برنامه‌نویسی پیشرفته

زمستان 1403 - بهار 1404

فاز دوم پروژه

تیم تعریف و طراحی پروژه:

رایا نمازی

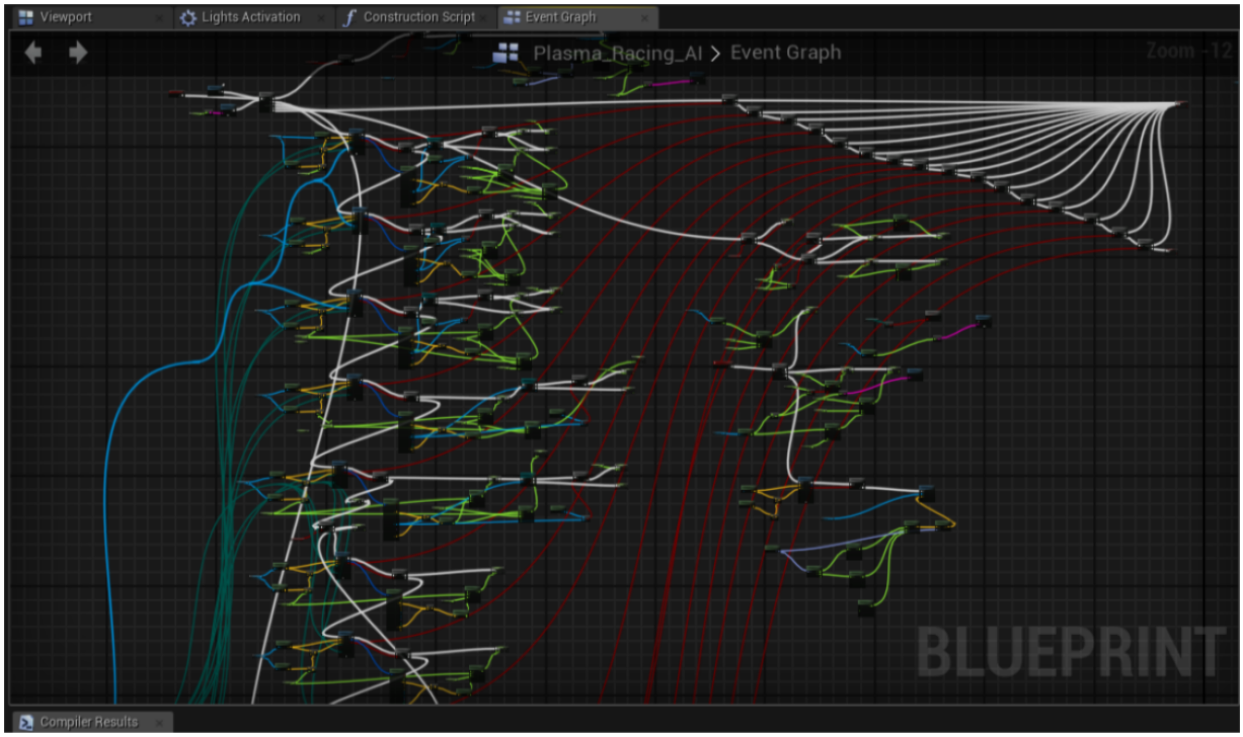
آرین همتی

آیدی‌ها روی اسامی لینک شده‌اند و کافیست روی آنها کلیک کنید

مقدمه‌ای بر فاز دوم (حتماً بخوانید!)

