
بسم الله الرحمن الرحيم

**دانشگاه صنعتی اصفهان**

مهندسی برق و کامپیوتر

# بررسی سیستم گراف انیمیشن در موتور بازی سازی آنریل و و پیاده‌سازی یک سیستم انیمیشن با استفاده از OpenGL

پایان‌نامه کارشناسی مهندسی کامپیوتر

**نامی نذیری**

استاد راهنما

**دکتر مازیار پالهنگ**

۱۴۰۱

**دانشگاه صنعتی اصفهان**

مهندسی برق و کامپیوتر

پایان‌نامه کارشناسی رشته مهندسی کامپیوتر آقای نامی نذیری

تحت عنوان

# بررسی سیستم گراف انیمیشن در موتور بازی سازی آنریل و و پیاده‌سازی یک سیستم انیمیشن با استفاده از OpenGL

در تاریخ ۱۳۹۵/۱۰/۲۰ توسط کمیته تخصصی زیر مورد بررسی و تصویب نهایی قرار گرفت:

| | |
|---|---|
| ۱- استاد راهنمای پایان‌نامه | دکتر مازیار پالهنگ |
| ۲- استاد داور | دکتر داور اول |
| ۳- استاد داور | دکتر داور دوم |
| سرپرست تحصیلات تکمیلی دانشکده | دکتر تحصیلات تکمیلی دانشکده |

# فهرست مطالب

**عنـــوان** **صفحه**



[1] Mesh
[2] Sub-Mesh
[3] Material
[4] Texture

## فهرست شکل‌ها

**عنـــوان** **صفحه**

# فهرست جدول‌ها

**عنـــوان** **صفحه**

# فهرست الگوریتم‌ها

**چکیده**

**کلمات کلیدی**

# فصل اول
# مقدمه

## ۱-۱ موتور بازی‌سازی

موتورهای بازی‌سازی پلتفرم‌هایی هستند که ساخت بازی‌های رایانه‌ای را آسان‌تر می‌کنند. آن ها به شما این امکان را می‌دهند تا عناصر بازی مانند انیمیشن، تعامل با کاربر یا تشخیص برخورد میان اشیاء را در یک واحد ادغام و ترکیب کنید. [۱] زمانی که از اصطلاح موتور بازی‌سازی استفاده می‌کنیم منظورمان نرم‌افزارهای قابل توسعه‌ای هستند که می توانند پایه و اساس بسیاری از بازی‌های مختلف باشند. [۲] موتورهای بازی‌سازی متشکل از اجزای مختلفی هستند که قابلیت‌های لازم برای ساخت بازی را فراهم می‌کنند. رایج ترین اجزای موتور بازی عبارتند از: [۱]

- مولفه‌ی صدا: نقش اصلی این مولفه تولید جلوه‌های صوتی در بازی است.
- موتور رندر: وظیفه اصلی این مولفه تبدیل داده‌های ورودی به پیکسل‌ها، برای به تصویر کشاندن بر روی صفحه است.
- مولفه هوش مصنوعی: این مولفه مسئولیت ارائه‌ی تکنیک‌هایی برای تعریف قوانین رفتار شخصیت‌هایی را دارد که توسط بازیکنان کنترل نمی‌شوند.
- مولفه انیمیشن: نقش اصلی این مولفه اجرای انیمیشن‌های مختلف مانند حرکت است.
- مولفه شبکه: وظیفه اصلی این مولفه قادرساختنِ بازیِ همزمانِ بازیکنان با یکدیگر، از طریق استفاده از دستگاه‌های

متصل به اینترنت است.

- مولفه منطق یا مکانیک بازی: این مولفه قوانین حاکم بر دنیای مجازی، ویژگی‌های شخصیت‌های بازیکنان، هوش مصنوعی و اشیاء موجود در دنیای مجازی و همچنین وظایف و اهداف بازیکنان را تعریف می‌کند.
- ابزارهای نرم‌افزاری: وظیفه اصلی این ابزارها افزایش راندمان و سرعت تولید بازی با موتور بازی‌سازی است. آن‌ها توانایی اضافه‌کردن بسیاری از عناصر مختلف را به بازی‌ها، از انیمیشن و جلوه‌های صوتی گرفته تا الگوریتم‌های هوش مصنوعی، را فراهم می‌کنند.

## ۲-۱ موتور بازی‌سازی آنریل

اولین نسل موتور بازی‌سازی آنریل توسط تیم سوینی، بنیانگذار اپیک گیمز [۱]، توسعه یافت. سوینی در سال ۱۹۹۵ شروع به نوشتن این موتور برای تولید بازی تیراندازی اول شخصی به اسم غیرواقعی [۲] کرد. نسخه‌ی دوم موتور بازی‌سازی آنریل در سال ۲۰۰۲ منتشر شد. نسخه سوم نیز در سال ۲۰۰۴ پس از اینکه حدود ۱۸ ماه در حال توسعه بود منتشر شد. در این نسخه، معماری پایه‌ای موجود در نسخه‌ی اول مانند طراحی شی‌گرا، اسکریپت‌نویسی مبتنی بر داده و رویکرد نسبتا ماژولار نسبت به زیرسیستم‌ها وجود داشت. اما برخلاف نسخه دوم که از یک خط لوله با عملکرد ثابت [۳] استفاده می‌کرد، این نسخه به صورتی طراحی شده بود تا بتوان قسمت‌های سایه‌زنی سخت‌افزاری [۴] را برنامه‌نویسی کرد.

موتور بازی‌سازی آنریل ۴ در سال ۲۰۱۴ در کنفرانس توسعه‌دهندگان بازی [۵] منتشر شد. این نسخه با طرح کسب‌وکار اشتراکی برای توسعه‌دهندگان در دسترس قرار گرفت. این اشتراک به صورت ماهانه، با پرداخت ۱۹ دلار آمریکا به توسعه‌دهندگان این اجازه را می‌داد تا به نسخه‌ی کامل موتور، از جمله کد منبع C++ آن دسترسی پیدا کنند. البته در سال ۲۰۱۵ اپیک گیمز موتور بازی‌سازی آنریل را به صورت رایگان برای همگان منتشر ساخت. آخرین نسخه آنریل به اسم موتور بازی‌سازی آنریل ۵ در سال ۲۰۲۰ معرفی شد. این نسخه از تمام سیستم‌های موجود از جمله کنسول‌های نسل بعدی پلی‌استیشن ۵ [۶] و ایکس‌باکس سری X/S [۷] پشتیبانی می کند. کار بر روی این موتور حدود دو سال قبل از معرفی آن شروع شده بود. در سال ۲۰۲۱ نسخه‌ای از آن به صورت دسترسی اولیه منتشر شد. به طور رسمی در سال ۲۰۲۲ نسخه‌ی کامل این موتور برای توسعه‌دهندگان انتشار یافت. [۳]

[1] Epic Games
[2] Unreal
[3] fixed-function pipeline
[4] shader hardware
[5] GDC
[6] PlayStation 5
[7] Xbox Series X/S

## ۱-۳ انیمیشن‌های کامپیوتری

# فصل دوم
# انیمیشن‌های اسکلتونی

انیمیشن اسکلتونی تکنیکی در انیمیشن‌های کامپیوتری است که به وسیله‌ی آن شخصیت‌های درون بازی متحرک می‌شوند. این سیستم به دو بخش کلی تقسیم می‌شود. یک بخش،یک مش یا پوسته است که برای به نمایش کشاندن شخصیت در محیط سه‌بعدی استفاده می‌شود و بخش دوم یک اسکلت است. این اسکلت مجموعه سلسله مراتبی از قطعات به هم پیوسته است که به هر قطعه یک مفصل گویند. در این فصل به بررسی این دوبخش و تکنیک‌های موجود در انیمیشن‌های اسکلتونی خواهیم پرداخت

## ۲-۱ مدل اسکلتونی

در انیمیشن‌های اسکلتونی از مدل‌های اسکلتونی استفاده می‌شود. هر مدل اسکلتونی از دو بخش مدل و اسکلت تشکیل‌شده‌است.

