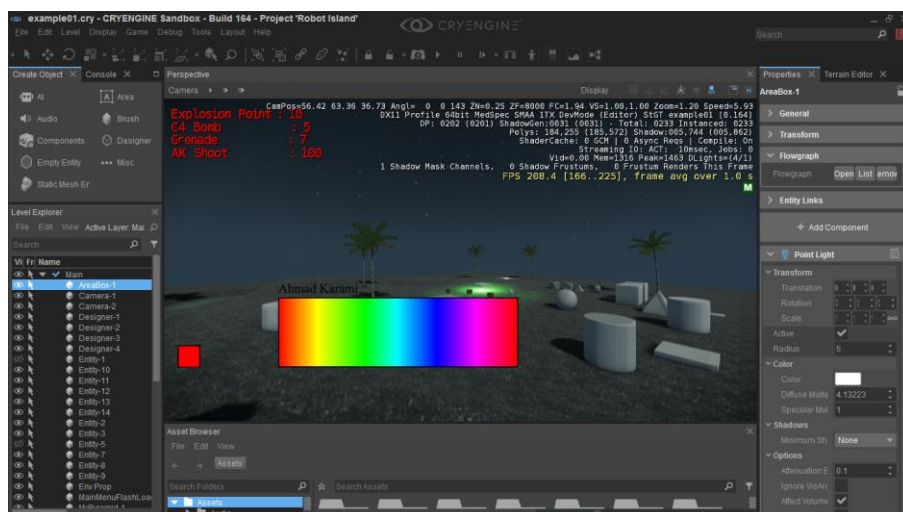


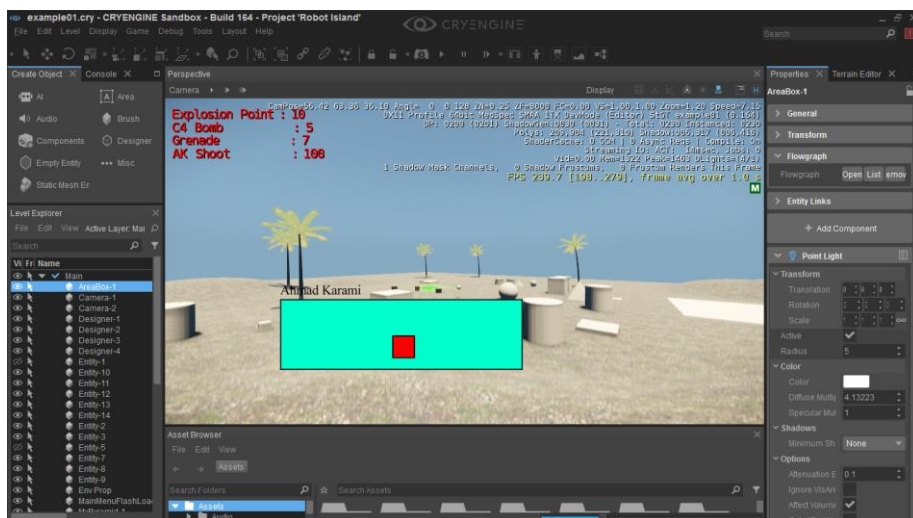
دکمه `MyButton` را انتخاب کنید و طبق پنجره `Actions`، کد یا اسکریپت زیر را به صحنه اضافه می کنیم



شما می توانید دکمه های بیشتری را به منوی تان در بازی اضافه کنید، شما لازم است که تگ event بیشتری را داخل MainMenu.xml و تابع های بیشتری (on(press) در داخل دکمه های فلش با دستور fscommand اضافه کنید.

در تصویر زیر می بیند که مرحله بازی در شب است و با کلیک کردن بر روی دکمه رنگین کمانی، بازی از شب به روز تغییر خواهد یافت

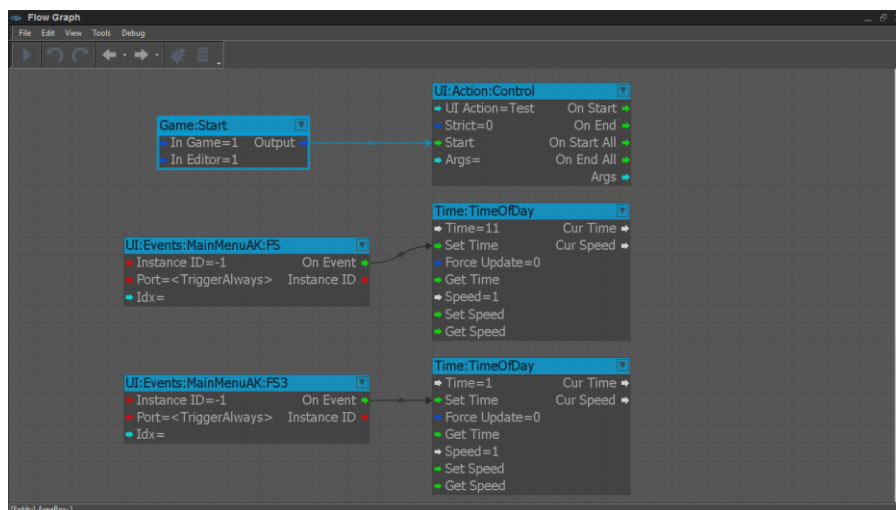




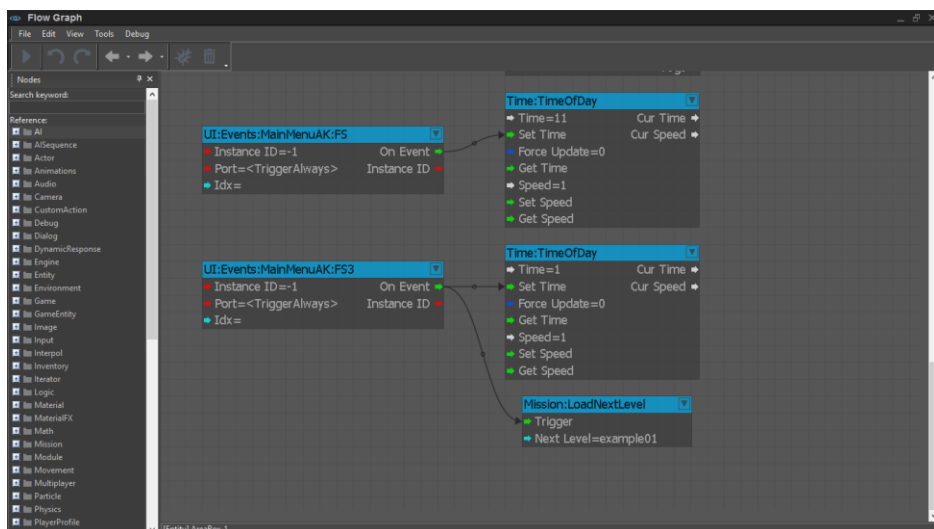
در دو تصویر بالا مربع قرمز رنگ نیز نقش ماوس را ایفا می کند و شما موفق شدید که درک کنید ارتباط دو تکنولوژی کرای انجین و فلش چگونه امکان پذیر است، یک مثال تکمیلی دیگر به صورت تصویری را ببینید، در فایل MainMenu.xml یک تگ دیگر با نام FS۳ اضافه کرده ام و fscommand را با okay۰۳ در نظر گرفته ام، من UI Action با نام فایل test در فلوگراف را بدون تغییر رها کرده ام.

```
<?xml version="1.0"?>
- <UIElements name="AhmadKaramiMainMenu">
  - <UIElement name="MainMenuAK" controller_input="1" cursor="1" keyevents="1" mouseevents="1">
    - <Gfx layer="3" file="BFC.gfx">
      - <Constraints>
        <Align max="1" scale="1" valign="top" halign="left" mode="dynamic"/>
      </Constraints>
    </Gfx>
  - <functions>
    - <function name="Test01" desc="my func is great" funcname="func01">
      <param name="MyFunc" desc="my function is great"/>
    </function>
  </functions>
  - <events>
    <event name="FS" desc="" fscommand="okay02"/>
    <event name="FS3" desc="" fscommand="okay03"/>
  </events>
</UIElement>
</UIElements>
```

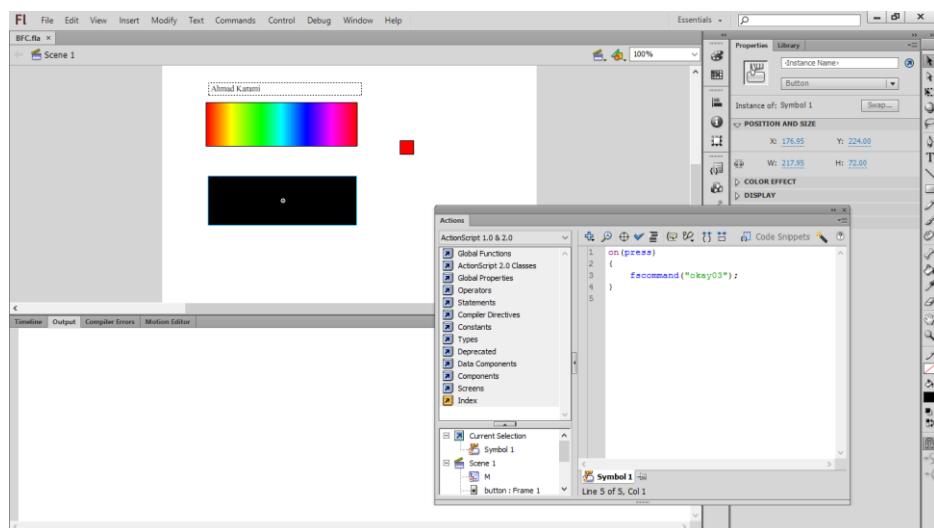
در فلوگراف در بخش مربوط به اضافه کردن نودها به یک Entity خالی، نود FS۳ را به نود TimeOfDay وصل کرده‌ام، در واقع با کلیک برروی دکمه رنگین کمان زمان بازی به روز تغییر خواهد یافت و با کلیک برروی دکمه سیاه زمان بازی به شب تغییر داده خواهد شد.



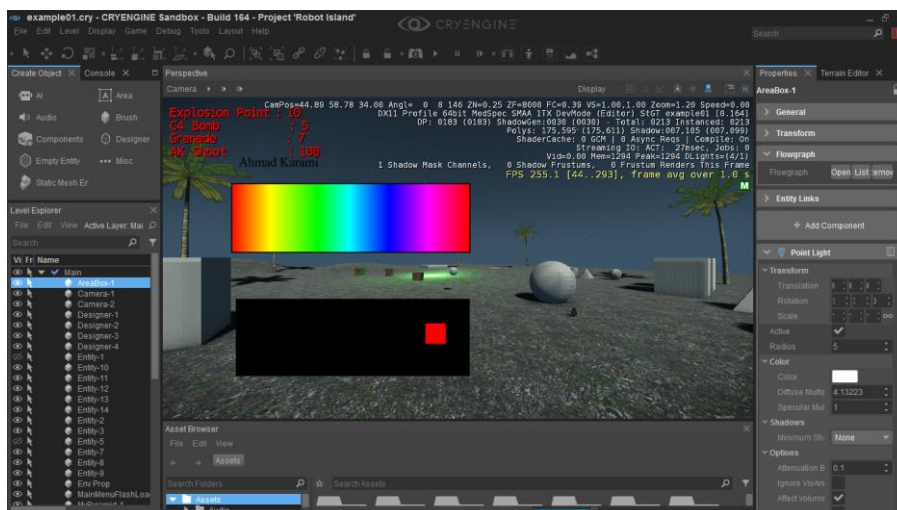
در تصویر زیر من یک نود با نام Mission:LoadNextLevel اضافه کرده‌ام که با کلیک برروی دکمه سیاه رنگ مرحله ۰۱ example در بازی را لود خواهد کرد، شما می‌توانید مراحل دیگری از بازی را نیز با همین شیوه لود کنید، برای دیدن لود مرحله شما نباید در ادیتور باشید و برای دیدن نتیجه لود، باید از بازی خروجی بگیرید با پسوند exe یا در محیط ویژوال استودیو بازی را اجرا کنید و نتیجه لود مرحله را مشاهده کنید.



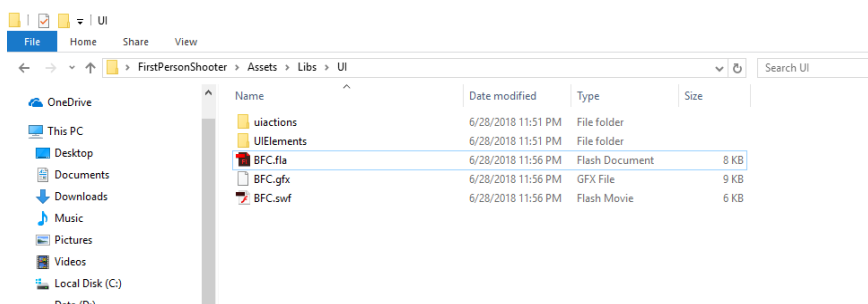
در فایل فلش یک Shape سیاه را اضافه کرده و به دکمه سیاه تغییر دادم و دکمه سیاه را انتخاب کردم و کد اسکریپتی زیر را در پنجره Actions نوشتم



نتیجه به صورت زیر نمایش داده خواهد شد و با کلیک برروی دکمه رنگین کمان زمان بازی روز خواهد بود و با کلیک برروی دکمه سیاه زمان بازی شب خواهد شد، از آنجایی که بازی در ادیتور اجرا می شود، لود مرحله ۰۱ example با کلیک برروی دکمه سیاه رنگ اتفاق نخواهد افتاد.



این ساده ترین مثال برای ایجاد رابط کاربری در بازی با استفاده از تکنولوژی Adobe Flash است و نودهای زیاد دیگری در فلوگراف وجود دارند که می توان از این نودها در این کتاب صرف نظر کرد اما دو بخش جلوه های ویژه دورین و ساخت منوی بازی مهمترین مباحث مربوط در فلوگراف هستند که پوشش دادم



همانطور که در شکل بالا می بینید، منبع فایل BFC fla، فایل MainMenu.xml و فایل های دیگر در مسیر پروژه Island Robot بر روی DVD در ضمیمه این کتاب وجود دارد

فصل یازدهم

زبان جدید Schematyc و

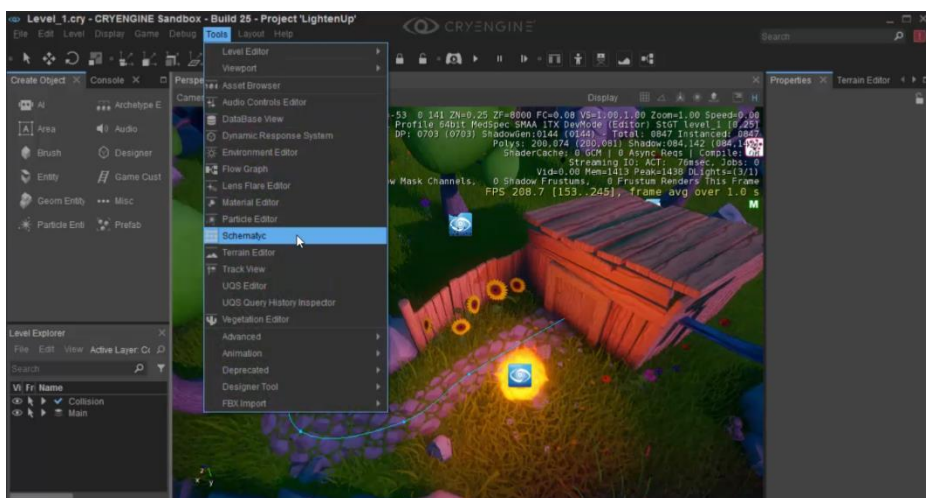
محیط ادیتوری آن

نمای کلی از پنجره سیماتیک

در کرای انجین ۵,۳ اولین نسخه از سیماتیک به صورت آلفا منتشر شد، این سیستم جدید برنامه نویسی بصری با بیشترین استقبال کاربران دنیا مواجه شد، زبان جدید بسیار پویا و جذاب است و شما می توانید به سرعت پروتوتایپ های خودتون را طراحی کنید، این سیستم جدید با نام سیماتیک یا به زبان فارسی شیماتیک نام برده می شود، با انتشار کرای انجین ۵,۴ شوک عجیبی به کاربران وارد شد، زبان سیماتیک در بخش کامبونت ها توسعه یافته بود اما دسترسی به توابع و متغیرها در بخشی از کامبونت ها غیرممکن بود، این مسئله باعث شد که صدای اعتراض زیادی از طراحان بازی به کرایتک برسد، برخلاف نسخه ۵,۳ کرای انجین، در نسخه کرای انجین ۵,۴ نمی توانستید حتی یک بازی خیلی ساده با سیماتیک بسازید، رابط کاربری سیماتیک در نسخه ۵,۴ تغییر یافته بود اما عملاً دسترسی به اشیاء داخلی پریفب های سیماتیکی غیرممکن بود، کرایتک با انتشار میزگردی خبری اعلام کرد که ما در حال باز طراحی مجدد سیماتیک توسط تیم توسعه دهنده بازی Hunt هستیم و با بازطراحی سیماتیک می خواهیم سیماتیک نسخه دو را به زودی منتشر کنیم، این خبر بسیار مهم بود و امیدوارکننده، کرایتک همچنان اعلام کرد که اولویت اول طراحان بازی زبان ++C باشد و ما همچنان در حال توسعه سیماتیک و دریافت بازخورد کاربران در سطح جهان هستیم، فراموش نکنید که شما نیز اولین اولویت تان زبان ++C باشد و زبان #C نیز دومین اولویت و حتی می تواند اولویت سوم باشد، چون سیماتیک در میان طراحان بازی محبوب تر است، حتی اگر آموزش های ویدئویی برای آن منتشر نشود، سیماتیک براساس شکل و ظاهر کاملاً قابل درک است و اگر شما قبلاً با