به فاز دوم پروژه درس برنامه‌نویسی پیشرفته خوش آمدید. در فاز اول پروژه شما یک مدل مینیمال از بازی‌ای که قرار است در طول این ترم پیاده‌سازی کنید را کامل کردید. در این فاز شما با پیاده‌سازی شالودهٔ فیزیکی بازی از جمله حرکت شتاب دار (برای انتیتی‌ها و همچنین پنجره‌ها) و برخورد، و یک مدل کلی از قابلیت‌ها و Skill Tree، مقدمات پیاده‌سازی اجزای اصلی بازی در فاز جاری را به پایان رساندید. هدف اصلی این فاز، تکمیل پیاده‌سازی لاجیک بازی و مدل‌های مورد نیاز برای پیاده‌سازی بازی است و تمرکز کمتری بر گرافیک بازی وجود خواهد داشت. اهداف آموزشی اساسی این فاز شامل کلین‌کدینگ، استفاده صحیح از معماری و دیزاین‌پترن‌ها و همچنین پیروی از اصول بنیادی SOLID است. مشابه فاز قبل، اکیداً توصیه می‌شود همچنان رعایت مدل MVC را در سرتاسر کد خود رعایت کنید و اگر متوجه اشکالی در رابطه با این اصل در کدهای فاز قبلی شدید، آن را حتماً قبل از ادامه پیاده‌سازی برطرف کنید. پیاده‌سازی اصولی این دو فاز، در راستای SOLID و MVC کار شما در فاز سوم پروژه را بسیار ساده‌تر خواهد کرد. برای اینکه شهود درستی از فلسفه رعایت مدل MVC داشته باشید توصیه می‌شود این کارگاه را ببینید. همچنین توصیه می‌شود قبل از شروع این فاز، مقداری درباره معماری‌های Event-Driven مطالعه داشته باشید. به این منظور مطالعه این منبع و همچنین مطالعه درس Concurrency In Swing توصیه می‌شود. در این فاز لایه‌های بسیاری از لاجیک و گرافیک کد شما روی هم اضافه می‌شوند و لذا چالش اصلی شما در این فاز، پیاده‌سازی اصولی این لایه‌ها خواهد بود. شما با رعایت این اصول و انتخاب معماری‌های مناسب، می‌توانید در عین راحتی کد زدن و جلوگیری از وقوع مشکلات مکرر و زمان‌بر در میانه پروژه، توسعه‌پذیری و تغییرپذیری آن را نیز حفظ کنید و در فاز سوم پروژه نیز زمان کمتری برای رفع مشکلات بالقوه در معماری لایه‌های کد خود صرف کنید. این در حالیست که رعایت نکردن این اصول می‌تواند منجر به ایجاد یک کد پیچیده و غیرقابل فهم و غیرقابل توسعه شود که مشکلات فراوانی در توسعه فاز سوم پروژه (شبکه، مولتی‌پلیر و ...) ایجاد خواهد کرد. فلذا اکیداً توصیه می‌شود قبل از دست به کد شدن، چندین بار (داک را به طور کامل بخوانید و زمان کافی صرف کنید تا معماری‌ای قابل اطمینان و نسبتاً دقیق از لایه‌های کد خود طراحی کنید و بتوانید به راحتی هر قسمت از کد را تکمیل کنید. منابع تکمیلی برای مطالعه بیشتر درباره اصول SOLID و همچنین دیگر مباحث مورد بحث در کلین‌کدینگ در مقدمهٔ داک فاز اول آمده است. شما برای پیاده‌سازی این فاز به دانش کافی درباره طراحی Game Loop، مولتی‌تردینگ، ورک‌فلوی EDT, AWT و به خصوص Design Pattern ها و معماری نیاز خواهید داشت. در این فاز از شما انتظار می‌رود با رعایت اصول کلین‌کدینگ و معماری‌های مناسب، کدی توسعه‌پذیر پیاده‌سازی کنید و از coupling در ساختار کد خود دوری کنید.

نکته بسیار مهم: برخی موارد تعریف شده در داک فاز قبل (مانند مکانیک‌های حرکت، برخورد و Impact) برای پیاده‌سازی اجزای مختلف این فاز ضروری هستند و در نمره این فاز به طور مستقیم تاثیر دارند. در صورت عدم پیاده‌سازی آنها در فاز اول، طبیعتاً نمره بخش‌هایی که برای فعالیت به این موارد وابسته‌اند را دریافت نخواهید کرد. موارد مربوط به کلین‌کدینگ و رعایت درست معماری‌ها و دیزاین پترن‌ها در حدود 20 درصد نمره این فاز را تشکیل می‌دهد. رعایت بخش‌هایی اعم از این بخش و حرکت شتاب‌دار و ابزارهای مورد استفاده که رعایت آنها در تمام فاز اول حائز نمره است، در این فاز و فاز بعدی نیز حائز نمرهٔ مجزا خواهند بود. بخش‌های دیگر فاز اول پروژه در بارم‌بندی این فاز لحاظ نخواهند شد.



تقریباً تمام متن‌های رنگی در سراسر این داک یا جملات و عبارات مهم در فرآیند پیاده‌سازی هستند و یا با کلیک روی آنها می‌توانید به بخش‌هایی از داک یا لینک منابع مفید منتقل شوید فلذا حتما به این قسمت‌ها توجه ویژه داشته باشید.

محیط بازی (Game Env.)

فاز جدید، HUD جدید!

در این فاز از شما انتظار می‌رود علاوه بر Wire Length, Packet Loss, Coins قابلیت‌های فعال شبکه (که در ادامه معرفی خواهند شد) را نیز در صفحه بازی به نمایش بگذارید. مشابه فاز قبل، HUD شما می‌تواند مخفی شود و یا همواره در معرض دید باشد.

طراحی شبکه‌های پیچیده: کلین‌کدینگ در دنیای واقعی!

در فاز قبل شما موفق شدید به عنوان یک اپراتور شبکه، سیستم‌های موجود در شبکه را به نحوی به یکدیگر متصل کنید که پکت‌ها را به سلامت به مقصد برسانید. در این فاز اما، با استقبال کاربران از سرویس شما، تعداد سیستم‌ها افزایش پیدا کرده است. از آنجا که تغییر دادن زیرساخت‌های موجود هزینه‌بر است، باید به عنوان یک اپراتور کاربلد (!) تلاش کنید شبکه را طوری طراحی کنید که تا حد ممکن توسعه‌پذیر باشد تا در صورت اضافه شدن سیستم‌های جدید به شبکه، هزینه ایجاد اتصالات برای ایجاد یک شبکه پایدار معقول باقی بماند. در ادامه توضیحات این فاز، ابزارهای جدید کنترل حرکت پکت‌ها روی شبکه به تفصیل معرفی خواهد شد.