## ۲-۲ شبکه‌ی[1] چندضلعی

در گرافیک کامپیوتری سه‌بعدی و مدل‌سازی جامد، شبکه چندضلعی مجموعه‌ای از رئوس، لبه‌ها و وجوه است که شکل یک جسم چندوجهی را مشخص می‌کند. وجوه معمولاً از مثلث‌ها (شبکه مثلثی)، چهار ضلعی‌ها (چهار گوشه)، یا دیگر چند ضلعی‌های محدب ساده ( n ضلعی‌ها ) تشکیل شده‌اند. دلیل استفاده از این نوع چند ضلعی‌ها آسان‌تر بودن به نمایش کشیدن آن‌ها در محیط سه‌بعدی است. البته در حالت کلی اشیاء ممکن است از چندضلعی‌های مقعر و یا حتی چندضلعی‌های دارای سوراخ نیز تشکیل شده‌باشند.

اشیاء ایجادشده توسط مش‌های چند ضلعی باید انواع مختلفی از عناصر، از جمله رئوس، لبه‌ها، وجوه، چندضلعی‌ها و سطوح را در خود ذخیره کنند. در بسیاری از نرم‌افزارهای سه‌بعدی، فقط رئوس، لبه‌ها و یکی از دو مورد وجوه یا چندضلعی‌ها ذخیره می‌شوند. در اکثر سیستم‌های رندر [2] فقط از وجوه سه‌ضلعی (مثلث‌ها) استفاده می‌شود. بنابراین در این حالت چندضلعی‌های مدل، باید به شکل مثلث باشند. البته سیستم‌های رندر ای وجود دارند که از چهارضلعی‌ها یا چندضلعی‌های با تعداد اضلاع بالاتر نیز پشتیبانی می‌کنند و یا در لحظه این چندضلعی‌ها را به مجموعه‌ای از مثلث‌ها تبدیل می‌کنند که در این صورت باعث می‌شود نیازی به ذخیره‌ی مش به شکل مثلثی نباشد.

بنابراین چهار قسمت اصلی یک مش چندضلعی، رئوس، لبه‌ها، وجوه و چندضلعی‌ها هستند. توضیح کوتاهی درباره‌ی هر کدام از این موارد را در زیر می‌توانیم مشاهده کنیم.

### ۲-۲-۱ راس

راس‌ها معمولا یک موقعیت در فضای سه‌بعدی همراه با اطلاعات دیگر مانند رنگ، بردار نرمال و مختصات بافت را شامل می‌شوند. در راس‌های مربوط به مش‌های اسکلتونی اطلاعاتی مانند تعداد مفاصلی که بر روی این راس تاثیر می‌گذارند همراه با وزن تاثیرگذاری آن‌ها، می‌تواند اضافه شود.

### ۲-۲-۲ لبه

ارتباط بین دو راس را لبه گویند.

### ۲-۲-۳ وجه

مجموعه‌ای بسته از لبه‌ها را وجه گویند. وجه‌ها می‌توانند از سه لبه (وجه مثلثی) یا از چهار لبه (وجه چهارگوش) تشکیل شده باشند.

---

[1] Mesh
[2] renderer

### ۲-۲-٤ چندضلعی

یک چندضلعی مجموعه‌ای همسطح از وجوه است. در سیستم‌هایی که از وجه‌های چند ضلعی پشتیبانی می‌کنند، وجوه و چندضلعی‌ها یکسان هستند ولی در صورتی که سیستم مورد نظر تنها از سه یا چهار ضلعی‌ها پشتیبانی کند، در این صورت به چند ضلعی‌ها، مجموعه‌ای از وجوه گفته می‌شود.

## ۲-۳ مدل

مدل [۱] درواقع هر شیٔای است که در محیط سه‌بعدی قرار می‌گیرد و به تصویر کشیده می‌شود. هر مدل می‌تواند از چند زیرمش تشکیل شود. به عنوان مثال یک ماشین را درنظر بگیریم. موجودیت ماشین می‌تواند یک مدل باشد که در محیط سه‌بعدی قرار می‌گیرد. مدل ماشین می‌تواند از چند زیرمش مانند چرخ‌ها، لاستیک‌ها و بدنه‌ی ماشین تشکیل شود. دلیل وجود داشتن یک موجودیت کلی به اسم ماشین این است که یک شخصی مانند طراح محیط و یا طراح مرحله نمی‌خواهد هر بار که ماشینی را در محیط قرار دهد، تک تک زیرمش‌ها را به صورت دستی در صحنه وارد کند و در سر جای خودش قرار بدهد.

## ۲-٤ زیرمش [۲]

چندضلعی‌های دارای یک نوع ماده [۳] را یک زیرمش گویند. همانطور که اشاره شد، هر مدل از چند زیرمش تشکیل می‌شود. دلیل این تقسیم این است که در هر عملیات به تصویر کشیدن [۴] تنها یک ماده می‌تواند به تصویر کشیده شود. مثلا در مثال ماشین، قسمت‌های مختلف ماشین از ماده‌های مختلفی تشکیل می‌شود. به طور مثال چرخ ماشین می‌تواند از جنس آلومینیوم باشد، لاستیک چرخ از جنس پلاستیک باشد و یا حتی قسمت‌های داخلی ماشین مانند صندلی ماشین، از جنس چرم باشد. بنابراین باید این قسمت‌ها به صورت جدا قرار گیرند تا بتوان هر قسمت را با توجه به ماده‌ی موردنظر آن به تصویر کشاند.

## ۲-۵ ماده [۵]

ماده‌ها شامل پارامترهای قابل تنظیمی هستند که با تنظیم آن‌ها، به گرافیک اعلام می‌شود که چگونه باید یک مثلث را به تصویر بکشد. این پارامترها می‌توانند شامل موارد زیر باشند ولی محدود به آن نمی‌شوند

۱. میزان کدورت و شفافیت شئ

---

[۱] گاهی به جای استفاده از واژه‌ی مدل، از واژه‌ی مش هم استفاده می‌شود.

[2] Sub-Mesh
[3] Material
[4] Render
[5] Material

۲. میزان براقی شئ

۳. رنگ(بافت) شئ

۴. سایه‌زنی پیکسلی یا راسی [1]

### ۲-۶ بافت [2]

بافت یک تصویر دوبعدی و یا سه‌بعدی است که می‌تواند در ماده استفاده شود. این تصاویر به عنوان ورودی در برنامه دریافت شده و پس از اینکه یک شناسه به آن ها تخصیص داده شد، در کارت گرافیک قرار می‌گیرند. ماده‌ها با استفاده از این شناسه می‌توانند در صورت لزوم به این بافت‌ها دستیابی پیدا کنند.