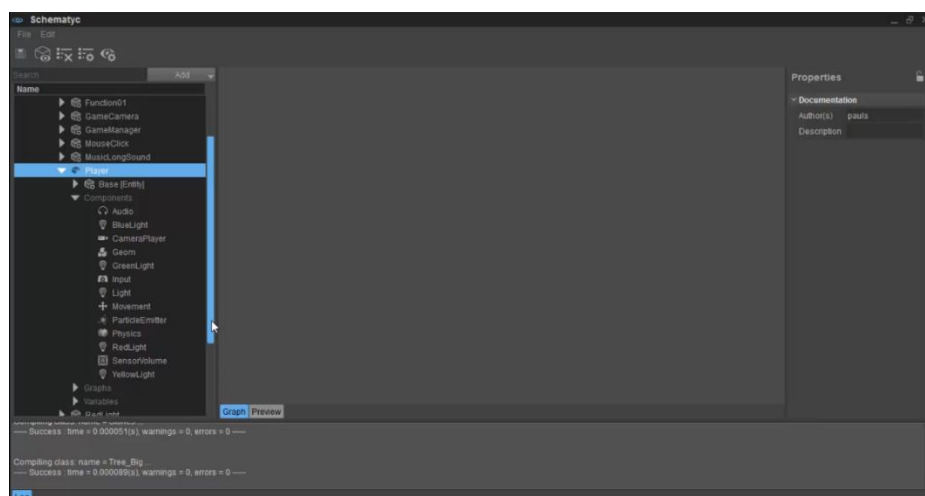
زبان های ویژوال اسکریپتینگ دیگر کار کرده باشید، با صرف چند ساعت سیماتیک را کاملا یاد خواهید گرفت، زبان سیماتیک گرافیکی است و برای ساخت بازی های بزرگ با این زبان شما با چند هزار گراف سروکار خواهید داشت و هر گراف می تواند شامل صدها گره یا نود باشد، طعم سیماتیک را تجربه کنید، چقدر خوش مزه است، ابتدا رابط کاربری سیماتیک در کرای انجین ۵،۳ را تشریح می کنم و سپس به رابط کاربری سیماتیک در کرای انجین ۵،۴ می پردازم، دلایل این کار را تشریح خواهم کرد

برای دسترسی به سیماتیک به منوی Tools مراجعه کرده و گزینه Schematyc را انتخاب کنید



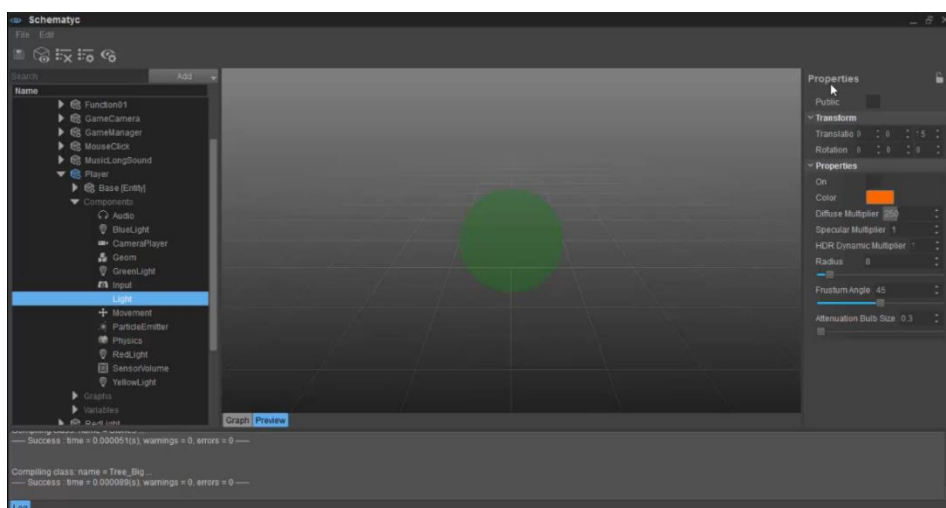
این پروژه LightenUp است و شامل کلاس های مختلف بوده و بر روی DVD در ضمیمه این کتاب برای نسخه کرای انجین ۵,۳ موجود است، اگرچه من این پروژه را توسعه داده ام و کلاس های بیشتری به همراه یک مجموعه آموزش ویدئویی در DVD قرار داده ام و با حل مثال های مختلف درک بسیار بالاتری از سیستم ویرژوال اسکریپتینگ یا به اصطلاح نودبیس را به شما می

دهد



در پروژه های مختلف در سیماتیک شما فضای Graph و Preview را تجربه خواهید کرد، در اینجا فضای گراف خالی از نودها است و همچنین می بینید که تب Graph فعال است، این دو فضا برای هر کلاس می تواند وجود داشته باشد اما برای کامبونت هایی مانند Movement ، Input فضای Preview وجود ندارد، چون کامبونت Movement ، Input فاقد فضای سه بعدی Preview هستند، این دو کامبونت مفهوم گرافایی دارند و در زمان طراحی مرحله و اجرای بازی مفهوم پیدا خواهند کرد، کلاس پلیر در شکل بالا شامل کامبونت های مختلف Audio ، BlueLight ، CameraPlayer،

Geom, GreenLight, Input, Light, Movement, ParticleEmitter, Physics, RedLight, SensorVolume, YellowLight می بینید که تعداد این کامبونت ها زیاد است، در واقع این حداقل کامبونت هایی است که در یک پلیمر می تواند وجود داشته باشد، در کرای انجین ۵،۵ تعداد بسیار زیادی از کامبونت ها اضافه شده است



در تصویر بالا فضای سه بعدی Preview را در تب Preview می بینید و یک کره سبز رنگ با نام کامبونت و البته از نوع SensorVolume وجود دارد، در این تصویر کلاس پلیمر شامل بخش های Base[Entity] , Graphs , Components و Variables است و برای هر کلاسی که شما ایجاد می کنید این ۴ بخش همیشه وجود خواهد داشت.

Base[Entity]: این بخش به صورت پیش فرض و با صرف در نظر گرفتن یک ID برای کلاس است

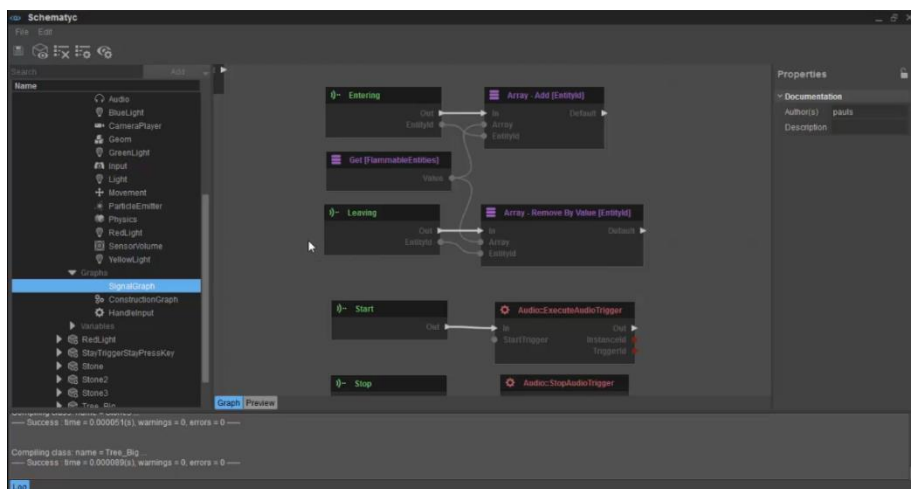
Components: این بخش کامبونت های مختلف را در خود جای می دهد و شما هر کامبونتی را که اضافه می کنید، در این بخش قرار می گیرد.

Graphs: برنامه نویسی برای کلاس و ارتباط با توابع، متدها و متغیرها برای کامبونت ها در رابطه با این بخش است، شما در این بخش کامبونت

ها، سیگنال های داخلی (محلی) و خارجی (سراسری)، ثابت های نام دار، متغیرها، سیگنال های دریافت کننده را مدیریت و کنترل می کنید، در واقع هر اتفاقی که قرار است به صورت رویداد رخ دهد، در این بخش برنامه نویسی می شود، فراموش نکنید که با زبان C++ می توانید کامپونت ها و کلاس ها را ایجاد کنید و در سیماتیک به کار ببرید.

Variables : متغیرهای داخلی (محلی) و خارجی (سراسری) را در این بخش و یک بخش دیگر (در ادامه توضیح داده خواهد شد) می توانید تعیین کنید، نوع این متغیرها می تواند از انواع `Float` , `Int32` , `Int16`, `Int64`, `Byte` , `EntityID` , `Entity` , `Enum` و غیره باشد، از آنجایی که در این بخش یک کلاس وجود دارد، داخل یک کلاس نوع متغیر داخلی یا همان محلی خواهد بود، متغیرها مانند ظرف هایی هستند که می توانند اعداد، اشیاء و محتوای دیگری را در خود نگه دارند تا برنامه نویس یا طراح بازی بتواند در منطق بازی از آنها استفاده کند.

بر روی **DVD** در ضمیمه این کتاب، توضیحات بیشتری را می توانید در رابطه با این بخش ها با ویدئوهایی آموزشی دریافت کنید یا با مرجعه به اکانت یوتیوب من یا آپارات من بیشترین آموزش های ویدئویی را دریافت کنید



در تصویر بالا فضای گراف با تب Graph را با نودهای مختلف می بینید، سیگنال های داخلی با نام های مختلف به صورت سبزرنگ با نودهای (گره ها) سیاه رنگ می بیند، سیگنال های داخلی به شرح زیر هستند:

Start : هنگامی نودها اجرا می شوند که بازی شروع شود، در واقع با شروع بازی این سیگنال فقط یکبار اجرا می شود، مثلاً انتساب محتوای مناسب به متغیرهای مختلف داخل کلاس ها

Stop : هنگامی نودها اجرا می شوند که بازی خاتمه یابد، در واقع با خروج از بازی این سیگنال فقط یکبار اجرا می شود، مثلاً ذخیره امتیاز و ذخیره پارامترهایی که لازم است در شروع مجدد بازی استفاده شوند، اگرچه در زمان اجرای بازی نیز می توانید این متغیرها را ذخیره کنید

Update : هنگامی نودها اجرا می شوند که بازی در حال اجرا باشد و این نودها در هر برهه زمانی (DeltaTime) چندین بار اجرا می شوند، این روند تا هنگامی که داخل بازی هستیم و از بازی خارج نشده ایم اتفاق می افتد، مثلاً کنترل ماوس، صفحه کلید برای کامبونت های Movement و Input در این سیگنال می توانند پردازش شوند

Entering : در کامبونت SensorVoulme وقتی یک شی وارد این محدوده از فضای سبز رنگ (تصویر کره سبز رنگ بالا را به یاد آورید) می شود، این سیگنال اجرا می شود، در واقع با ورود یک شی به این محدوده عمل TriggerEnter اتفاق می افتد

Leaving : در کامبونت SensorVoulme وقتی یک شی این محدوده از فضای سبز رنگ (تصویر کره سبز رنگ بالا را به یاد آورید) را ترک می کند، این سیگنال اجرا می شود، در واقع با خروج یک شی از این محدوده عمل TriggerExit اتفاق می افتد

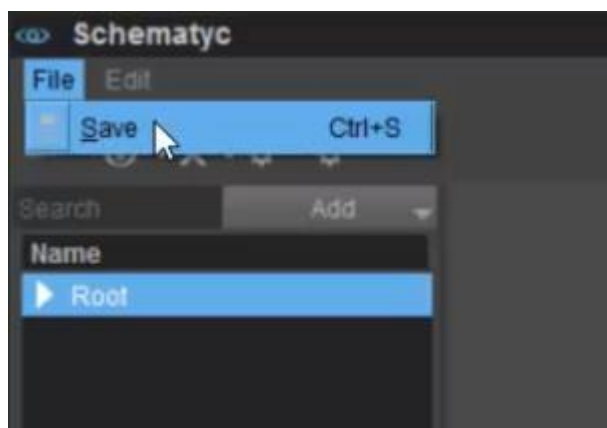
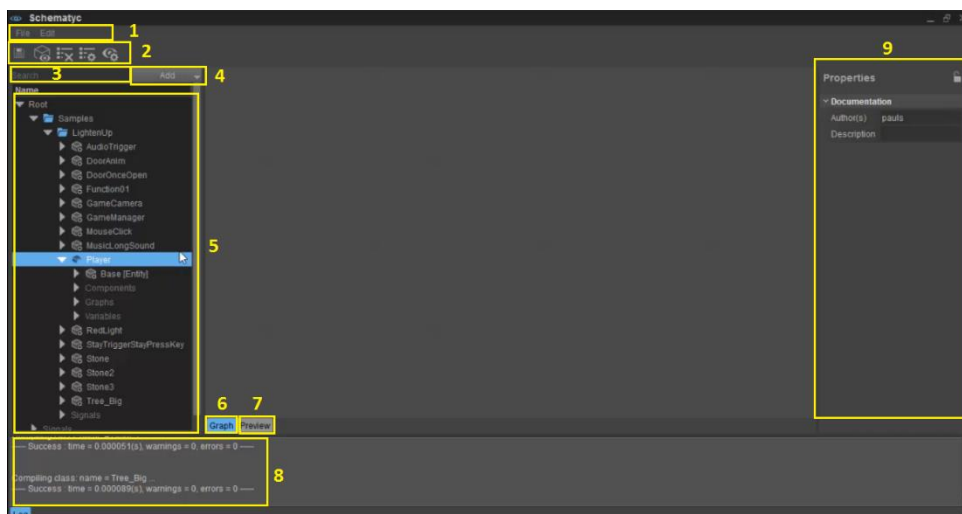
سیگنال های دیگری نیز وجود دارند که خارج از بحث این کتاب است و برای بحث پروگرافینگ در سیماتیک، کتاب جداگانه ای می طلبد.

رابط کاربری سیماتیک

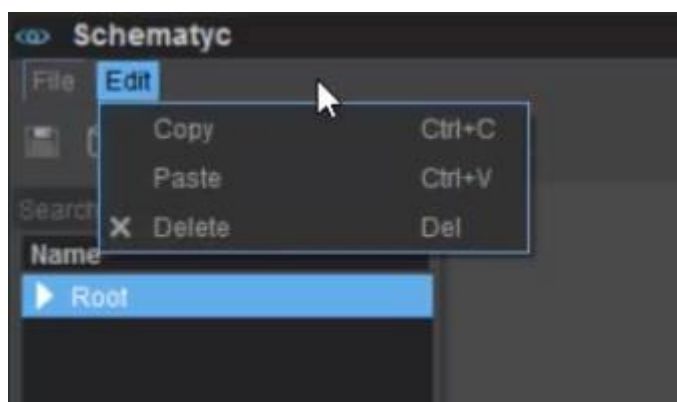
به دنیای شگفت انگیز سیماتیک خوش آمدید، بیشترین توسعه زبان پروگرافینگ سیماتیک را در کرای انجین ۵،۵ شاهد هستیم، این سیستم جدید برنامه نویسی با دریافت بیشترین فیدبک و محبوبیت در بین کاربران جهان است، زیرا که کرایتک بیشترین تمرکز را برای این زبان در نظر گرفته است، در واقع تمرکز API ها برای سیماتیک بسیار بیشتر از زبان C# است، این مهم در نقشه راه شرکت کرایتک در این مقطع زمانی به آدرس زیر کاملاً مشخص است :

www.CryEngine.com/Roadmap

این تصویر رابط کاربری سیماتیک را به بخش ۹ مختلف با کادرهای زرد رنگ در کرای انجین ۵،۳ تقسیم می کند



بخش اول - منوی اصلی که شامل منوی File و Edit تشکیل شده است که برای ذخیره کردن تغییرات در بخش Preview ، Graph است و تکثیر و نمونه برداری از نودهای مختلف است.