### ۲-۷ اسکلت

به مجموعه ای از مفاصل که به صورت سلسله مراتبی به یکدیگر متصل می‌شوند، اسکلت گویند. پس از آنکه هنرمندان مدل شخصیت را طراحی می‌کنند در طی یک مرحله که به آن Rigging گویند، ساختار سلسله مراتبی اسکلت را به وجود می‌آورند. در انیمیشن‌ها درواقع این اسکلت است که حرکت می‌کند و با حرکتش باعث حرکت مدل شخصیت می‌شود.

### ۲-۸ Skinning

تا اینجا با دو مفهوم مدل و اسکلت آشنایی پیدا کردیم ولی نگفتیم که این دو چگونه به هم مرتبط می‌شوند. به عملیاتی که طی آن مفاصل موجود در اسکلت به مدل متصل می‌شود skinning گویند. طی این مرحله هر راس موجود در پوسته‌ی مش به یک یا چند مفصل متصل می‌شود. برای اینکه چگونه رئوس مش، این مفاصل را دنبال کنند الگوریتم‌های مختلفی مطرح شده‌است که در فصل پیاده‌سازی به یکی آن‌ها اشاره خواهد شد.

### ۲-۹ ژست شخصیت

ژست یک شخصیت نشان‌دهنده‌ی نحوه‌ی قرارگیری مفصل‌ها در اسکلت است. ژست‌های مختلف با دروان، حرکت یا تغییر اندازه‌ی مفاصل درون اسکلت به وجود می‌آیند. همانگونه که اشاره شد، اسکلت یک مدل در مرحله‌ی Rigging به وجود می‌آید و در همین مرحله با استفاده از Skinning به مدل متصل می‌شود. زمانی که این عمل صورت می‌گیرد

[1] Vertex or Pixel shader
[2] Texture

مدل در یک ژست به خصوص قرار دارد که به آن ژست حالت اتصال [۱] یا ژست مرجع [۲] گویند. به صورت کلی شخصیت در این حالت به صورتی ایستاده‌است که پاهایش کمی از هم باز است و بازوهایش به شکل حرف T کشیده است. به همین جهت گاهی به ژست حالت اتصال، ژست T [۳] هم گفته می‌شود. این حالت خاص به این دلیل انتخاب می‌شود که اندام‌ها را از بدن دور نگه دارد و اینکار باعث می‌شود که فرایند اتصال رئوس به مفصل آسان‌تر شود.

همانگونه که اشاره‌شد مفاصل به صورت سلسله مراتبی به یکدیگر متصل هستند. یعنی نحوه‌ی قرارگیری آن‌ها متناسب با نحوه‌ی قرارگیری والدشان است. این کار باعث می‌شود که مفاصل به صورت طبیعی حرکت کنند. یعنی در صورتی که والد حرکت کند، به واسطه‌ی آن فرزند نیز حرکت می‌کند. زمانی که ژست شخصیت در این حالت والد، فرزندی قرار دارد به آن ژست محلی [۴] گفته می‌شود. حالت دیگری نیز وجود دارد که موقعیت هر مفصل نسبت به فضای مختصاتی مدل درنظر گرفته می‌شود. به ژست شخصیت در این حالت ژست جهانی [۵] گفته می‌شود.

### ۲-۱۰ کلیپ‌های انیمیشنی

در یک فیلم انیمیشنی، تمام بخش‌های یک صحنه قبل از ساخت هر انیمیشن به دقت برنامه‌ریزی می‌شود. این شامل حرکات هر شخصیت، لوازم موجود در صحنه و حتی حرکات دوربین نیز می‌شود. این بدان معنی است که کل صحنه را می‌توان به عنوان یک دنباله طولانی و پیوسته از فریم‌ها، متحرک ساخت. در این حالت در صورتی که شخصیتی خارج از دوربین هستند لازم نیستند که متحرک شوند.

کلیپ‌های انیمیشنی متفاوت از این هستند. یک بازی، یک تجربه‌ی تعاملی است بنابراین نمی‌توان از قبل چگونه حرکت کردن شخصیت‌ها و رفتار آن‌ها را پیش‌بینی کرد. حتی تصمیمات شخصیت‌های غیربازیکن کامپیوتری نیز می‌توانند تابعی از اقدامات غیر قابل پیش‌بینی بازیکن انسانی باشد. به این ترتیب، کلیپ‌های انیمیشنی مربوط به بازی تقریبا هیچ‌گاه از مجموعه‌ای از فریم‌های طولانی و به هم پیوسته تشکیل نمی‌شوند. درعوض، حرکت شخصیت بازی باید به تعداد زیادی حرکات ریز تقسیم شود. منظور از کلیپ‌های انیمیشنی این حرکات کوتاه و یکتا است.

بنابراین هر کلیپ به صورتی طراحی شده است که یک عمل کاملا مشخص را انجام دهد. برخی از این کلیپ‌ها به گونه‌ای طراحی شده اند که بتوان آن را به صورت حلقه شونده تکرار کرد. به عنوان مثال چرخه‌ی راه‌رفتن یا دویدن می‌توانند از این نوع کلیپ‌ها باشند. و حرکاتی مانند پریدن یا دست تکان دادن از نوعی هستند که تنها یک‌بار پخش می‌شوند.

بنابراین به طور کلی حرکات هر شخصیت بازی معمولا به هزاران کلیپ تقسیم می‌شود. [۲]

---

[1] Bind Pose
[2] Reference Pose
[3] T Pose
[4] Local Pose
[5] Global Pose

### ۲-۱۱ ترکیب انیمیشن

اصطلاح ترکیب انیمیشن به هر تکنیکی اطلاق می‌شود که در آن بیش از یک کلیپ انیمیشن در ژست نهایی کاراکتر سهیم می‌شود. به صورت دقیق تر در این عمل دو یا چند ژست برای ایجاد یک ژست خروجی برای اسکلت شخصیت، با یکدیگر ترکیب می‌شوند. همانطور که در بخش قبل گفته شد، کلیپ‌های انیمیشنی، کلیپ‌های کوتاه و یکتایی هستند. با استفاده از روش ترکیب می‌توان مجموعه‌ای از کلیپ‌های انیمیشنی را با یکدیگر ترکیب کرد تا مجموعه‌ی جدیدی از انیمیشن‌ها را بدون نیاز به ایجاد دستی و از پایه‌ی آن ها تولید کنیم. به عنوان مثال، با ترکیب یک انیمیشن راه رفتن آسیب دیده با راه رفتن بدون آسیب دیدگی، می‌توانیم سطوح مختلفی از آسیب دیدگی در هنگام راه‌رفتن را به وجود آوریم. از ترکیب می‌توان برای درون‌یابی بین حالات مختلف چهره، حالت‌های مختلف بدن و حالت‌های مختلف حرکتی استفاده کرد. از ترکیب می‌توان برای یافتن یا حالت میانی بین دو حالت شناخته شده در زمان‌های مختلف استفاده کرد. این کار زمانی استفاده می‌شود که بخواهیم ژست یک شخصیت را در نقطه‌ای از زمان پیدا کنیم که دقیقا با یکی از فریم‌های نمونه موجود در داده‌های انیمیشن مطابقت ندارد. همچنین می‌توانیم از ترکیب موقتی انیمیشن برای انتقال هموار از یک انیمیشن به انیمیشن دیگر، با ترکیب تدریجی انیمیشن مبدا به مقصد در مدت زمان کوتاهی استفاده کنیم.

# فصل سوم
# سیستم انیمیشن گراف در موتور بازی‌سازی آنریل

در این فصل ابتدا توضیحاتی راجع به آنریل انجین داده می‌شود و سپس در رابطه‌ی سیستم انیمشن گراف این انجین صحبت خواهد شد.

## ۳-۱ بازیگران، پیاده‌ها و شخصیت‌ها

اشیا در آنریل می‌توانند به سه کلاس کلی بازیگران، پیاده‌ها و شخصیت‌ها دسته‌بندی می‌شوند. بازیگران کلاس پایه‌ی تمامی اشیا ای هستند که به صورت فیزیکی می‌توانند در محیط سه‌بعدی قرار گیرند. پیاده‌ها کلاسی مشتق شده از بازیگران هستند که بازیکنان می‌توانند کنترل آن‌ها را بدست گیرند و در محیط حرکت کنند. و در نهایت شخصیت‌ها پیاده‌هایی هستند که دارای مش اسکلتونی، توانایی شناسایی برخورد و منطق حرکتی هستند. آنها مسئول تمام تعاملات فیزیکی بین بازیکن یا هوش مصنوعی، با جهان هستند و همچنین مدل های اولیه شبکه و دریافت ورودی را پیاده سازی می کنند. اگر بخواهیم شخصیت درون بازی از انیمیشن‌های اسکلتونی استفاده کند، باید از این کلاس بهره ببریم.

### ۳-۲ اجزاء

اجزاء [1] مجموعه‌ای از توابع و ویژگی‌ها است که می‌تواند به یک بازیگر اضافه شود. بنابراین بازیگران می‌توانند حاوی مجموعه‌ای از ActorComponents باشند که این اجزاء می‌توانند برای موارد مختلفی از جمله کنترل نحوه‌ی حرکت بازیگران، نحوه‌ی رندر شدن و غیره استفاده شوند.

زمانی که یک مولفه به یک بازیگر اضافه می‌شود، آن بازیگر می‌تواند عملکردهای موجود در آن مولفه را استفاده کند. به عنوان مثل یک مولفه نور نقطه‌ای باعث می‌شود که بازیگر مانند یک نور نقطه‌ای، نور ساطع کند. یا یک مولفه صورتی به بازیگر این توانایی پخش صدا را می‌دهد.

مولفه‌ها حتما باید به یک بازیگر متصل شوند و به خودی خود نمی‌توانند وجود داشته باشند. درواقع وقتی ما مولفه‌های مختلف را به بازیگر خود متصل می‌کنیم درواقع در حال قرار دادن قطعه‌ها و تکه‌هایی هستیم که مجموع آن‌ها یک بازیگر را به عنوان یک موجودیت واحد که در محیط سه‌بعدی قرار می‌گیرد تعریف می‌کنند. به عنوان مثال چرخ‌های یک ماشین، فرمان ماشین، چراغ‌ها و غیره همه به عنوان مولفه‌های ماشین درنظر گرفته می‌شوند در حالی که خود آن ماشین، بازیگر است.

### ۳-۳ شخصیت‌ها

هر شخصیت در آنریل از سه مولفه‌ی اصلی تشکیل شده است.

- Skeletal Mesh Component
- Character Movement Component
- Capsule Component

همانطور که در فصل‌های گذشته اشاره‌شد شخصیت‌ها برای پخش انیمیشن‌ها نیاز به یک مش اسکلتونی دارند. مولفه‌ی Skeletal mesh Component مش اسکلتونی اصلی مرتبط با شخصیت است. این مولفه‌ای است که برای ما در این پروژه اهمیت زیادی دارد.

مولفه‌ی Character Movement Component همانطور که از اسمش مشخص است برای منطق حرکت در حالت‌های مختلف از جمله راه‌رفتن افتادن و غیره استفاده می‌شود. این مولفه شامل تنظیمات و عملکردهای مربوطه برای کنترل حرکت است.

و در نهایت مولفه‌ی Capsule Component وظیفه‌ی تشخیص برخورد در هنگام حرکت را دارد.

---

[1] Components

**فصل چهارم**

# معماری موتور بازی‌سازی آنریل

در این فصل ابتدا به معماری موتور آنریل پرداخته شده و پس از آن درباره‌ی ابزارهایی که این انجین در اختیار ما می‌گذارد صحبت می شود.

## ٤-١ UObject و Actors

کلاس پایه برای تمامی کلاس‌های دیگر در موتور آنریل UObject است. شئ‌ها [1] نمونه‌هایی از کلاس‌هایی هستند که از UObject ها ارث می‌برند. بازیگران [2] نمونه‌هایی از کلاس‌هایی هستند که از. AActor ارث برده اند. کلاس AActor کلاس پایه برای تمامی اشیائی است که می‌توانند در جهان بازی قرار گیرند. به صورت کلی، بازیگران را می‌توان به عنوان یک کل یا موجودیت در نظر گرفت و اشیاء را قطعات تخصصی‌ای درنظر گرفت که در این موجودیت به کار می‌روند که به آن ها جزء [3] می‌گویند. بنابراین اجزاء یک نوع خاصی از اشیاء هستند که بازیگران می‌توانند آن ها را به عنوان یک زیرشئ [4] به خود متصل کنند.

به عنوان مثال اگر یک ماشین را درنظر بگیریم. ماشین به عنوان یک موجودیت کلی به عنوان بازیگر درنظر گرفته

[1] Objects
[2] Actors
[3] Component
[4] sub-object

می‌شود. در صورتی که قسمت‌های مختلف این ماشین مانند درِ ماشین یا چرخ ماشین اجزای آن ماشین درنظر گرفته می‌شوند. در ادامه این مثال، اگر کاربر قرار باشد که این ماشین را کنترل کند، یک جزء دیگر می‌تواند مسئولیت تغییر سرعت و جهت ماشین بر اساس ورودی کاربر را داشته باشد. [۴، ۵]

## ۲-٤ Components

همانطور که گفته‌شد، اجزاء نوع خاصی از اشیاء هستند که بازیگران می‌توانند به عنوان اشیاء فرعی به خود متصل کنند. کلاس پایه برای تمامی اجزاء، کلاس UActorComponent است. از آنجایی که استفاده از اجزاء تنها راه ممکن برای پرداخت [۱] مش‌ها [۲]، تصاویر ،پخش صداو درواقع هرچیزی که بازیکن هنگام بازی در جهان مشاهده یا تعامل می‌کنند هستند، بنابراین در نهایت از انواعی از این نوع اجزاء در توسعه‌ی بازی استفاده می‌شود.