در بخش اول شما می توانید نودها را نیز حذف کنید و البته با کلیدهای میانبر صفحه کلید نیز به این گزینه ها در منوی اصلی دسترسی داشته باشید، فراموش نکنید که به عنوان یک طراح بازی هرچه بتوانید با صفحه کلید بیشتر کار کنید، سریع تر می توانید بازی تان را طراحی کنید، این میانبرها به ما قدرت می دهند که زمان کمتری را مصرف کنیم (زمان ارزشش از طلا بیشتر است ۱۰۰۰ سکه طلا نمی تواند ۱۰ دقیقه از عمرتان را برگرداند)

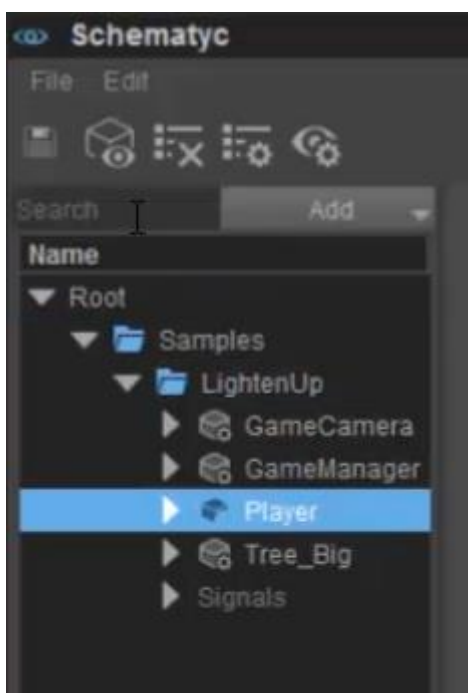
بخش دوم: نوار ابزارها

ابزارهای حیاتی که به ما کمک می کنند رابطه ما با سیماتیک کاربرپسندانه (دوستان صمیمی) باشیم، در این تصویر چشم های کرای انجین وجود دارند، این چشم ها فوق العاده اند و اجازه بدید با این دکمه های چشمی کار کنیم، از سمت چپ به راست دکمه ها را توضیح می دهم و اینجا ۵ دکمه وجود دارد.

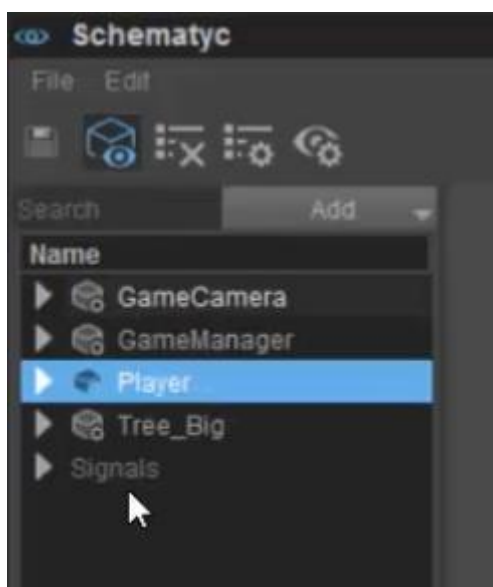


دکمه اول - ذخیره تمام تغییرات : همه کلاس ها را با این دکمه ذخیره کنید، گراف ها، متغیرها، پیش نمایش فضای سه بعدی و کامپونت ها با همین دکمه، سریع و تمیز

دکمه دوم- باز کردن یا بسته شدن لیست تمام کلاس ها بر اساس پوشه ها : اینجا به صورت نمایش لیستی از کلاس ها است، اگر ده ها کلاس که تعدادی کشوی ویژگی و بخش های باز شده باشد، نصف دیگر بسته و نصف دیگر نیمه باز باشد، باعث سردرگمی ما خواهد شد، اگر در بهترین حالت سردرگم نشویم (برنامه نویسان در بین انسان ها کمتر گیج می شوند، چون دودمان ریاضی دانان را در خود دارند)، باعث مصرف زمان ارزشمندمان خواهیم شد، این دکمه به شما در باز و بسته شدن لیست ها با مدیریت پوشه ها را برعهده دارد.



در این تصویر می بینید که جداسازی و تفکیک بندی کلاس ها براساس پوشه های Samples و LightenUp اتفاق افتاده است



در این تصویر نیز می بینید که لیست کلاس های مختلف بدون پوشه ها نمایش داده شده است، اگر تعداد کلاس ها افزایش یابد، دکمه دوم به شدت به ما کمک خواهد کرد.

دکمه سوم-پاک کردن log های متنی : log در واقع از عبارات متنی تشکیل شده است که به شما کمک می کند، مجموعه ای از پیام ها را بر اساس نوع آن مشاهده کنید

۱-پیام هایی که حاوی اطلاع رسانی هستند (Info یا Information) مانند زمان کامپایل پروژه

۲-پیام هایی که حاوی هشدار (Warning) هستند مانند متغیرهایی که تعریف شده اند اما هنوز استفاده نشده اند

۳-پیام هایی که خطاها (Error) را نمایش می دهد و شما برای رفع اینگونه خطاها اقدام کنید مانند خطای تقسیم بر صفر، خارج از محدوده آرایه یا نوع داده

```

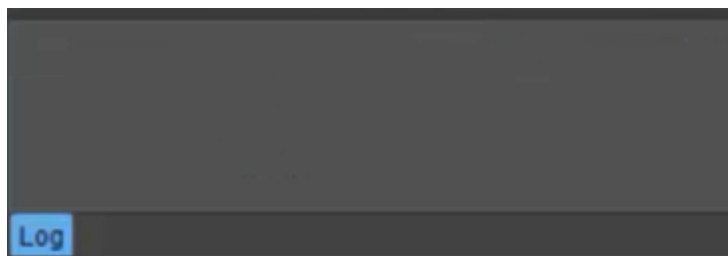
--- Success : time = 0.000565(s), warnings = 0, errors = 0 ---

Compiling class: name = Tree_Big ...
--- Success : time = 0.000439(s), warnings = 0, errors = 0 ---

```

Log

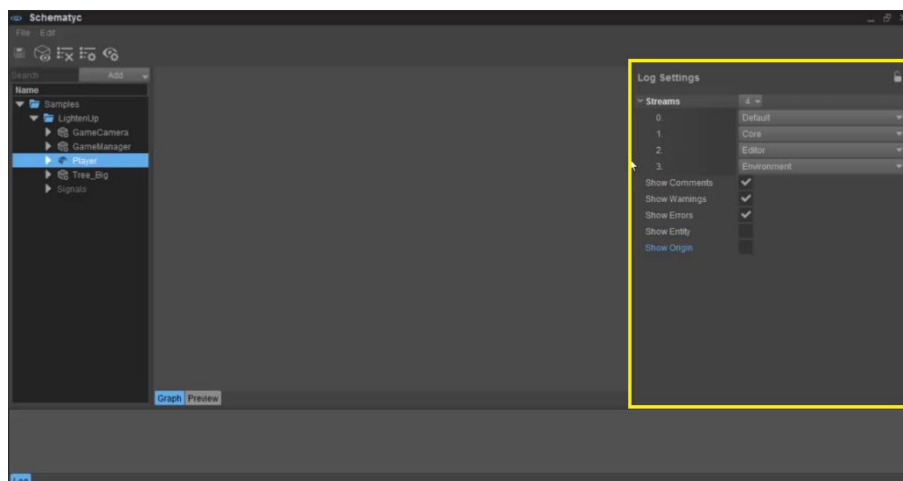
نمایشی از پیام اطلاع رسانی که در تب Log نشان داده شده است



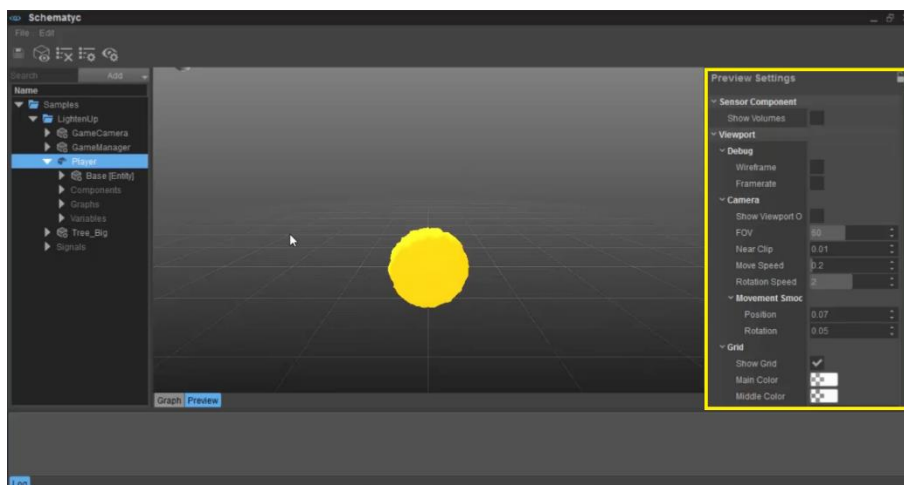
نمایشی از پاک شدن پیام ها در تب Log به وسیله دکمه سوم

۴-نمایش تنظیمات مربوط به log ها در پانل Properties

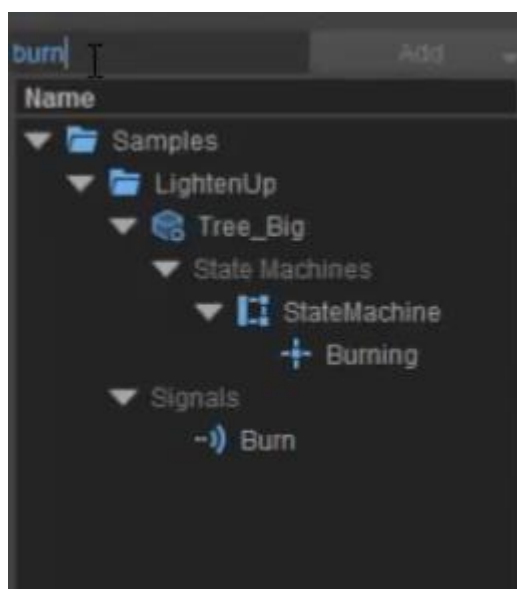
دکمه چهارم : این دکمه جریان و نوع نمایش پیام ها را در تب Log مشخص می کند، شما با توجه به تنظیمات این پنجره به سیماتیک می گوید که چه نوع پیام هایی را با منشاء کدام ابزار، محیط، ادیتور و غیره را نمایش دهد، منشاء این پیام کدام Entity است را برای شما نمایش دهد، در کادر زرد رنگ پنجره تنظیمات log مشخص است.



دکمه پنجم : - نمایش تنظیمات در رابطه با پانل **Preview** : در کادر زردرنگ می بیند که تنظیمات مختلف و متفاوتی از فضای سه بعدی سیماتیک را در اختیار شما قرار می دهد، این تنظیمات شامل نمایش یا عدم نمایش حجم در کامبونت های **SensorVolume**، نمایش توری یا سیمی از مدل ها، نمایش فریم ریت، تنظیمات دوربین طراحی سیماتیک (نه کامبونت های دوربین - اشتباه نکنید)، نحوه کنترل حرکت چرخش کلیدهای صفحه کلید در طراحی سیماتیک است (نه کامبونت های **Input** , **Movement** - اشتباه نکنید، این تنظیمات فقط در فضای سه بعدی سیماتیک مرتبط اند)، این تنظیمات قرار داده شده است تا کاربران کرای انجین بهتر و ساده تر بتوانند در فضای سه بعدی سیماتیک در تب **Preview** جریان کار و طراحی کلاس ها و انتقال کامبونت ها را انجام دهند، این فضا می تواند از رنگ سیاه یا خاکستری به هر رنگ دلخواه دیگری با **Guideline** های سفارشی تبدیل شود، بر روی **DVD** در ضمیمه این کتاب بیشترین توضیحات را در رابطه با این بخش ارائه کرده ام.



بخش سوم - جستجو: تعداد عناصر و المان های کلاس ها و همچنین تعداد کلاس ها می توان بسیار زیاد باشد، سیماتیک نسخه ۲ به زودی منتشر میشود و پوشش دادن و ساخت بازی های بزرگ با سیماتیک نسخه ۲ امکان پذیر است، تعدادی از عناصر و المان ها می تواند نام های مشترک یا غیر مشترکی داشته باشد و در این میان پیدا کردن المان ها در انبار گاه سیماتیک با بخش جستجو به سادگی آب خوردن است، ما از سیماتیک و تیم کرای انجین متشکریم برای اضافه کردن این ابزار فوق العاده، همچنین در تصویر پایین کلمه burn جستجو شده است و نمایشی از سلسله مراتب (ساختار درختی) عناصر با نام burn را می توانید ببیند که در اینجا دو عنصر وجود دارد، عنصر سیگنال داخلی (محلی) و عنصر StateMachine



بخش چهارم - اضافه کردن : اینجا مهمترین بخش است و به قلب سیماتیک وارد شده اید، اینجا شریان های اصلی سیماتیک است و راکتور منابع عظیم کدنویسی در اینجا وجود دارد، مراقب باشید و به دقت این بخش را مطالعه کنید تا دچار سردرگمی نشوید

در تصویر زیر دکمه ای با نام Add (اضافه کردن) وجود دارد، در واقع این فقط یک دکمه نیست، اینجا بیشتر از سه سناریو مطرح می شود ۱- ساخت اشیاء سراسری (خارجی) ۲- ساخت اشیاء محلی (داخلی) ۳- ارتباط اشیاء سراسری و محلی با همدیگر در تصویر زیر ابتدا ساخت اشیاء سراسری را توضیح می دهیم :

اولین گزینه Folder : برای سازماندهی و مدیریت ساده تر و راحت تر اشیاء و مشخص نمودن سلسه مراتب یا ساختار درختی اشیاء کاربرد دارد
دومین گزینه Enumeration : در دنیای ساخت بازی ها ما می توانیم کلمه یا کلماتی را با اعداد صحیح برچسب دار کنیم مثلاً بگوییم $BlueCar=3$ ،

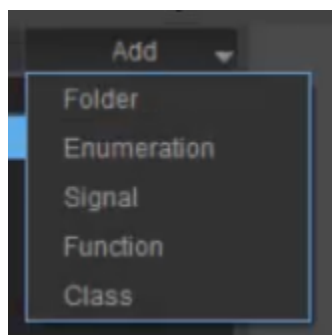
۷=GreenCar باشد و هر زمانی که با مراجعه به آدرس Car.BlueCar در گراف ها عدد ۳ جایگزین شود، تغییر اعداد ممکن است و با تغییر عدد لازم نیست اسم برچسب دار عوض شود، در واقع این مفهوم به مجموعه نام دار عددی نیز معروف است

سومین گزینه Signal : سیگنال ها در واقع همان رویدادهای تابعی یا متدی هستند که بر اساس یک اتفاق کلید می خورند، مثلاً هنگامی که بازی شروع به اجرا شود سیگنال Start و هنگامی که بازی در حال اجرا شدن است سیگنال Update در حال اجراست، این مفاهیم در بالا توضیح داده شده است و برای درک بهتر آن می توانید به ویدئوهای آموزشی بر روی DVD در ضمیمه این کتاب مراجعه کنید

چهارمین گزینه Function : در واقع دو نوع متد و تابع وجود دارند، توابع یا متدهایی که سیستمی هستند و از قبل نوشته شده اند مانند توابع Sinx و Cosx و توابع دیگری هستند که شما آنها را می نویسید مثلاً جمع اعداد اول دو رقمی

پنجمین گزینه Class : در C++ کلاس ها قلب برنامه را تشکیل می دهند (برای بازی سازی نیز همینگونه است) و مفاهیم کلاس ها و کلاس بندی ، مفاهیم برنامه نویسی رویه گرا یا تابع گرا را بلعیدند و در واقع وقتی شما در ذهن تان می گوید که اتومبیلی را با مشخصات مورد علاقه تان را خریداری خواهم کرد، کلاس اتومبیل تان را در ذهن تان ساخته اید، شما اوضاع خوبی در این تحریم ها دارید و اتومبیل مورد علاقه تان را می خرید و حالا شما یک شی به نام اتومبیل دارید، در گذشته شما باید چند صد یا چند هزار خط کد می نوشتید تا با توابع و رویه ها برنامه ای را می ساختید و حالا با چند ده خط کد کلاس را طراحی می کنید و با یک خط کد اختصاص حافظه برای

کلاس شی تان را می سازید،خب سیماتیک لعنتی فقط با چند تا کلیک ماوس این کار را برای شما انجام می دهد.