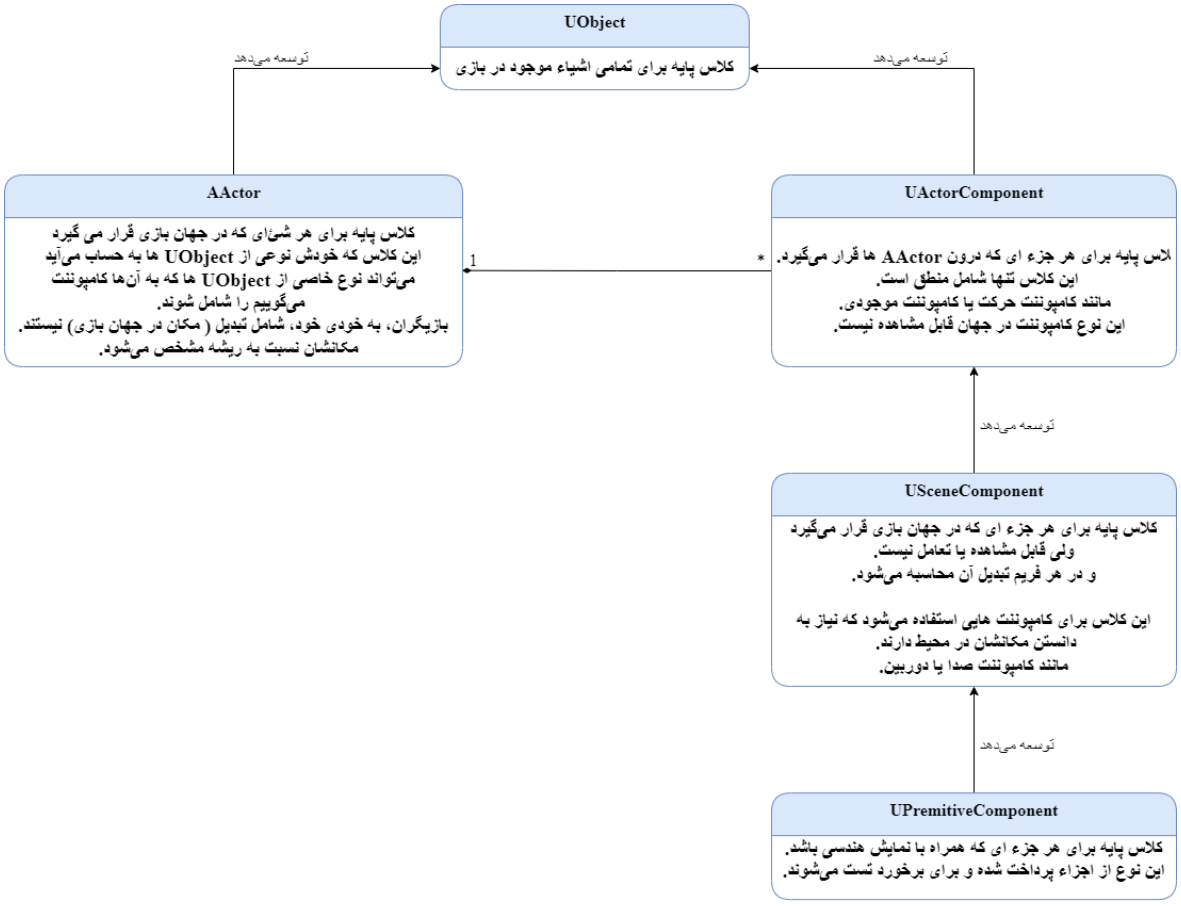
برای ساخت اجزاء، چند کلاس اصلی وجود دارد که در هنگام ایجاد اجزاء باید به آن توجه کرد.

- اجزای بازیگر[۳]: این کلاس بیشتر برای رفتارهای انتزاعی مانند حرکت، مدیریت موجودی یا ویژگی و سایر مفاهیم غیرفیزیکی مفید هستند. این نوع از اجزاء هیچ گونه مکان فیزیکی یا چرخشی در جهان ندارد.
- اجزای صحنه [۴]: این کلاس فرزند کلاس اجزای بازیگر است و از رفتارهای مبتنی بر مکان پشتیبانی می‌کند که به نمایش هندسی نیاز ندارند. این کلاس می‌تواند شامل بازوهای فنری، دوربین‌ها، نیروهای فیزیک و حتی صدا شود.
- اجزای اولیه [۵]: این کلاس فرزند کلاس اجزای صحنه است. درواقع این کلاس همان کلاس اجزای صحنه، همراه با نمایش هندسی است که عموما برای نمایش عناصر بصری و برخورد [۶] یا همپوشانی [۷] با اشیاء فیزیکی استفاده می‌شود. این کلاس می‌تواند شامل مش‌های استاتیک [۸] یا اسکلتی [۹]، اسپرایت‌ها یا بیلبوردها، سیستم‌های ذرات [۱۰] و همچنین حجم برخورد [۱۱] جعبه، کپسول و کره شود.

[۵]

---

[1] Render
[2] Meshes
[3] ActorComponent
[4] SceneComponent
[5] Primitive Components
[6] collide
[7] overlap
[8] static mesh
[9] skeletal mesh
[10] particle systems
[11] collision volumes




شکل ۴-۱ - تصویر UML کلاس‌های پایه‌ی موتور بازی‌سازی آنریل

# فصل پنجم
# پیاده سازی

در این بخش به روش‌ها و ابزارهای استفاده شده در پیاده‌سازی سیستم انیمیشن اشاره خواهد شد

## ۵-۱ ابزارها

### ۵-۱-۱ OpenGL

OpenGL یک واسط برنامه نویسی کاربردی [1] است که با فراهم کردن توابع مختلف به توسعه‌دهندگان امکان دستکاری گرافیک و تصاویر را می‌دهد. OpenGL یک کتابخانه‌ی رندرینگ است. یک "شئ" به خودی خود در OpenGL مفهومی ندارد و به صورت مجموعه‌ای از مثلث‌ها و حالات مختلف درنظر گرفته می‌شود. بنابراین وظیفه‌ی ما است که بدانیم چه شئ‌ای در کدام قسمت صفحه رندر شده است. این کتابخانه تنها وظیفه‌اش، کشیدن تصاویری که است که می‌خواهیم به تصویر کشیده‌شوند. در این صورت اگر می‌خواهیم تصویری را به‌روزرسانی کنیم و یا به عنوان مثال شئ‌ای را تحرک دهیم باید به OpenGL درخواست دهیم که صحنه را دوباره‌برای ما رندر کند. [۶]

به صورت کلی OpenGL را می‌توان یک ماشین حالت بزرگ درنظر گرفت. هر حالت شامل مجموعه‌ای از متغیرها است که نحوه‌ی عملکرد OpenGL را مشخص می‌کند. به مجموعه‌ی این حالت‌ها OpenGL context نیز می‌گویند. در واقع context را می‌توان یک شئ درنظر گرفت که کل OpenGL را دربر می‌گیرد. عموما تمامی تغییرات، روی

[1] API

context فعلی اعمال می‌شود و سپس رندر می‌شود. [۶][۷]

### GLFW ۵-۱-۲

از آنجایی که به‌وجودآوردن یک پنجره‌ی جدید و همچنین context وابسته به نوع سیستم‌عامل است بنابراین نیازمند کتابخانه‌ای هستیم که بتواند این موارد را برای ما مدیریت کند. GLFW یک کتابخانه‌ی منبع باز و چندپلتفرمی برای OpenGL است که یک API ساده و مستقل از پلتفرم برای تولید پنجره‌ها، زمینه‌ها [۱] و سطوح، خواندن ورودی و مدیریت رویدادها [۲] را ارائه می‌کند. این کتابخانه از سیستم‌عامل‌های ویندوز ، مک و لینوکس و سیستم‌های مشابه یونیکس پشتیبانی می‌کند. [۸]

### GLAD ۵-۱-۳

کتابخانه‌های گرافیکی مانند OpenGL وظیفه‌ی پیاده‌سازی توابع گرافیکی را ندارند بلکه می‌توان آن‌ها را مانند یک هدر در زبان برنامه‌نویسی C++ دانست که تعریف اولیه توابع را دارند. پیاده‌سازی این توابع در درایورهای GPU قرار دارند. دسترسی به این اشاره‌گرهای تابع به خودی خود سخت نیست ولی از آنجایی که این اشاره‌گر ها وابسته به پلتفرم هستند بنابراین کار طاقت فرسایی است. وظیفه‌ی کتابخانه‌ی GLAD فراهم سازی و کنترل این اشاره‌گرهای تابع است. [۹]

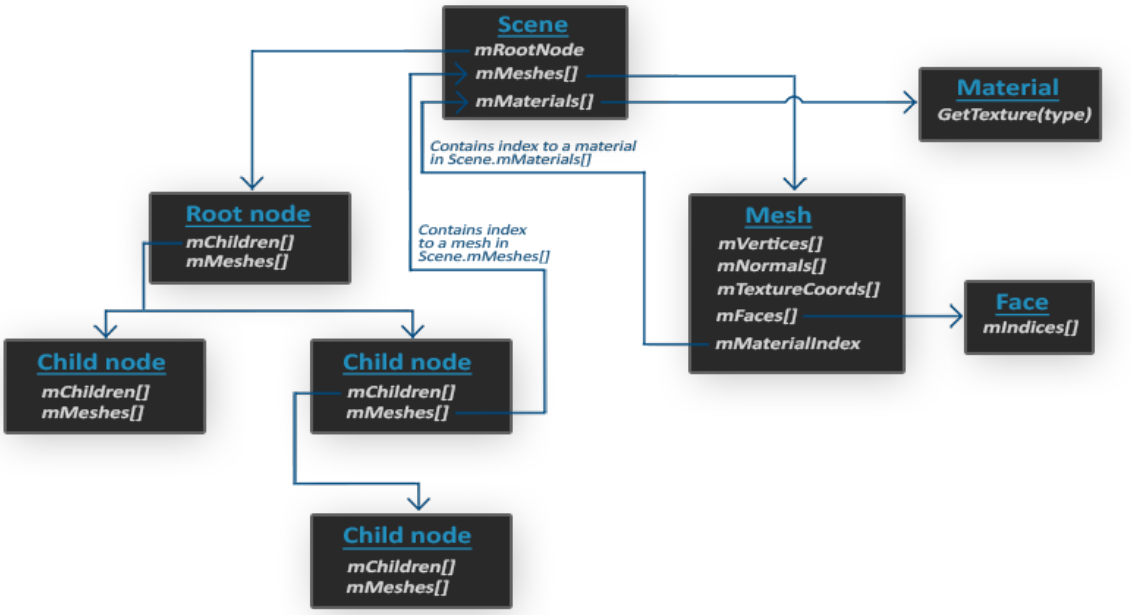
### GLM ۵-۱-٤

GLM یک کتابخانه‌ی ریاضی برای نرم‌افزارهای گرافیکی مبتنی بر زبان برنامه‌نویسی سایه‌ی OpenGL [۳] است. این کتابخانه تنها شامل یک هدر C++ است. توابع و کلاس‌های موجود در این کتابخانه به صورتی نامگذاری و طراحی شده‌آند که بسیار به GLSL نزدیک باشند.

### Assimp ۵-۱-۵

Assimp یک کتابخانه برای بارگذاری و پردازش صحنه‌های هندسی از فرمت‌های مختلف است. می‌توان با استفاده از آن مواردی همچون مش‌های استاتیک و یا اسکلتونی، مواد [۴] ، انیمیشن های اسکلتونی و داده‌های بافت را از فایل بارگذاری کرد. زمانی که این مدل‌ها بارگذاری می‌شوند این کتابخانه آن‌ها را در ساختاری به شکل زیر ذخیره می‌کند و بعد از آن می‌توان از این ساختار، داده‌های مورد نظر خود را خواند و از آن‌ها استفاده کرد. [۱۰][۱۱]

[1] Contexts
[2] Events
[3] OpenGL Shading Language(GLSL)
[4] Materials




شکل ۵-۱ - ساختار کلاس‌های کتابخانه‌ی Assimp [۱۱]

### ۵-۱-۶ stb

این کتابخانه برای بارگذاری تصاویر استفاده می‌شود. در این پروژه از این کتابخانه برای بارگذاری تصاویر بافت‌ها در کنار کتابخانه‌ی Assimp استفاده شده است. [۱۲]

## ۵-۲ پیاده‌سازی

این بخش دو هدف کلی را دنبال می‌کند.

۱. نمایش مدل گرافیکی و اجرای انیمیشن بر روی آن

۲. ترکیب انیمیشن‌های مختلف به وسیله‌ی ماشین حالت

### ۵-۲-۱ نمایش مدل گرافیکی

همانطور که گفته شد مدل‌ها یا اشیاء سه‌بعدی به خودی خود مفهومی در OpenGL ندارند. آنچه برای OpenGL اهمیت دارد لیستی از مثلث‌ها است تا آن‌ها را به تصویر بکشد. مدل‌های سه‌بعدی از رئوس، لبه و وجوه تشکیل می‌شوند و در فرمت‌های مختلفی مانند FBX ذخیره می‌شوند. در این پیاده‌سازی، از کتابخانه‌ی Assimp برای خواندن این داده‌ها استفاده شده است.

### ۵-۲-۲ قرارگیری مدل سه‌بعدی در کارت گرافیک

آنچه برای OpenGL اهمیت دارد این است که به آن مجموعه‌ای از مثلث‌ها داده شود تا برایمان ترسیم کند. برای اینکار به صورت عمومی از ۳ آرایه مختلف استفاده می‌شود که به نام‌های VBO ، VAO و EBO شناخته می‌شوند. VBOs [۱] یک آرایه یا بافری است که تمامی رئوس مدل سه‌بعدی ما را در خود جای می‌دهد. همانطور که در بخش ۲-۲-۱ اشاره شد، رئوس علاوه بر اینکه شامل اطلاعات موقعیت مکانی در محیط سه‌بعدی هستند، شامل اطلاعات دیگری نظیر رنگ، بردار نرمال، مختصات بافت و... نیز می‌توانند باشند. بنابراین باید به صورتی به کارت گرافیک اعلام کنیم که این داده‌ای که در آرایه‌ی VBOs قرار دارد را چگونه تفسیر کند. اینکار با استفاده از یک آرایه‌ی دیگر به نام VAO [۲] صورت می‌گیرد. در نهایت گفتیم که آنچه برای کارت گرافیک اهمیت دارد دریافت مثلث‌ها است. بنابراین باید به طریقی بگوییم کدارم رئوس با اتصال به یکدیگر مثلث تشکیل می‌دهند. اینکار نیز با استفاده از آرایه‌ی EBOs [۳] صورت می‌گیرد.