خدای من این تصویر پایین و تصویر بالا چند گزینه مشترک دارد! اما چرا؟
 ۵ گزینه بالا در تصویر ۵ گزینه پایین نیز وجود دارد؟! خب نگرانی شما به این دو پرسش شبیه هم کاملاً منطقی است و زمان این رسیده است که باید ساخت اشیاء محلی (داخلی) را توضیح دهم و توضیح این بخش برای مبتدیان کمی گیج کننده به نظر میرسد،خب من ساخت اشیاء داخلی و خارجی رو مثل پدر و مادر می دونم،پدر اکثر مواقع خارج از خانه است و مادر نیز اکثر مواقع داخل خانه است،فرزندانی که داخل خانه هستند تحت تاثیر دستورات و راهنمایی های دلسوزانه مادر قرار می گیرند و وقتی که همین فرزندان خارج از خانه هستند این دستورات و راهنمایی های دلسوزانه پدر هست که باید مد نظر قرار داشته شود،این یک مثال سر انگشتی هست،در واقع عناصر و المان های خارجی که در تصویر بالا به صورت ۵ گزینه ای توضیح داده شد،نقش همان پدر را دارند و عناصر و المان هایی که در تصویر پایین وجود دارد،نقش همان مادر را دارد،ارتباطات بین مادر و پدر با این عناصر مفهوم پیدا می کنند،کلیه عناصر و المان های داخلی به صورت محلی (داخلی) و تنها داخل همان شی و یا همان کلاس است و اگر بخواهید بین اشیاء یا کلاس ها در داخل بازی ارتباط برقرار کنید باید از اشیاء یا کلاس سراسری (خارجی)

استفاده کنید،خب من از توضیح مجدد ۵ گزینه بالا در تصویر پایین صرف نظر می کنم و به عناصر داخلی و توضیحات آن می پردازم،لازم به ذکر است وقتی کلاسی را در مفهوم ساخت اشیاء سراسری ایجاد می کنیم،گزینه های پایین برای عناصر و المان های کلاس تان پدیدار می شود



State Machine: میراثی از سیستم برنامه نویسی بصری اولیه با نام FSM بود که در سیماتیک موجود است،این سیستم با نام "وضعیت ماشین به صورت محدود" است که برای هر گره از گراف کدهایی را اجرا می کند

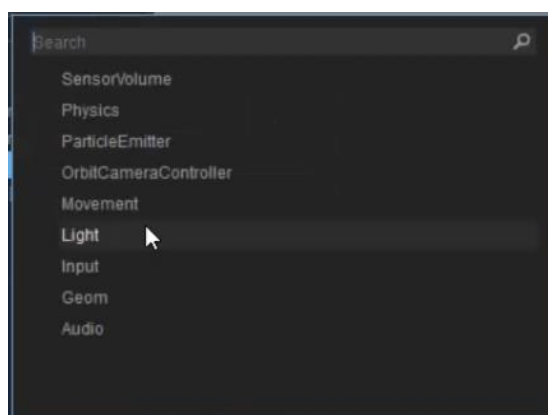
Variable: ساخت متغیرها برای کلاس و شی است،همانطور که از اسم این عنصر مشاهده می شود،متغیر می تواند محتوای داشته باشد و آن محتوا می تواند تغییر کند،البته در سیماتیک نیز می توانیم متغیرهایی را ایجاد کنیم که تغییر نکند که معادل همان ثابت یا Const است،متغیرها می توانند سراسری یا محلی یعنی در داخل فقط شی یا خارج از شی بتوان به آن دسترسی داشت،البته متغیرهایی که به توان خارج از شی به آن دسترسی داشت،مورد تایید برنامه نویسان حرفه ای نیست و به جای آن مفاهیم Property پیاده سازی می شوند،نوع دیگری از متغیرها با نام static شناخته می شوند،متغیر استاتیک یا ایستا توسط همه اشیاء قابل دسترس است،مثلا همه دشمنان

مرحله باید به میزان سلامتی کلاس پلیر در شی پلیر دسترسی داشته باشند، در اینجا میزان سلامتی پلیر یا Health آن باید از نوع static یا ایستا باشد، در برنامه نویسی C++ کرای انجین این مفاهیم را پوشش خواهیم داد.

Timer : اینجا واحدهای زمانی با نام DeltaTime یا FixedDeltaTime به صورت جدا بر اساس سیگنال مربوط جمع می گردند، مثلاً مجموعه ای از DeltaTime ها با جمعش یک تایمر را تولید می کند که در هر ۵ ثانیه یا هر ۳ ثانیه چه اتفاقی را تولید کند، مثلاً یک سکو اسلحه وجود داشته باشد و در هر ۶۰ ثانیه یک جعبه مهمات تولید شود (به شرط آنکه سکو خالی باشد)

Signal Receiver : این بخش به دریافت سیگنال ها اشاره می کند، سیگنال هایی که سفارشی هستند

Component : اجزاء یا جزییاتی بصری و غیر بصری که با ساخت آن Entity ها خلق می شوند، همه ی این عناصر با مشارکت همدیگر Entity های قدرتمند را پدید می آورند (راه برای ساخت بازی ها را هموار کرده ام، سورپرایزها در راه اند...)، انواع کامبونت ها در کرای انجین ۵،۳ شامل عناصر زیر هستند و در کرای انجین ۵،۴ این تعداد بسیار بیشتر شده و در کرای انجین ۵،۵ اوج شکوفایی را در تعداد این کامبونت ها می بینیم، مطمئناً در کرای انجین ۵،۶ همه شگفت زده خواهند شد!



SensorVolume : ناحیه حساس به حجم که هر شی که وارد آن شود به شرط آنکه **Listener** به نوع آن شی حساس شده باشد، سیگنال های **Entering** و **Leaving** عمل ورود و خروج از شی را به صورت شبیه سازی 트리گیری دقیقا مثل **TriggerExit** , **TriggerEnter** را انجام می دهند، مثلا به در نزدیک می شوید، در باز می شود، مثال های ویدئویی آن بر روی **DVD** در ضمیمه این کتاب موجود است

فیزیک : منظور در اینجا فیزیک نیوتونی در کرای انجین است، این بخش نمونه ای اولیه از قوانین فیزیک را به صورت کلاسیک شبیه سازی می نماید، مثلا اختصاص جرم به شی بر حسب واحد کیلوگرم و اختصاص عمل جاذبه به شی بر حسب متر بر مجذور ثانیه

Particle Emitter : این کامبونت را نمی توان از بازی ها جدا دانست، بلکه وجود سیستم ذرات مانند انفجار، آتش، آبشار، مه و غیره در هر بازی لازم است

OrbitCameraController : اختصاص دادن دوربین به هر شی ای که دوست دارید و سپس مدیریت دوربین های مختلف در سطح مرحله را انجام دهید، مثلا در محدوده ای از یک شهر ویران شده هستیم و موجودات فضایی در آن وجود دارند، شما در هر محدوده ای از شهر به دوربین های کنترل ترافیک یا حتی پس کوچه ها دسترسی دارید، چقدر سناریو جذابی است

Movement : بخش دیگری از فیزیک نیوتونی در اینجا نهفته است، حرکت بر حسب متر بر ثانیه را برای هر شی از پلیمر تا دشمنان اختصاص دهید، حرکت دادن اشیاء با این کامبونت امکان پذیر است

Light : این کامبونت با پارامترهای زیادی که در خود دارد، نورهای مختلف، متفاوت و البته انیمیشن دار مثل نورهای چشمک زن و خیلی بیشتر را برای شما به بهترین شیوه شبیه سازی می نماید

Input : مدیریت و کنترل صفحه کلید و ماوس در این کامبونت قرار دارد، هر کلیدی یا کلیک یا چرخش یا نگه داشتن ماوس یا صفحه کلید اتفاق می افتد از وجود این کامبونت نشات می گیرید

Geom : مدل های سه بعدی یا مدل های هندسی به تعبیری دیگر از طریق این کامبونت بارگذاری می شوند، می خواهید مدل هایی مانند ماشین ها، چراغ راهنما و رانندگی، خانه ها و دیگر اشیاء سه بعدی را در مرحله بچینید، این کامبونت به شما کمک می کند

Audio : پخش صداهای مختلف با پسوندهای ogg , wav , mp3 با این کامبونت کاملاً امکان پذیر است و همچنین وجود توقف صدا از ویژگی این کامبونت است، این کامبونت با کلید تنظیماتش با پنجره Audio Controls Editor در ارتباط است.

بر روی DVD در ضمیمه این کتاب توضیحات و مثال های فراوانی برای درک بهتر کامبونت ها به صورت آموزش های ویدئویی موجود است و با مراجعه به اکانت من در **GitHub** نیز می توانید مثال های سی شارپ به همراه آموزش های ویدئویی به صورت کامبونت دریافت کنید

بخش پنجم : لیست تمام کلاس ها، متغیرها، سیگنال ها، تایمرها و غیره که در دکمه Add بالا توضیح داده شد، در این بخش قابل مشاهده است، شما به عنوان طراح بازی مدل های مختلف از منطق بازی طراحی کرده اید و حالا می توانید با استفاده از لیست موجود تمامی آنها را ببینید و این مدل های منطقی را به صورت Entity های مختلف در بازی استفاده نمایید، برای کاربرد و نحوه استفاده کافیسست به پنجره اصلی کرای انجین برگردید و به پنجره Create Object مراجعه نمایید و برروی دکمه components کلیک

نماید و به بخش Schematyc با این نام کاتالوگ رجوع کنید و لیست اشیاء تولید شده را مشاهده کنید، بر روی DVD در ضمیمه این کتاب در این رابطه آموزش های ویدئویی وجود دارد.

بخش ششم : این بخش گراف مربوط به هر کلاس را نشان می دهد و نودهایی را که قرار است متصل کنید یا وجود دارند و متصل شده اند را در این بخش می توانید ببینید، در واقع برنامه نویسی با مفهوم پروگرافینگ در این بخش است و بر روی DVD در ضمیمه این کتاب، آموزش های ویدئویی با مثال های خوب توضیح داده شده و نشان داده شده است

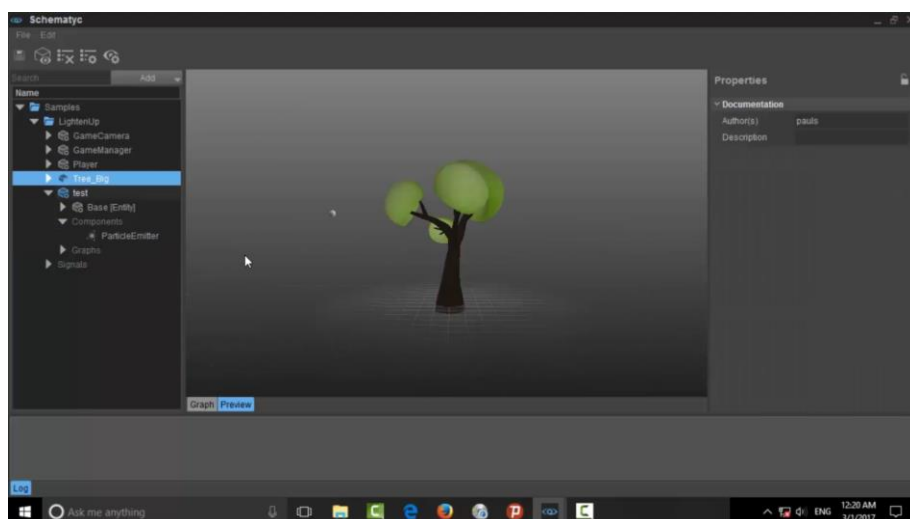
بخش هفتم : در اینجا فضای سه بعدی بی نهایتی را می بینید که می تواند انواع نورها، انواع سیستم ذرات، دوربین و دیگر کامبونت ها را نشان دهد، این بخش برای طراحی بهتر کامبونت و البته طراحی بهتر کلاس ها است تا تمرکز اصلی بر روی طراحی کلاس ها و اجماع کامبونت ها در این بخش انجام شود نه در پنجره پرسپکتیو در سندباکس و همچنین لازم به ذکر است که در صفحات قبلی کتاب این بخش توضیح داده شده است

بخش هشتم : پیام های مختلف از نوع Info , Warning , Error در اینجا نمایش داده می شود، در صفحات قبلی کتاب این بخش توضیح داده شده است

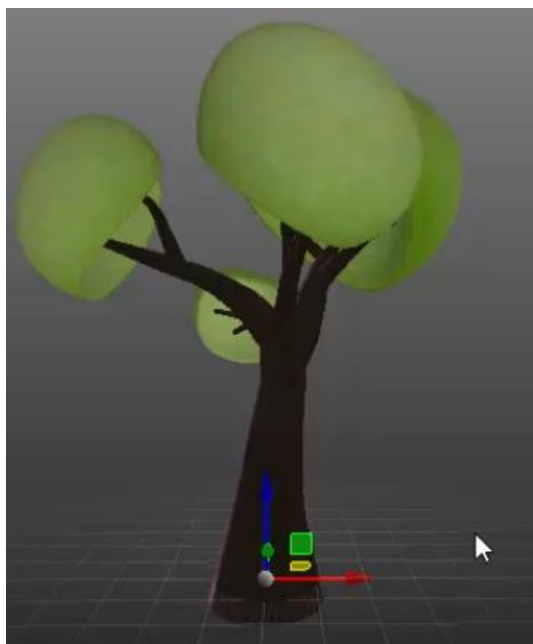
بخش نهم : این بخش برای نمایش خصوصیات، پارامترها، ویژگی ها، اختصاص متغیرها، نوع Enum ها و غیره به کار برده می شود، این بخش نیز بر روی DVD در ضمیمه این کتاب با آموزش های ویدئویی توضیح داده شده است.

کلیدهای میانبر صفحه کلید و ماوس در فضای سه بعدی سیماتیک

حالا نوبت آن است که کلیدهای میانبر صفحه کلید و ماوس که در ادیتور سیماتیک بسیار مهم هستند را به صورت کوتاه، مختصر و صد البته مفید با تصاویر مختلف توضیح دهیم.