### ۵-۲-۳ اسکلت شخصیت

اسکلت یک شخصیت به صورت مجموعه‌ای از مفاصل که به صورت سلسله مراتبی به یکدگیر متصل‌اند، تعریف می‌شود. در این پیاده‌سازی کلاس Bone نشان‌دهنده‌ی هر مفصل است. هر Bone یک والد دارد و می‌تواند به هر تعدادی فرزند داشته باشد. با توجه به تعریف آورده شده از اسکلت، کلاس اسکلت که با Skeleton مشخص شده، شامل لیستی از این مفاصل به همراه اشاره‌گری به مفصل ریشه است.

### ۵-۲-٤ اتصال اسکلت و مدل سه‌بعدی

اصطلاحی که برای اتصال اسکلت و مدل سه‌بعدی استفاده می‌شود Skinning است. در این روش هر راس موجود در مدل، به یک یا چند مفصل متصل می‌شود. الگوریتم به کاررفته در این پیاده‌سازی،الگوریتم linear blend skinning نام دارد. در این الگوریتم زمانی که یک راس به یک مفصل می‌شود به آن یک وزن نسبت داده می‌شود. این وزن نشان‌دهنده‌ی میزان تاثیرگذاری این مفصل بر روی این راس است. به بیانی دیگر، این وزن نشان می‌دهد که اکر این مفصل به مکان جدید منتقل شود، این انتقال چقدر بر روی آن راس تاثیر می‌گذارد. بنابراین برای بدست آوردن انتقال نهایی راس، باید انتقال راس را نسبت به هرکدام از مفاصلی که به آن متصل است را بدست آوریم، سپس انتقال نهایی برابر مجموع وزن‌دار تمامی این انتقال‌ها خواهد بود.

---

[1] Vertex Buffer Objectss
[2] Vertex Array Objects
[3] Element Buffer Objects

### ۵-۲-۵ انیمیشن

هر بازی‌های کامپیوتری هر کلیپ انیمیشنی شامل یک حرکت منحصر به فرد شخصیت داخل بازی است. هر کلیپ شامل ژست‌های اسکلت در فاصله‌های زمانی مشخصی است. در واقع آنچه باعث حرکت شخصیت می‌شود حرکت اسکلت شخصیت است. زمانی که اسکلت شخصیت با استفاده از یک انیمیشن جابه‌جا می‌شود، مدل شخصیت نیز با استفاده از روش‌های skinning که در بالا توضیح داده‌شد همراه این اسکلت حرکت می‌کند. انیمیشن‌ها از طریق کلاسی به اسم Animation Clip مدل‌سازی شده اند. این کلاس شامل آرایه‌ای از ژست های شخصیت در مدت زمان‌های مشخصی است. همراه یک اشاره‌گری به اسکلت شخصیت. نکته‌ی قابل توجه این است که هر کلیپ انیمیشنی مربوط به یک نوع اسکلت می‌شود. به زبانی دیگر نمی‌توان انیمیشنی که براس اسکلت شخصیت انسانی طراحی شده است را بر روی یک حیوان، مانند فیل اجرا کرد.

### ۶-۲-۵ پخش‌کننده‌ی انیمیشن

این سیستم وظیفه‌اش پخش کردن انیمیشن بر روی اسکلت شخصیت است. این سیستم با گرفتن یک انیمیشن و یک اسکلت، این انیمیشن را بر روی آن اسکلت اجرا می‌کند. همانطور که گفتیم، انیمیشن ها ژست شخصیت را در فاصله‌های زمانی مشخصی در خود ذخیره می‌کنند. وظیفه‌ی این سیستم این است که با استفاده از یک زمان‌سنج که نشان‌دهنده‌ی زمان فعلی بازی است ژست مناسب شخصیت را از داخل انیمیشن بدست آورد. قابل ذکر است که ممکن است این ژست با توجه به زمان بازی و فاصله‌های زمانی داخل انیمیشن از درون‌یابی دو ژست پشت سر هم در آن کلیپ بدست آید.

### ۷-۲-۵ الگوریتم پخش‌کننده‌ی انیمیشن

هر شخصیت درون بازی، اگر از نوع شخصیت اسکلتونی باشد، دارای یک پخش کننده ی انیمیشن خواهد بود.

در تصویر زیر تابع به‌روزرسانی اسکلت به وسیله‌ی انیمیشن را می‌توان مشاهده کرد.

```
currentTime += deltaTime;

const double currentAnimationTime = (currentTime - startTimeForCurrentAnim);
AnimationPose currentPose = currentClip->GetPoseForCurrentFrame(currentAnimationTime *
    currentClip->GetFramePerSecond());

SetSkeletonPose(currentPose);
```

برای اینکه بتوان یک انیمیشن را پخش کرد نیاز است دو مورد زیر را بدانیم.

۱. زمان فعلی درون بازی(CurrentTime)

۲. زمان شروع پخش انیمیشن فعلی(StartTimeForCurrentAnimation)

در ابتدا زمان فعلی درون بازی را برای این پخش‌کننده به‌روزرسانی می‌کنیم. سپس برای بدست آوردن زمان فعلی انیمیشن می‌توان از فرمول زیر استفاده کرد

CurrentAnimationTime = CurrentTime - StartTimeForCurrentAnimation

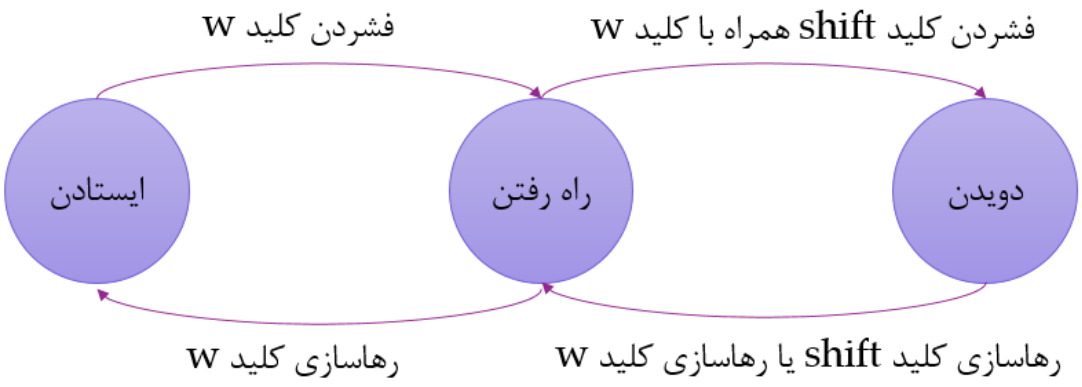
در نهایت با استفاده از این مقدار می‌توان ژست مورد نظر را از داخل کلیپ انیمیشنی بدست آورد. در نهایت نیز این ژست را بر روی اسکلت شخصیت اعمال می‌کنیم.

### ۵-۲-۸ ماشین حالت انیمیشن

یکی از روش‌های ترکیب انیمیشن‌های مختلف با یکدیگر، استفاده از ماشین حالت متناهی است. یک ماشین حالت متناهی شامل چندی حالت مختلف است که هر کدام از این حالات، حالتی از وضعیت سیستم را مشخص می‌کنند. زمانی که از ماشین حالت استفاده می شود سیستم می‌تواند در هر لحظه تنها در یکی از این حالات قرار گیرد. البته سیستم می‌تواند با دریافت ورودی از یک حالت به حالت دیگری رود.