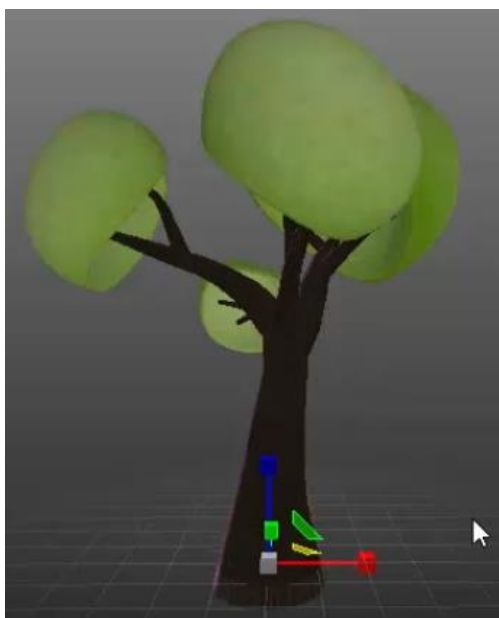
نگه داشتن و حرکت دادن کلیک سمت راست ماوس باعث چرخش دوربین در فضای سه بعدی می شود و کلید W باعث حرکت دوربین به سمت جلو در فضای سه بعدی و کلید S حرکت دوربین به سمت عقب در فضای سه بعدی و کلید D حرکت دوربین به سمت راست در فضای سه بعدی و کلید A حرکت دوربین به سمت چپ در فضای سه بعدی سیماتیک است و کلید ۱ ابزار حرکت و کلید ۲ ابزار چرخش و کلید ۳ ابزار تغییر اندازه را انتخاب و اختصاص می دهد، شکل های پایین نمایش این ابزارهاست



زدن دکمه ۱ بر روی صفحه کلید و انتخاب گیزموس Move در فضای Preview در سیماتیک



زدن دکمه ۲ بر روی صفحه کلید و انتخاب گیزموس Rotate در فضای Preview در سیماتیک



زدن دکمه ۳ بر روی صفحه کلید و انتخاب گیزموس Scale در فضای Preview در سیماتیک

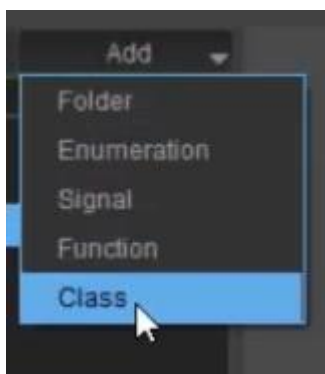
شما در پنجره پرسپکتیو (پنجره طراحی مرحله) در سندباکس نیز می توانید از این کلید ها استفاده کنید
بر روی DVD در ضمیمه این کتاب آموزش های خوبی در رابطه با، کلیدهای میانبر صفحه کلید و ماوس به خوبی توضیح داده شده است

یک مثال از ایجاد نور و دوربین در سیماتیک

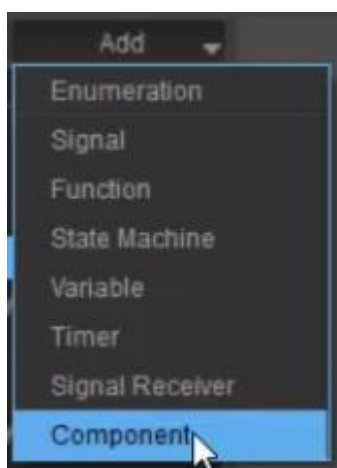
هدف از این مثال نمایش ایجاد Entity های مختلف است که به ساده ترین روشن ممکن با مفهوم جدید Component Entity System است و هر

تعداد Entity با ویژگی های مختلف می توانید بسازید، حالا به سراغ مثال می رویم.

۱- پنجره سیماتیک را از طریق منوی Tools باز کنید و سپس برروی دکمه Add کلیک کنید و گزینه Class را مطابق شکل زیر انتخاب کنید و یک اسم با نام ۰۱ Example به آن اختصاص دهید

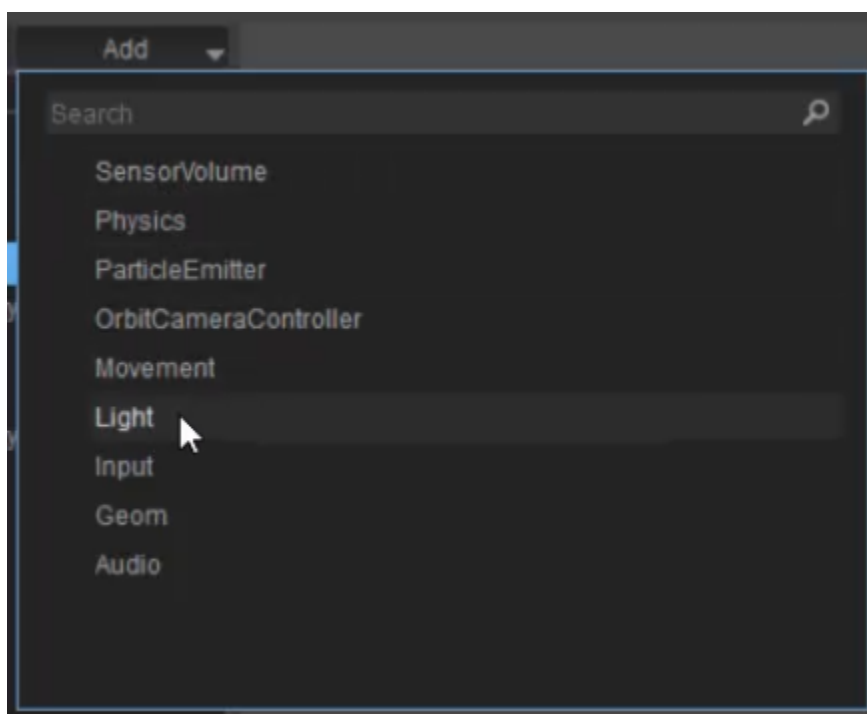


۲- کلاس ۰۱ Example را انتخاب کنید و دوباره برروی دکمه Add کلیک کنید تا کادر کشوی مطابق شکل زیر باز شود و در بین گزینه ها، Component را انتخاب کنید

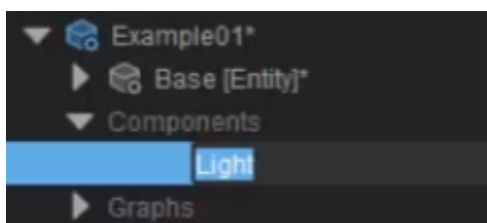


۳- همانطور که می بیند با انتخاب گزینه component لیست کشوی باز شده طبق تصویر پایین و ۹ کامپونت مختلف نمایش داده می شود (برای

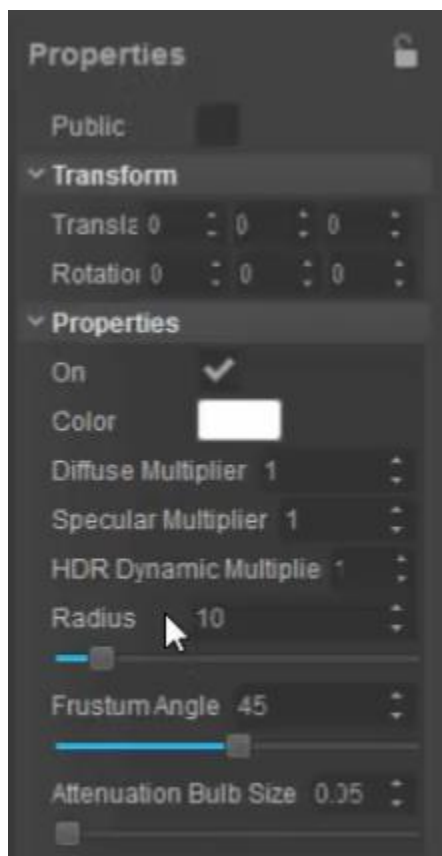
توضیحات این کامبونت ها و دیگر گزینه ها به صفحات قبلی بازگردید) و
گزینه Light را انتخاب کنید



۴- با انتخاب گزینه کامبونت Light مثل آنچه که در تصویر زیر می بینید
ایجاد خواهد شد و حالا باید به پارامترهای کامبونت Light مراجعه کنید و
تغییرات را بر روی این پارامترها انجام دهید.

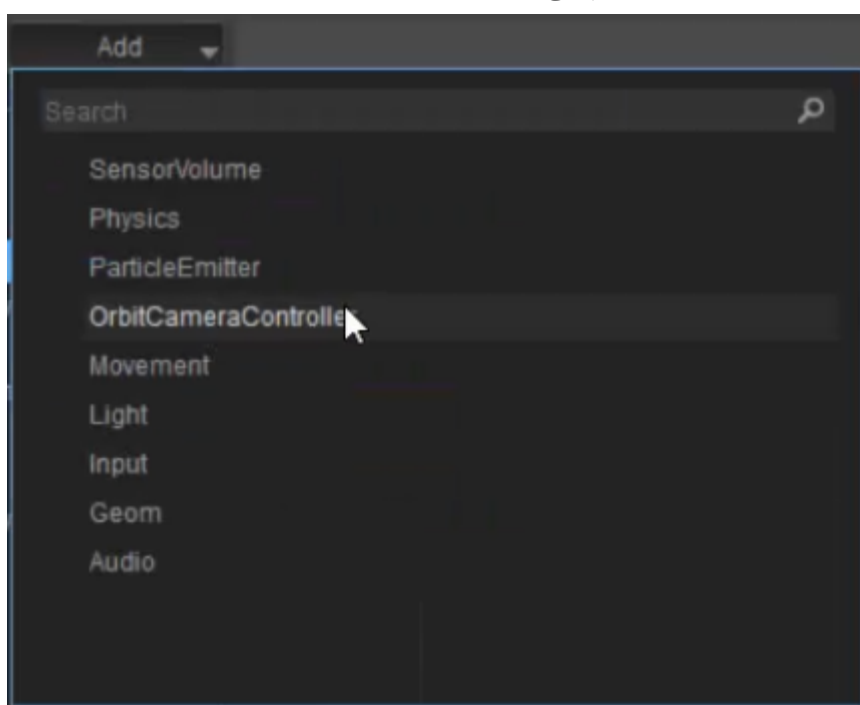


همانطور که در شکل زیر می بینید در پانل Properties ویژگی مختلف کامبونت Light نمایش داده شده است و شما با تغییرات دلخواه و مدنظرتان سفارشی از ساخت نور داشته باشید

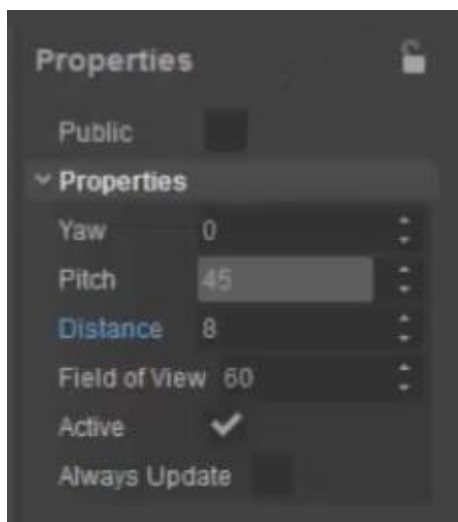


لطفا پارامترهای Diffuse Multiplier را به عدد ۱۰۰ و Radius شعاع نور را به ۶۰ تغییر دهید و Color را به رنگ نارنجی تغییر دهید و بقیه پارامترها را رها کنید (در ساخت اشیاء بعدی توسط خودتان می توانید تغییرات را بروی این پارامترها لحاظ کنید)

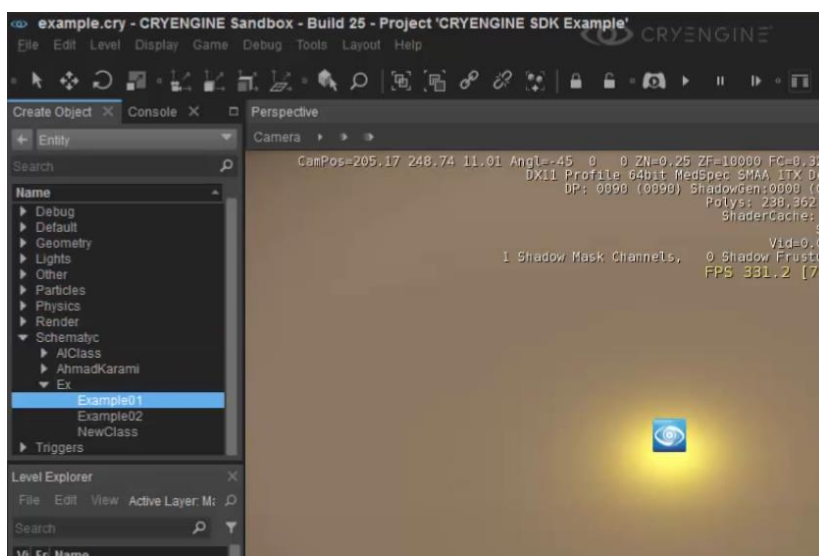
حالا نوبت این هست که کامبونت Camera را اضافه کنید، پس ابتدا کلاس Example ۰۱ را انتخاب کنید و سپس برروی دکمه Add کلیک کنید و با نمایش لیست کشوی گزینه Component را انتخاب کنید و طبق آنچه که در شکل زیر می بینید، گزینه OrbitCameraController را انتخاب کنید و اسم آن را به Camera تغییر دهید (تا جایی که می توانید اسمی کامبونت ها ساده باشند و حداکثر چند کلمه معنی دار باشد که به شما کمک کند، کامبونت ها چه کاری را انجام می دهند)



طبق تصویر زیر در پنجره Properties تغییرات برروی پارامترها را انجام دهید، توضیحات و تشریح پارامترها در کامبونت های مختلف در این کتاب نمی گنجد، تمرکز اصلی این کتاب برروی زبان C++ است



حالا به پنجره SandBox بروید و از پنجره Create Object بر روی دکمه components کلیک کنید و در کاتالوگ schematyc کلاس Example ۰۱ را به مرحله به صورت Drag & Drop اضافه کنید ، یک نور به شعاع ۶۰ متر و میزان و شدت ۱۰۰ واحد نوری ایجاد خواهد شد که یک دوربین در فاصله ۸ متری بر روی این نور فوکوس کرده است.



برای مشاهده توضیحات بیشتر برای رابط کاربری سیماتیک می توانید بر روی DVD در ضمیمه این کتاب مراجعه بفرمایید

دلایل این کار که ابتدا رابط کاربری سیماتیک را در کرای انجین ۵,۳ و سپس رابط کاربری سیماتیک در کرای انجین ۵,۴ را توضیح دادم :

۱- سیماتیک در کرای انجین ۵,۳ شروع شد و برای اولین بار در بازی Hunt به کار گرفته شد، این نسخه آلفا بود و کرایتک منتظر فیدبک های کاربران دنیا از این سیستم جدید بود، من جزء سه نفری بودم که بهترین آموزش های ویدئویی را برای سیماتیک منتشر کردم.

۲- با انتشار کرای انجین ۵,۴ اضافه شدن تعداد زیادی از کامبونت ها و جداسازی کلاس ها از کتابخانه ها در سیماتیک باعث شد روند توسعه آن متوقف شود، کرایتک اعلام کرد ما بررسی مجدد را برای رابط کاربری و به باز طراحی این سیستم پرداخته ایم

۳- تیم توسعه سیماتیک در کرای انجین ۵,۳ با تیم توسعه سیماتیک در بازی Hunt در کرای انجین ۵,۴ جدا بودند و دو تیم متفاوت بودند

۴- کرایتک تصمیم گرفت که تیم سیماتیک کرای انجین ۵,۳ را منحل و تیم توسعه سیماتیک در بازی Hunt را استخدام کند

۵- با انتشار سیماتیک در کرای انجین ۵,۴ ظاهر و کارکرد آن کاملاً فرق داشت با سیماتیک کرای انجین ۵,۳، طبق بخشی از آموزش های ویدئویی من به صورت زیرنویس کوتاه انگلیسی (فقط تصویری) و ترجمه آن به زبان ژاپنی (توسط یکی از دوستان ژاپنی ام)، دسترسی به بخش زیادی از توابع در سیماتیک کرای انجین ۵,۴ غیر ممکن بود چون این سیستم جدید در حال بازطراحی مجدد بود، آموزش های ویدئویی من در سیماتیک کرای انجین ۵,۴ متوقف شد اما اساس آن در کرای انجین ۵,۳ همچنان ادامه دارد و مورد استقبال کاربران جهان قرار گرفته است.