دلیل استفاده از ماشین حالت متناهی برا سیستم انیمیشن این است که همانگونه که گفتیم، کلیپ‌های انیمیشنی، شامل ویدیوهای کوتاهی هستند که یک حالت مشخصی از شخصیت را بیان می‌کنند. در یک بازی، با توجه به ورودی بازیکن، شخصیت درون بازی می‌تواند در حالت‌های متفاوتی قرار گیرد. با استفاده از ماشین حالت می‌توان به تمامی این حالت‌ها رسیدگی کرد.

به عنوان مثال، تصویر زیر نشان‌دهنده‌ی یک ماشین‌حالت برای حرکت شخصیت است. شخصیت در ابتدا در حالت ایستاده قرار دارد و با گرفتن ورودی‌های مختلف از کیبورد، می‌تواند به حالت‌های دیگری رود.




شکل ۵-۲ - ماشین حالت برای حرکت شخصیت

برای پیاده‌سازی ماشین حالت متناهی، این سیستم به سه کلاس کلی شکسته‌شده‌است. کلاس AnimationStateMachine که وظیفه‌ی مدیریت حالت‌ها و انتقال از یک حالت به حالت دیگری را دارد. کلاس AnimationState که نشان‌دهنده‌ی

حالت شخصیت است. هر AnimationState شامل یک کلیپ است و هر زمانی که این حالت فعال می‌شود این کلیپ پخش می‌شود. در نهایت کلاس Transition که شامل توابع انتقال است. هر حالت می‌تواند شامل چندین انتقال باشد. و وظیفه‌ی AnimationStateMachine است که بررسی کند، اگر انتقالی امکان‌پذیر بود، آن را انجام دهد.

### ۵-۲-۹ به‌روزرسانی ماشین حالت انیمیشن

وضعیت توابع انتقال تاثیرگذاری مستقیمی در وضعیت سیستم به‌روزرسانی ماشین حالت انیمیشن دارد. وضعیت انتقال می‌تواند سه حالت زیر را داشته باشد.

۱. حالت عادی [۱]

۲. حالت در حال انتقال [۲]

۳. حالت اتمام انتقال [۳]

حالت اول حالت عادی است که نشان‌دهنده‌ی وضعیت عادی ماشین حالت است. در این وضعیت، توابع انتقال حالت فعلی بررسی می‌شوند تا در صورتی که شرایطشان برقرار شود، تغییر حالت رخ دهد. علاوه بر آن انیمیشن حالت فعلی با استفاده از کلاس پخش‌کننده آپدیت می‌شود. در صورتی که توابع انتقال مقدار درست [۴] را بازگردانند، ماشین به وضعیت دوم که وضعیت درحال انتقال است، تغییر وضعیت می‌دهد. در این وضعیت با توجه به زمانی که مشخص شده، ژست شخصیت با استفاده از درون‌یابی خطی از حالت فعلی به حالت جدید تغییر می‌کند. پس از اینکه انتقال به صورت کامل انجام شد، وضعیت ماشین حالت به اتمام انتقال تغییر می‌یابد. زمانی که ماشین در این وضعیت قرار گرفته یعنی به حالت جدید منتقل شده، بنابراین لازم است انیمیشن را از حالت جدید گرفته و آن را به کلاس پخش کننده داده تا آن را پخش کند. پس از این کار وضعیت ماشین دوباره به حالت عادی تغییر می‌یابد و همه‌ی این موارد دوباره تکرار می‌شوند.

```
if(transitionStatus == TransitionStatus::normal)
{
  for (const auto& transition : currentState->GetTransitions()) // loop through
      transitions of the current state
  {
    if (transition->Evaluate())
    {
      transitionStatus = TransitionStatus::transitioning;
      currentState = animationStatesMap.at(transition->to);
      TransitionFromPose = animator->GetPoseAtCurrentTime();
      TransitionToPose = currentState->GetAnimClip()->GetPoseForCurrentFrame(0);
      currentTime = 0;
      transitionTime = transition->transitionTime;
```

[1] Normal
[2] Transitioning
[3] Finished
[4] True

```
            break;
        }
    }
}

if (transitionStatus == TransitionStatus::normal)
{
    animator->Update(deltaTime);
}
else if(transitionStatus == TransitionStatus::transitioning)
{
    if(TransitionUpdate(deltaTime))
    {
        transitionStatus = TransitionStatus::finished;
    }
}
else if(transitionStatus == TransitionStatus::finished)
{
    animator->ChangeAnimationClip(*(currentState->GetAnimClip()), 0);
    transitionStatus = TransitionStatus::normal;
}
```

# پیوست‌ها

# مراجع


[1] Barczak, A. M. and Woźniak, H., “Comparative study on game engines”, *Studia Informatica. Systems and Information Technology. Systemy i Technologie Informacyjne*, No. 1-2, 2019.

[2] Gregory, J., *Game Engine Architecture*, A K Peters/CRC Press, 3rd ed. , 2018.

[3] “Unreal engine wikipedia”, https://en.wikipedia.org/wiki/Unreal_Engine.

[4] “Unreal engine architecture”, https://docs.unrealengine.com/4.27/en-US/ProgrammingAndScripting/ProgrammingWithCPP/UnrealArchitecture/.

[5] “Unreal engine components”, https://docs.unrealengine.com/4.27/en-US/ProgrammingAndScripting/ProgrammingWithCPP/UnrealArchitecture/Actors/Components/.

[6] “Using opengl”, https://www.khronos.org/opengl/wiki/Getting_Started.

[7] “Opengl state macine”, https://learnopengl.com/Getting-started/OpenGL.

[8] “Glfw”, https://github.com/glfw/glfw.

[9] “Glad”, https://github.com/Dav1dde/glad.

[10] “Assimp”, https://assimp-docs.readthedocs.io/en/v5.1.0/about/introduction.html.

[11] “Assimp class hierarchy”, https://learnopengl.com/Model-Loading/Assimp.

[12] “stb”, https://github.com/nothings/stb.

# Analysis of the animation graph in Unreal Engine and implementation of an animation system using OpenGL

Nami Naziri
nami.naziri@yahoo.com

May 22, 2022
Department of Electrical and Computer Engineering
Isfahan University of Technology, Isfahan 84156-83111, Iran
Degree: Bachelor of Science Language: Farsi

**Supervisor: Maziar Palhang, Assoc. Prof., palhang@cc.iut.ac.ir.**

**Abstract**

**Keywords**

**Isfahan University of Technology**

Department of Electrical and Computer Engineering

# Analysis of the animation graph in Unreal Engine and implementation of an animation system using OpenGL

A Thesis

Submitted in partial fulfillment of the requirements

for the degree of Bachelor of Science

**By**

**Nami Naziri**

Evaluated and Approved by the Thesis Committee, on May 22, 2022

1- Maziar Palhang, Assoc. Prof. (Supervisor)

2- First Examiner, Assoc. Prof. (Examiner)

3- First Examiner, Assist. Prof. (Examiner)

Department Graduate Coordinator: Reza Tikani, Assist. Prof.