نمایی از پنجره سیماتیک در کرای انجین ۵,۴

در تصویر زیر می بینید که سه کامبونت به نام های `Input` , `Mesh` , `Point Light` اضافه کرده ام

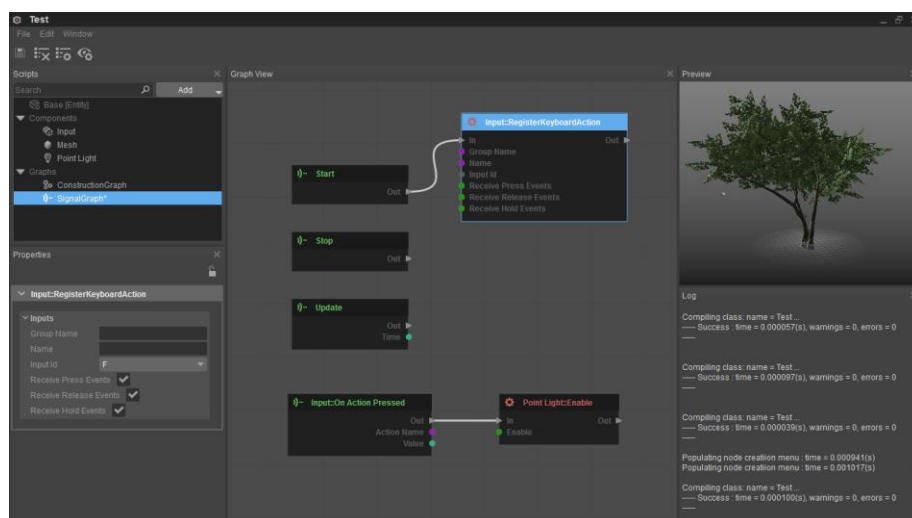
کامبونت `Input` : مدیریت و کنترل دکمه های موجود برروی دستگاه های صفحه کلید، ماوس، `XBox` و `Playstation4`

کامبونت `Mesh` : همان معادل کامبونت `Geom` در سیماتیک کرای انجین ۵,۳ است برای لود و نمایش مدل های سه بعدی

کامبونت `Point Light` : نمایش دادن نور نقطه ای که در اینجا شعاع نور را به ۶۰ و شدت آن را به ۱۰۰ تغییر داده ام

و در ادامه، برای بخش گراف دکمه `F` را با استفاده از نود `RegisterKeyboardAction` ثبت کرده ام و زمانی که بازی شروع می شود دکمه `F` ثبت شده و منتظر رویدادهای فشار، نگه داشتن و رها کردن خواهد بود و زمانی که دکمه `F` برروی صفحه کلید زده می شود نور موجود در کنار درخت با استفاده از نود `Enable` خاموش می شود، من تیک نود `Enable` را برداشته ام.

همانطور که می بیند رابط کاربری سیماتیک در کرای انجین ۵,۴ تغییر خوبی داشته است، اگرچه بیشتر توابع در کامبونت ها از دسترس خارج شده است و در کرای انجین ۵,۵ شاهد انتشار نسخه سیماتیک نسخه ۱ یا حتی نسخه ۲ خواهیم بود که البته با مثال های بهتر تشریح خواهیم کرد

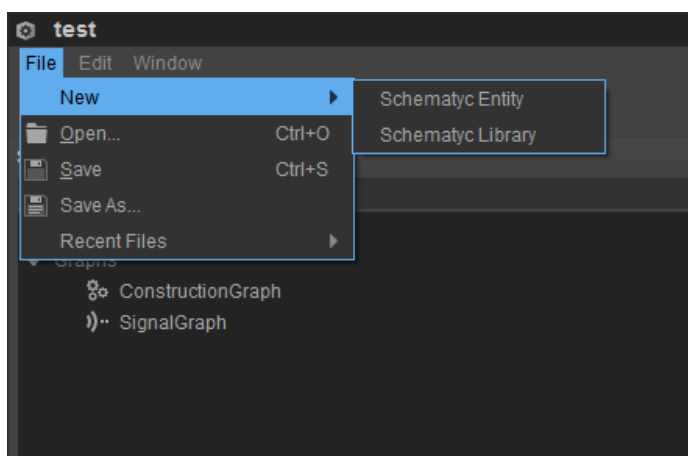


توضیحات در رابطه با تغییرات موجود در رابط کاربری سیماتیک کرای انجین ۵,۴ مفصل است و من به صورت مختصر و مفید آن را برای شما بیان می کنم.

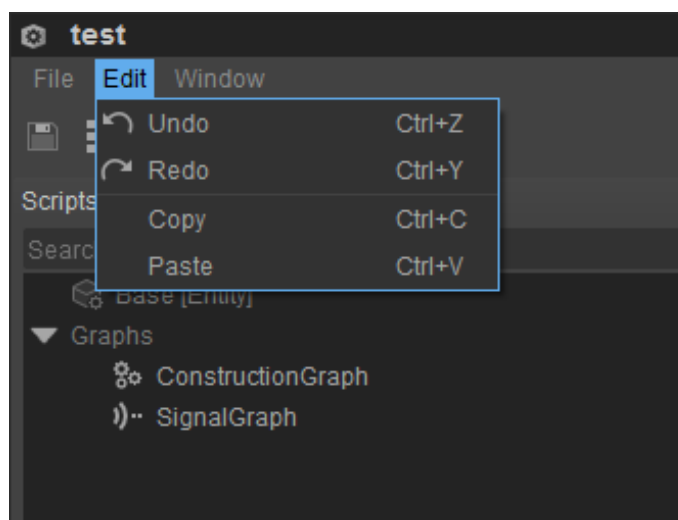
منوها

منوی File با گزینه New اضافه شده است و شما می توانید با استفاده از گزینه Schematyc Entity کلاس های سیماتیک را به صورت entity ایجاد کنید و البته این کلاس ها نیز به صورت پریفب سیماتیکی خواهد بود و گزینه بعدی Schematyc Library است که شما می توانید کتابخانه های سیماتیکی را ایجاد کنید، در واقع کتابخانه سیماتیک نقش همان اشیاء سراسری (خارجی) را دارد و برای ارتباط با کلاس های سیماتیک یا همان Schematyc Entity استفاده می شود و کلاس های سیماتیک یا همان Schematyc Entity نقش کلاس های داخلی (محلی) را برعهده دارند، در این نسخه از سیماتیک جداسازی وظایف با استفاده از این دو گزینه تفکیک شده است، گزینه Open کلاس ها یا کتابخانه هایی که قبلاً ذخیره شده در

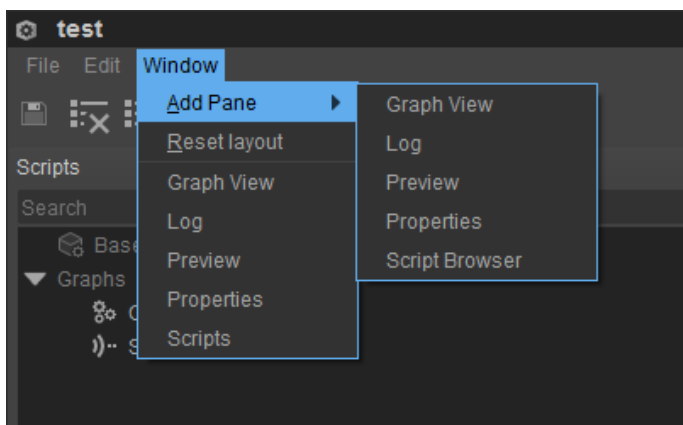
Assets Browser را باز می کند تا اعمال ویرایش، حذف و اضافه کردن المان ها را انجام دهید، با فشردن دکمه های **Ctrl** و **O** برروی صفحه کلید عمل باز شدن نیز انجام می شود و پنجره **Open** نمایش داده می شود، گزینه **Save** فایل کلاسی سیماتیکی یا کتابخانه سیماتیکی جاری را ذخیره می کند و با فشردن دکمه های **Ctrl** و **S** برروی صفحه کلید عمل ذخیره کردن نیز انجام می شود، گزینه **Save as...** فایل جاری سیماتیکی را با نام جدیدی و با عملیات جدید در ذخیره می کند و گزینه **Recent Files** لیست کلاس ها و کتابخانه هایی که اخیرا برروی آن کار کرده اید را نشان می دهد و با کلیک کردن برروی یکی از این فایل ها، فایل کلاس سیماتیکی یا فایل کتابخانه سیماتیکی را برای شما باز می کند.



منوی **Edit** شامل گزینه های **Undo** بازگشت به عقب که با زدن دکمه های ترکیبی صفحه کلید **Ctrl** و **Z** **Redo** عمل بازگشت به جلو با زدن دکمه های ترکیبی **Ctrl** و **Y**، عمل کپی با گزینه **Copy** و با زدن دکمه های **Ctrl** و **C** و عمل چسباندن یا الصاق نمودن با گزینه **Paste** و با زدن دکمه های **Ctrl** و **V** برروی صفحه کلید برای نودهای سیماتیک انجام می شود



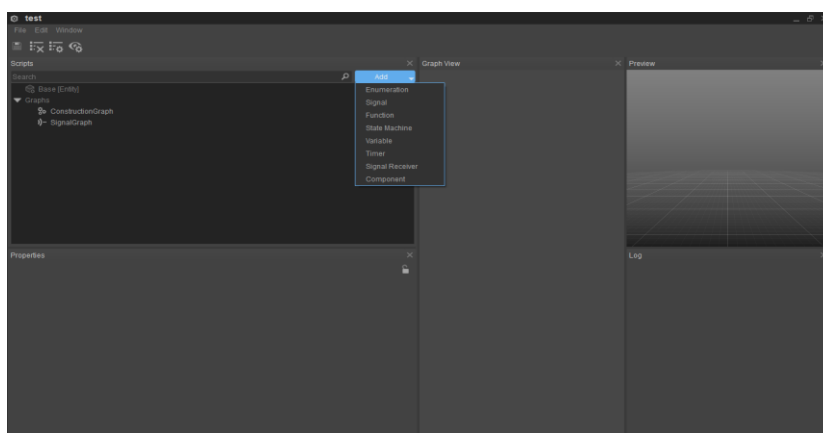
منوی Window : در نمای کلی دیدید که Layout سیماتیک در نسخه ۵,۴ کاملاً تغییر کرده است و در گزینه Add Pane شما می توانید پانل های مختلف را به Layout سیماتیک برگردانید، گاهی وقت ها به صورت عمودی یا غیرعمودی اتفاق می افتد که یکی یا چند پانل را می بینید و بر روی یک پانل کار می کنید برای بازگشت این پانل ها از این گزینه ها در گزینه اصلی Add Pane استفاده می کنید، اسم پانل ها در گزینه های نام برده شده در نمای پنجره سیماتیک مشخص است، گزینه Reset Layout همانطور که مشخص است که Layout سیماتیک را به حالت پیش فرض بر می گرداند و گزینه های دیگری که در پایین گزینه Reset Layout است به معنی باز شدن و وجود این پانل ها در پنجره سیماتیک است و وقتی که هر یک از این پانل ها را ببندید گزینه های مربوط از منوی Window حذف خواهد شد، در صفحات قبلی این فصل به توضیحات کافی برای این پانل ها پرداخته شده است



بررسی ایجاد کلاس (اشیاء داخلی)

همانطور که قبلاً نیز اشاره شد، ایجاد کلاس‌ها در سیماتیک، C# و یا حتی در C++ به معنی ایجاد پریفب‌ها است، در تصویر زیر می‌بینید که شما با ترفندهای مختلف و با امکانات مختلف می‌توانید مکانیزم و عملکرد ایجاد کلاس‌ها را به بهترین نحو ایجاد کنید، قبلاً در رابطه با این گزینه‌ها توضیحات مفصل و کافی ارائه شده است، شما می‌توانید کلاس‌ها را ویرایش کنید، صرف نظر کنید از گراف‌های قدیمی و گراف‌های نو را بکشید و رها کنید و از این سیستم جدید برنامه نویسی حداکثر استفاده را ببرید، در صفحات قبلی این کتاب به توضیحات کافی و مفصل در رابطه با نمای اصلی پنجره سیماتیک پرداخته شده است، وقتی برروی دکمه Add کلیک می‌کنید و لیست کشوی زیر را می‌بیند بدانید که شما در حال ساخت یک کلاس سیماتیکی هستید و بعداً می‌توانید با این کلاس یا هر کلاس دیگری در سیماتیک رفتار پریفب‌ها (Prefab) را ببینید، شما لازم نیست برای استفاده از Entity خاصی هر کار که به آن entity نیاز دارید دوباره آن را در سیماتیک خلق کنید، فقط کافی است یک بار آن را بسازید و هزاران بار از آن استفاده کنید، این یک تعریف برای پریفب است، اگر شما با کدهای C# یا

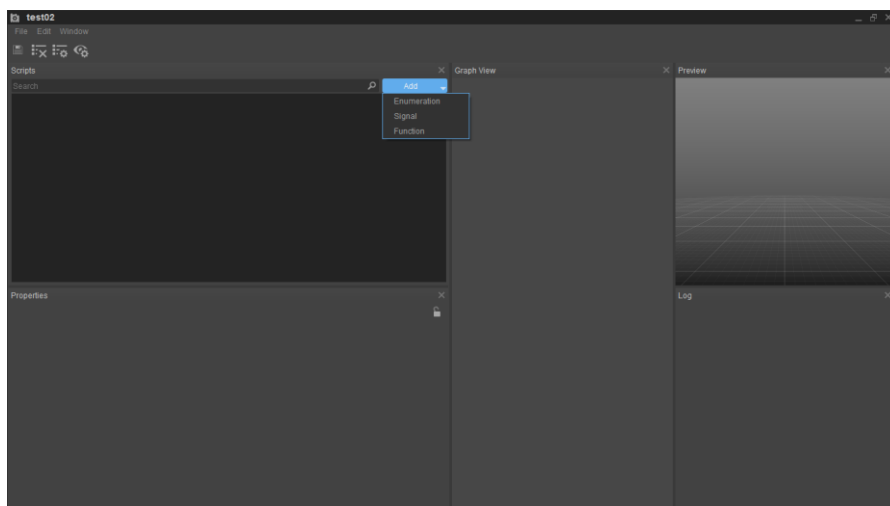
C++ کار می کنید، بعد از کامپایل این کدها، بستگی به نیازتان می توانید کامبونت های C# یا C++ را در این پنجره ثبت کنید و از آن به بعد از این کامبونت ها استفاده کنید، سیماتیک به شما اجازه استفاده از این ترفند زیرکانه را خواهد داد، همه ی این ابزارها و امکانات و خصوصا زبان C++ برای شما اضافه شده است تا کلاس های سیماتیکی خود را هر چه بیشتر توسعه دهید، این قدرت کرای انجین است، فراموش نکنید که کلاس های سیماتیک نمی توانند برای همدیگر پیام بفرستند یا پیام دریافت کنند، در واقع این کلاس ها نمی توانند ارتباط برقرار کنند، اما آیا راه حلی برای این مشکل وجود دارد؟ بله اشیاء داخلی نیاز به استفاده از اشیاء خارجی دارند و این راه حل در صفحه بعدی توضیح داده شده است.



بررسی ایجاد کتابخانه (اشیاء خارجی)

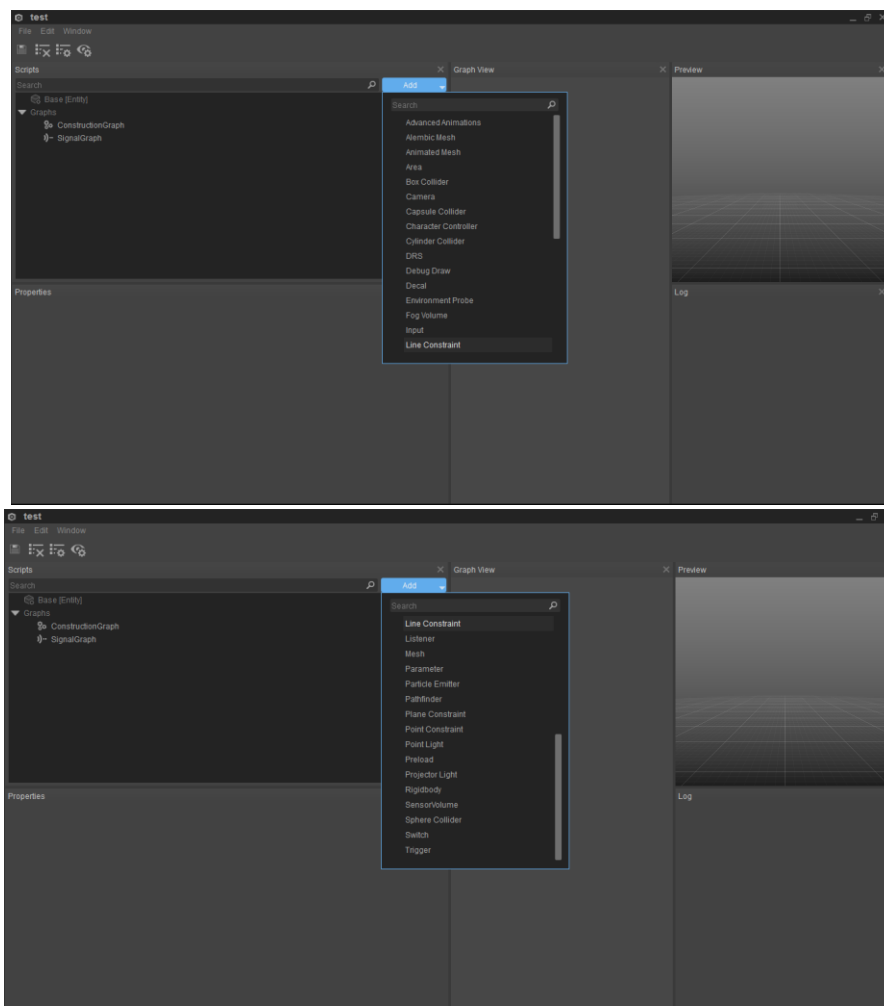
همانطور که در شکل زیر می بیند راه های ارتباطی مختلفی برای کلاس های سیماتیک وجود دارد، اگر تصویر زیر را در جایی دید بدانید که شما در حال ساخت یک کتابخانه هستید و از این کتابخانه می توانید داخل کلاس های سیماتیک استفاده کنید، قبلا در رابطه با گزینه های مختلف در صفحات قبلی توضیح مفصل و کافی ارائه شده است، مثلا شما دو شی از سیماتیک دارید و

این دو شی با استفاده از دو کلاس در سیماتیک ساخته شده اند، این دو شی مسافت همدیگر را از هم باید بدانند، مطمئناً این راه حل با استفاده از سیگنال های مختلف قابل پیاده سازی است و نیاز است از یک سیگنال سراسری استفاده کنیم، اما باید منتظر شویم که سیماتیک نسخه ۱ یا نسخه ۲ به صورت کامل منتشر شود.



بررسی کامبونت های کرای انجین ۵,۴

با انتشار کرای انجین ۵,۴، سیماتیک نیز توسعه یافت اما سیکل توسعه موقتاً متوقف شد و با بازطراحی مجدد آن انتظار می رود که بتوان بیشترین بازی های ساده تا نسبتاً پیچیده را با سیماتیک نسخه ۲ در کرای انجین ۵,۵ و یا احتمالاً در کرای انجین ۵,۶ ساخت و حالا به بررسی کامبونت های سیماتیک کرای انجین ۵,۴ می پردازم :



Advanced Animation : این کامبونت هنوز با سیستم انیمیشن در سیماتیک مجتمع سازی نشده است و ارتباطی با سیستم مدیریت انیمیشن در Mannequin Editor ندارد.

Alembic Mesh : نوعی دیگر از انیمیشن های موجود در مدل های سه بعدی است که انتظار می رود با سیستم انیمیشن کرای انجین مطابق و پشتیبانی شود، این کامبونت نیز هنوز تکمیل نشده است

Animated Mesh : انیمیشن های پایه و ساده از مدل های سه بعدی در این کامبونت مدیریت می شود، این کامبونت نیز هنوز با سیستم انیمیشن در کرای انجین تکمیل نشده است

Area : رفتار این کامبونت نامشخص است و به یک ناحیه اشاره می کند و تنها یک تابع برای فعال و غیر فعال شدن دارد، این طور به نظر می رسد که این کامبونت نیز در حال تکمیل یا در حال باز طراحی مجدد است

Box Collider : صرف نظر از **Softbody** ها (اشیاء نرمی که جاذبه دارند مانند بادکنک) که با سیستم پارچه یا با همان **Cloth** نیز پیاده سازی می شود، اشیاء سخت که **Rigidbody** نام دارند و جاذبه دارند مانند تکه سنگ یا توپ تنیس برای اینکه در داخل زمین فرو نروند و بر روی زمین قرار گیرند، باید بدنه ای سفت و سخت برای آن اشیاء شبیه سازی کرد که با نام **Collider** ها شناخته شده اند، نوع این بدنه سفت و سختی که می تواند از نوع اشکال ساده هندسی تا پیچیده باشد که در این کامبونت نوع بدنه سفت و سخت یا همان **Collider** از نوع **Box** یا جعبه است که داری طول ، عرض ، ارتفاع است

Camera : برای ساخت دوربین یا دوربین هایی در سطح مرحله از این کامبونت استفاده می شود

Capsule Collider : نوع بدنه سفت و سخت از نوع کپسول است مانند یک شی "کپسول گاز آتش نشانی"

Character Controller : نوع بدنه سفت و سختی که می تواند رفتار کاراکتر یا همان رفتار پلیر را شبیه سازی کند

Cylinder Collider : نوع بدنه سفت و سخت از نوع سیلندر باشد، هر مدلی که به شکل سیلندر یا شبیه سیلندر باشد

DRS : این کامبونت هنوز تکمیل نشده است اما در ارتباط با ادیتور

Dynamic Response System است

Debug Draw : این کامبونت برای ساخت پیغام های متنی دو بعدی

برروی صفحه نمایش و پیغام های سه بعدی در جهان بازی است، همچنین

تعیین محدوده نمایش این پیغام ها را کنترل می کند

Decal : نمایش یک متریال به صورت حجمی برروی یک سطح از

مدل، مانند جای ترمز ماشین برروی جاده یا پاشیدن خون برروی دیوار

Environment Probe : نورپردازی ریل تایم پیشرفته در محدوده یک

ناحیه یا حجم که با پارامترهای مختلف کنترل می شود، یک مثال کوچک از

پارامترهای این کامبونت میزان روشنی و تاریکی سایه ها در شب یا روز است

Fog Volume : برای نمایش دادن مه و ایجاد یک محدوده یا یک ناحیه یا

حجمی از مرحله که در داخل مه فرو رفته باشد

Input : کنترل و مدیریت دکمه های دستگاه ماوس، دکمه های دستگاه

صفحه کلید، دکمه های دستگاه Xbox ، دکمه های دستگاه

PlayStation۴

Line Constraint : برای شبیه سازی نوع خاص از رفتارهای فیزیکی اشیاء

است

Listener : شما برای شنیدن اصوات در بازی نیاز به این کامبونت دارید، این

کامبونت باید در کنار کامبونت دوربین پلیر اضافه شده باشد

Mesh : برای لود و استفاده از مدل های سه بعدی که ایستا و ثابت هستند

باید از این کامبونت استفاده نمود

Parameter : هنوز مشخص نیست این کامبونت چه کاری را انجام می دهد

اما در ارتباط با پنجره Audio Controls Editor است و شامل یک نود به

نام Set است

Particle Emitter : برای لود کردن و نمایش سیستم ذرات مانند

آتش، دود، بخار و ... استفاده می شود

Pathfinder : این کامبونت هنوز تکمیل نشده است و در رابطه با مسیری از

نقاط مختلفی که برای پیدا کردن هدف در الگوریتم های هوش مصنوعی در

نظر گرفته شده است، انتظار می رود با انتشار کرای انجین ۵,۵ با سیماتیک

نسخه ۱ یا نسخه ۲ کامبونت های هوش مصنوعی تکمیل شود

Plane Constraint : برای شبیه سازی نوع خاص از رفتارهای فیزیک

اشیاء است

Point Constraint : برای شبیه سازی نوع خاص از رفتارهای فیزیک اشیاء

است

Point Light : برای شبیه سازی نورهای نقطه ای مانند نور لامپ استفاده

می شود

Preload : هنوز مشخص نیست این کامبونت چه کاری را انجام می دهد اما

در ارتباط با پنجره **Audio Controls Editor** است و هیچ نودی ندارد

Projector Light : برای شبیه سازی نورهای مخروطی مانند نور چراغ قوه

استفاده می شود

Rigidbody : به اشیاء سختی که دارای جاذبه و شتاب باشند و همچنین

جاذبه در دنیای کرای انجین به صورت برداری است و کنترل باد بر روی اشیاء

و اثر باد بر روی اشیاء کاملاً امکان پذیر است و حتی تاثیر باد بر روی سیستم

ذرات مانند آتش و دود و غیره با پارامترهای x, y, z را می توان پیاده کرد

SensorVolume : برای شبیه سازی محدوده ای از یک شی مانند در اتاق

که وقتی به در اتفاق نزدیک شدیم در باز شود و وقتی از در اتاق دور شدیم

در بسته شود، این محدوده با نام **Trigger** در کدهای برنامه نویسی بازی

سازی نیز شناخته می شوند

Sphere Collider : نوع بدنه سفت و سخت می تواند به شکل یک کره با

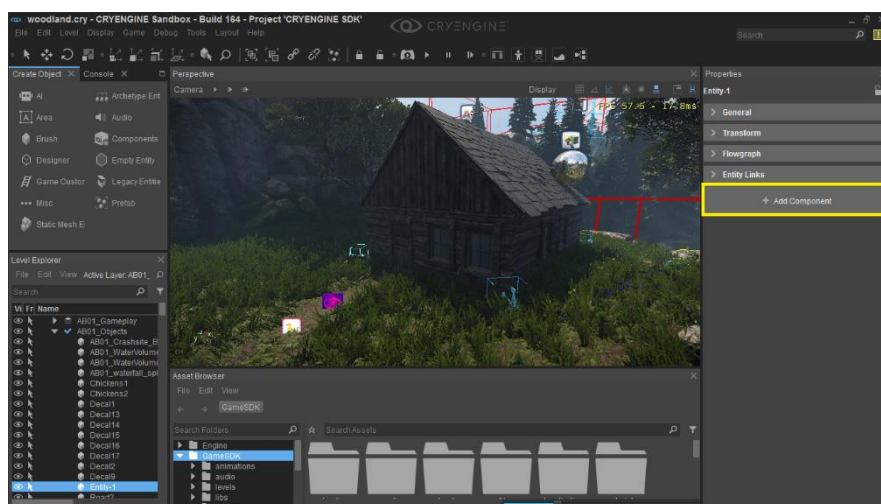
یک شعاع مشخص باشد مانند توپ فوتبال

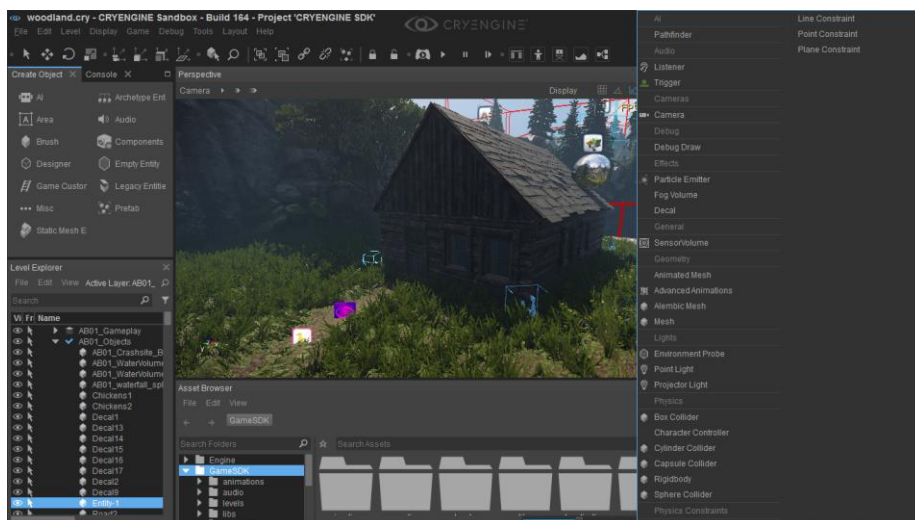
Switch : هنوز مشخص نیست این کامبونت چه کاری را انجام می دهد اما

در ارتباط با پنجره **Audio Controls Editor** است و شامل یک نود به نام **Set** است.

Trigger : برای پخش صدا یا توقف صدا از این کامبونت استفاده می شود و این تریگر با تریگر کدهای برنامه نویسی متفاوت است.

مطمئنا با انتشار کرای انجین و نمایش سیماتیک نسخه ۱ یا سیماتیک نسخه ۲ تغییرات عظیمی را در این کامبونت ها و توسعه این کامبونت ها شاهد خواهیم بود و کامبونت های متنوع و جالب دیگری نیز به سیماتیک اضافه خواهند شد، کار کردن با نودهای سیماتیک بسیار ساده است اما باید منتظر باشیم که ساخت بازی مانند کرایسیس ۳ با نودهای سیماتیک اساسا امکان پذیر است؟ سیماتیک هنوز به این درجه از توانمندی نرسیده است و با انتشار نسخه کرای انجین ۵,۵، امیدوار باشیم ساخت بازی مانند کرایسیس ۳ با سیماتیک محقق شود، زمان به ما نشان می دهد این سوال، حتما جوابی خواهد داشت، فعلا C++ را یاد بگیرید





همانطور که در تصاویر بالا نیز مشخص است با اضافه کردن یک شی خالی (Empty Entity) در پنجره Create Object و سپس با کلیک کردن بر روی دکمه Add Components در کادر زردرنگ، کلیه کامپونت های کرای انجین برای ما نمایش داده خواهد شد، شما نیز می توانید با زبان های C++ کامپونت های خود را به صورت خیلی سریع و ساده بسازید، فراموش نکنید که ساخت کامپونت ها با زبان C# نیز امکان پذیر است اما قدرت C++ بسیار بالاتر از C# است، هنوز API های مختلف مانند هوش مصنوعی برای C# تکمیل نشده است و کلیه کتابخانه ها از DLL های ساخته شده از C++ در C# استفاده می شود و این نشان می دهد که Schematyc و C# به شدت به زبان قدرتمند C++ وابسته هستند، جالب است بدانید که بالای ۸۵ درصد کرای انجین ۵،۴ با زبان C++ طراحی شده است و دوباره توصیه می کنم که از زبان C++ در کرای انجین استفاده کنید و باید صبر کنید تا توسعه سیماتیک حداقل در نسخه ۱ به اتمام برسد، این امیدواری وجود دارد که در کرای انجین ۵،۵ حداقل نسخه ۱ سیماتیک منتشر

شود، اگرچه هنوز توسعه کامبونت ها به اتمام نرسیده اما راه طولانی برای اتمام توسعه سیماتیک نیز نمانده است.

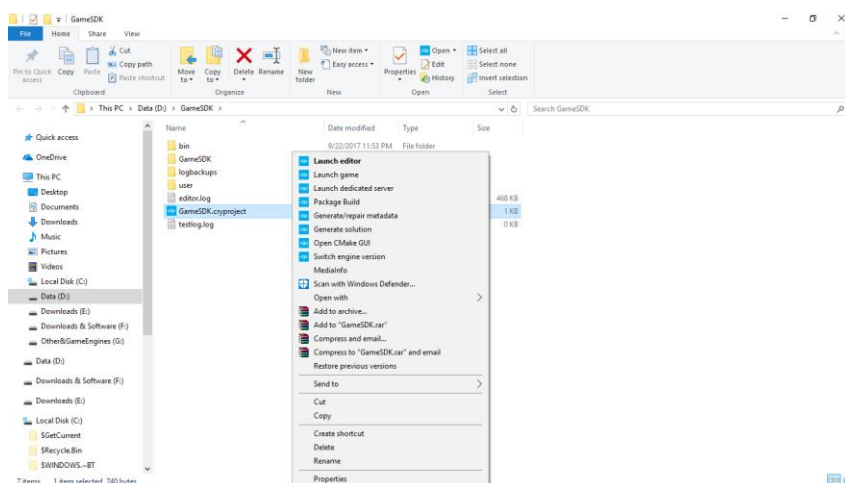
فصل دوازدهم

خروجی گرفتن از بازی و

آشنایی با گزینه های مختلف Launcher

خروجی گرفتن از بازی و آشنایی با گزینه‌های مختلف Launcher

تقریباً ساخت بازی تمام شده است و به مرحله پایانی از داستان پر فراز و نشیب بازی سازی نزدیک شدیم، ساخت بازی یکی از مهمترین کارهای دنیا است که هر شخص و گروهی توان انجام این مهم را ندارد یا ندارند، در پازل های کوچک چیدن تکه های بزرگ کار ساده ای است اما وقتی پازل بزرگ شود، تکه های آن نیز کوچک می شوند و آنگاه فرایند ساخت بازی طولانی تر می شود، ساخت بازی های ۲،۵ بعدی و ۳ بعدی بسیار بیشتر و بیشتر از یک تخصص است، ساخت بازی ها با سناریو ها می تواند بسیار پیچیده شوند، من در تلاش بودم که سناریوهای ساده ای برای پازل کوچک این کتاب تدارک ببینم، مطمئناً این پایان کتاب نیست، بلکه آغاز یک تحول و انقلاب در ساخت بازی های ویدئویی با تکنولوژی کرای انجین برای توسعه دهندگان ایرانی است، بگذارید در اینجا تکه مانده از پازل کوچک این کتاب را بچینم و پازل زیبای کرای انجین را تمام کنم، در تصویر پایین با راست کلیک کردن بر روی فایل GameSDK.cryproject در پروژه GameSDK یا هر پروژه ای که دارای فایل با پسوند cryproject* است منویی ظاهر می شود و شامل گزینه های مختلف است، من این گزینه ها را به ترتیب از بالا به پایین تشریح می کنم

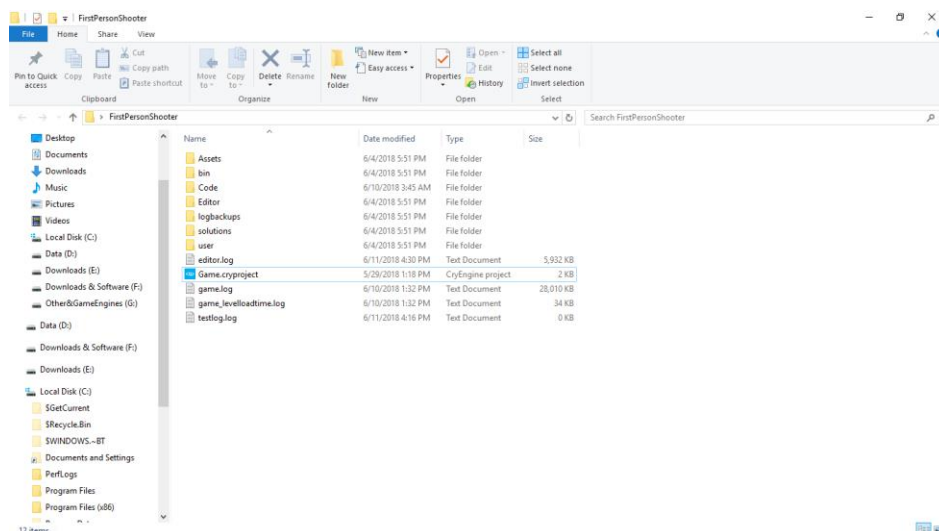


Launch Editor: محیط Sandbox برای انجین یا همان ادیتور برای انجین باز می شود

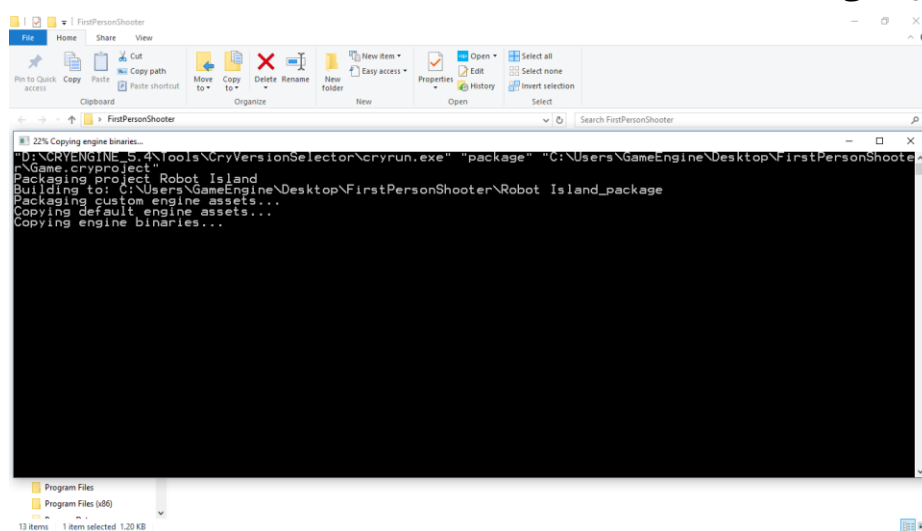
Launch Game: بازی در حال ساخت در یک پنجره باز می شود

Launch dedicated server: اگر بازی تحت شبکه باشد، نیاز است از این گزینه استفاده کنید و اختصاص سرور و IP ها برای بازی تان در شبکه انجام شود

Package Build: برای خروجی گرفتن و ساخت پروژه بازی و همچنین برای انتشار و فروش بازی بر روی DVD / Flash و دیگر رسانه های ذخیره سازی از این گزینه استفاده می شود، در واقع لانچر و کرای انجین پوشه ای را ایجاد می کنند و پروژه بازی شما را کامپایل و تفسیر می نمایند، پوشه ایجاد شده با اسم بازی تان را می توانید به صورت رایگان یا با پرداخت هزینه منتشر کنید



تصویر بالا قبل از کامپایل و ساخت پروژه بازی است و در تصویر پایین می بینید که با راست کلیک کردن برروی فایل Game.cryproject و انتخاب گزینه Package Build، پروژه شروع به کامپایل می شود و پنجره سیاه رنگی با نمایش روند کامپایل و گزارش از پیشرفت پروژه تان را به انتظار دعوت می کند.

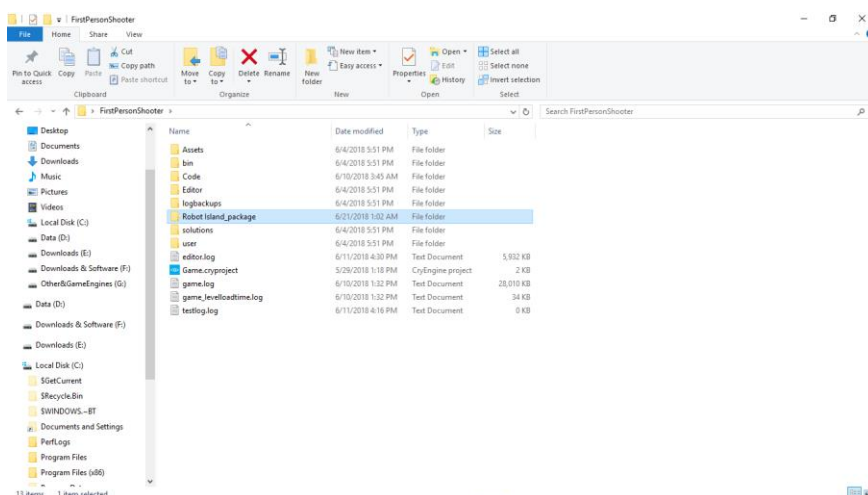


در تصویر پایین بعد از چند دقیقه انتظار، پروژه کامپایل می شود (هر چه پروژه بازی شما بزرگتر باشد، زمان انتظار برای کامپایل پروژه بیشتر خواهد بود) و گزارشات تکمیلی در رابطه با نام پروژه، اسم بازی، مسیر کامپایل شده، نحوه کامپایل Asset ها، حجم پروژه، تاریخ اتمام کامپایل و تاریخ ساخت بازی، تعداد هشدارها، تعداد خطاها، زبان های پشتیبانی شده و غیره را نشان می دهد

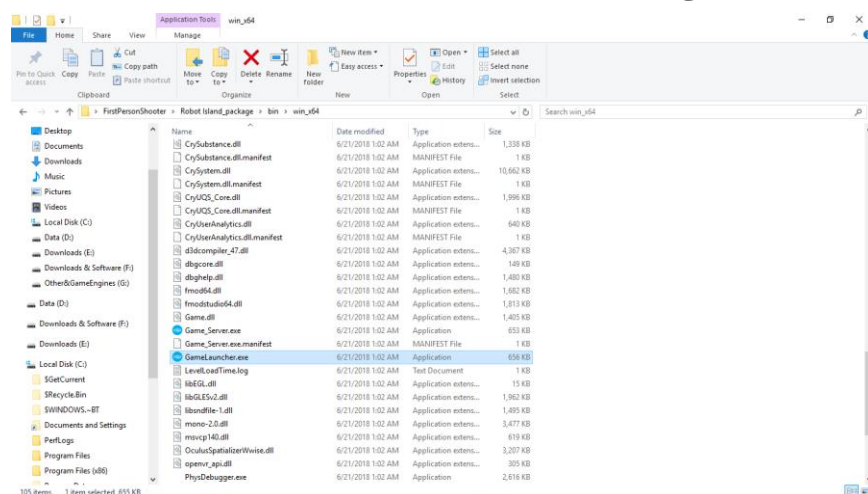
```

Select 100% Build packaged successfully
settypes=cgi.Meshchr.Skeleton;skin.SkinnedMesh;dds.texture;mtl.Material;css.AnimatedMesh;anim.MeshAnimation;cdf
.Character;caf.Animation;wav.Sound;ogg.Sound;cax.GeometryCache;lua.Script;pfx.Particles;xml.Xml;cry.Level(prop
erty)/brdfglossbias=0 (property)/brdfglossscale=16 (property)/cc_email=martins@crytek.de;timur@crytek.de;ser
gey@crytek.de (property)/ce_edgefx=0 (property)/inasecompressor=Cisqualish (property)/mailserver@mail.inte
n.crytek.de (property)/pcPC (property)/skipping= (property)/sourceroots=/Assets/Localization (property)/
stripmetadate (property)/targetroot=C:\Users\GameEngine\Desktop\FirstPersonShooter\Robot_Island_package\Asset
s/Localization (property)/vertexindexformat=us32 (property)/xmlfilterfiles=D:\CRYENGINE_5.4\tools\rc\xmlfilter
.txt (property)
0:29 Scanning directory 'C:\Users\GameEngine\Desktop\FirstPersonShooter\Assets\Localization\korean.xml' for
'.x'
0:29 RC can't find files matching 'korean.xml/x.x', 0 files converted
-----
0:29 Cleaning target folder ..\rc_PC_pure
0:29 0 entries (0 unique) found in list of processed files 'D:\CRYENGINE_5.4\Tools\rc\rc_outputfiles.txt'
0:29 423 entries in list of processed files
0:29 Saving D:\CRYENGINE_5.4\Tools\rc\rc_outputfiles.txt
0:29 Saving D:\CRYENGINE_5.4\Tools\rc\rc_deletedfiles.txt
0:29 Unloading CryPhysicsRC.dll
Memory: working set 39.1Mb (peak 401.4Mb), pagefile 8.0Mb (peak 368.8Mb)
Finished at: Thu Jun 21 01:02:50 2018
2 errors, 0 warnings.
Cleaning up temp folders...
Creating config files...
Build packaged successfully
Press Enter to exit
  
```

در تصویر پایین پوشه ای با نام بازی Robot_Island_Package ایجاد شده است و این همان بازی است که شما قرار است، آن را منتشر کنید و به شرکت ها برای فروش عرضه کنید (پیام من این است که خواهش میکنم از ساخت بازی های بی کیفیت و رایگان خودداری کنید، پیام من را به دیگران نیز انتقال دهید)



در مسیر پوشه همانطور که مشخص است فایل بازی با نام **GameLauncher.exe** در انتظار اجرا شدن است، بر روی فایل دوبار کلیک کنید (امشب برای این پیروزی بزرگ جشن بگیرید و از شرکت کرایتک - تیم توسعه و برنامه نویسی کرای انجین سپاسگزار باشید)

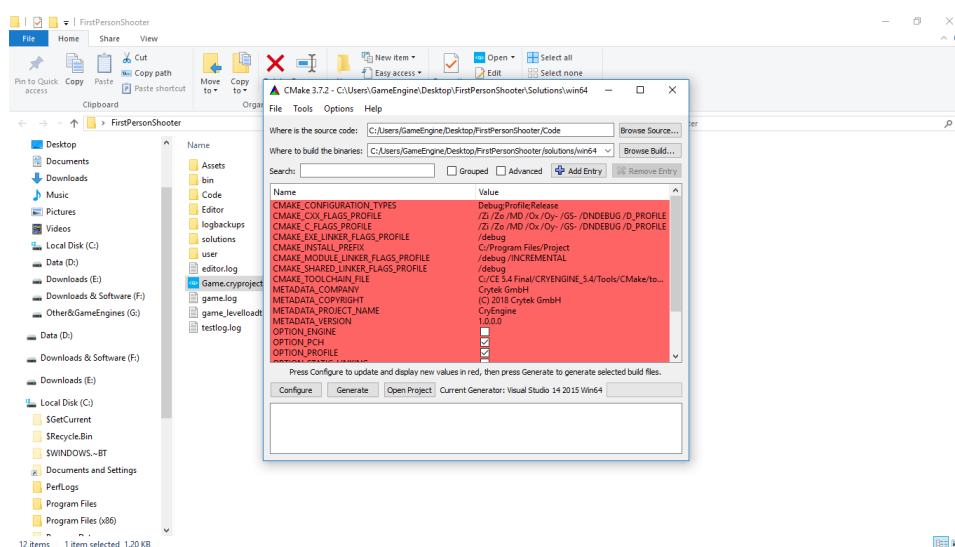


Generate/repair metadata: گاهی وقت ها ممکن است که شما Asset هایی را به پروژه اضافه کنید اما کرای انجین به درستی این asset

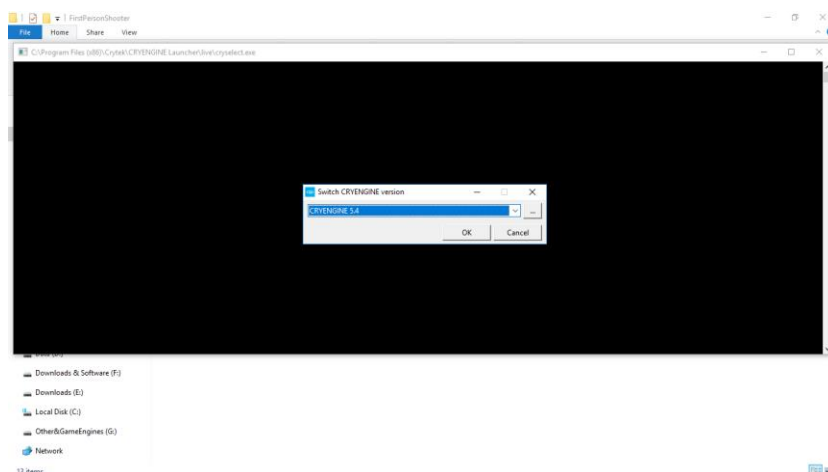
ها را کامپایل و تفسیر نکند، این گزینه به شما کمک می کند، انجام مجدد کامپایل برای asset هایتان مقدور شود و متفایل های تولید شده تعمیر و اصلاح یا مجدداً تولید شوند، با وارد کردن هر asset به کرای انجین وجود فایل وارد شده و تولید متفایل برای آن فایل وارد شده، باعث استفاده کردن آن asset در پروژه می شود، کرای انجین از متفایل ها برای مدیریت و کنترل و ویرایش asset ها استفاده می کند

Generate solution: این گزینه در فصل های گذشته توضیح داده شده است

Open CMake GUI: رابط کاربری را نشان می دهد که شما باید پروژه C++ را در بازی تان ایجاد کرده باشید، برای کامپایل و تغییرات مورد نظر تان در پروژه C++ نیاز است از این رابط کاربری استفاده کنید، این رابط به شما کمک می کند که دسترسی به کدهای Sandbox را با ویرایش مجدد انجام داده و عمل کامپایل بر روی نه تنها بازی تان بلکه بر روی کل تکنولوژی کرای انجین برای Sandbox داشته باشید



Switch engine version: شما می توانید همزمان از چندین نسخه کرای انجین استفاده کنید، این گزینه به شما کمک می کند که در پروژه تان کدام نسخه از کرای انجین لود شود، مثلاً نسخه ۵,۳ یا ۵,۴؟ با انتخاب این گزینه یک کادر محاوره ای باز می شود و شما می توانید نسخه کرای انجین تان را انتخاب کنید



حتماً کرای انجین ۵,۴ را دانلود کنید، قطعاً این یک انقلاب در زمینه طراحی و برنامه نویسی بازی ها است، من هنوز به بررسی و توضیح برنامه نویسی پایتون در کرای انجین نپرداخته ام، بازی های تحت شبکه را تشریح نکرده ام، مباحث QT را در کرای انجین بیان نکرده ام و مباحث هوش مصنوعی بسیار پیشرفته را پوشش نداده ام، جشن امشب می تواند زیباتر باشد وقتی که برای من پیام ارسال کنید و شریک شدن شادی تان با من را برای اتمام این کتاب فراموش نکنید.

ما را زدانی و نادانی در این قهقراست
چشم ما بر صبح امید که پرده برافکند
احمد کرمی بوکانی

<https://www.youtube.com/AhmadKarami>

<https://www.facebook.com/ahmad.karami.۹۶۷۴>

<http://ahmadkarami.blog.ir/>

به پایان آمدیم دفتر حکایت همچنان باقیست...

مدرسه ساخت بازیهای ویدئویی

ثبت نام کنید

آدرس : استان آذربایجان غربی - بوکان - جنب

کتابخانه پارک هلت - کوچه بهار ۲۰

مدرس: احمد کرمی

۰۹۳۵۷۹۴۲۱۷۰