Packet Mechanics v2.0

در این فاز، طراحی شبکه و سیم‌کشی‌ها و قابلیت‌های اپراتور در کنترل آنها دست‌خوش تغییراتی خواهند شد که در این فاز به آنها خواهیم پرداخت. همانطور که در فاز قبل اشاره شد، در این فاز سیم‌کشی‌ها لزوماً به صورت خطوط صاف نخواهند بود (توضیحات در ادامه آمده است). همچنین، واحد سکه (Coin) که به عنوان یک مورد از HUD از آن نام برده شده بود به بازی اضافه می‌شود:

سیم‌کشی‌های واقع گرایانه!

در این فاز از پروژه، اتصالات ایجاد شده در شبکه نمی‌توانند از روی سیستم‌ها عبور کنند. تا وقتی که سیم ایجاد شده، از روی یک سیستم عبور کرده باشد، این اتصال معتبر نخواهد بود و اجازه اجرای شبکه به شما داده نخواهد شد. در عوض، شما می‌توانید با پرداخت یک سکه، یک "انحنا" روی یک سیم ایجاد کنید! (پیشنهاد می‌شود انحناهای یک سیم را به صورت نقاطی رنگی روی سیم نمایش دهید) با این کار، شما می‌توانید این نقطه از سیم را تا شعاعی مشخص جابجا کنید تا آن را از موانع احتمالی عبور دهید. توجه کنید تغییرات شکل سیم باید به شکل پیوسته انجام شود. ایجاد انحنا روی یک سیم، طول سیم و مسیر حرکت پکت‌ها از روی این سیم را به کلی تغییر می‌دهد و در نتیجه یکی از چالش‌های احتمالی شما، طراحی راهکاری برای محاسبه طول سیم و تعیین مسیر حرکت پکت‌ها (با در نظر داشتن اثر شتاب و یا سرعت ثابت) روی آنهاست. همچنین روی هر سیم حداکثر 3 انحنا می‌توانید ایجاد کنید. ضمناً پس از ایجاد یک نقطه انحنا روی سیم، برای جابجا کردن نقطه انحنا نیاز به پرداخت سکه مجدد نیست.

راهنمایی: برای محاسبات مربوط به طول و همچنین نحوه حرکت صحیح (با در نظر گرفتن حرکت سرعت ثابت و یا حرکت شتاب‌دار) روی سیم‌های منحنی، هر یک از اتصالات شبکه را به صورت یک خط شکسته شامل تعداد مشخصی نقطه در نظر بگیرید

مکانیک‌های جدید مربوط به پکت‌ها (Packet Mechanics)

در این فاز به علت تغییر مکانیزم‌های سیم‌کشی، احتمال وجود سیم‌های طولانی افزایش می‌یابد. اگر طول سیم از حد مشخصی بیشتر باشد، یک پکت (وابسته به پروتکل‌های حرکتی) می‌تواند به صورت شتاب‌دار روی آن حرکت کند. حرکت شتاب‌دار روی یک سیم طولانی باعث افزایش شدید سرعت پکت می‌شود. از آنجا که سرعت انتقال داده‌ها در اتصالات شبکه محدود است، این اتفاق باعث آسیب به سیستم می‌شود! بنابراین در صورتی که سرعت یک پکت حین ورود به یک سیستم از حد مشخصی تجاوز کند،

سیستم مقصد به مدت مشخصی غیرفعال خواهد شد. در نتیجه مطلوب است که سرعت پکت‌ها را در حد محدودی حفظ کنید. به این منظور، در این فاز در فروشگاه آیتم‌های جدیدی قابل خرید خواهند بود که به شما برای کنترل سرعت حرکت پکت‌ها کمک می‌کنند. همان‌طور که در بخش بعد خواهید دید، در این فاز ممکن است سیستم‌ها به صورت ناگهانی از کار بیفتند. در این صورت، پکت بعد از رسیدن به سیستم مقصد، از طریق همان سیم به سیستم مبدا خود باز می‌گردد. همچنین در هنگام انتخاب یک پورت خروجی برای حرکت به سمت یک سیستم جدید، **پکت‌ها نباید پورت‌های منتهی به سیستم‌های غیر فعال را انتخاب کنند!**

انواع سیستم‌ها و پکت‌های موجود در شبکه

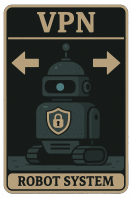
در این فاز علاوه بر سیستم‌های مرجع و سیستم‌های عادی، چهار دسته سیستم دیگر نیز به شبکه اضافه خواهند شد:

- ### سیستم‌های جاسوسی

با ورود یک پکت به یک سیستم جاسوسی، این پکت می‌تواند از هر یک از سیستم‌های جاسوسی موجود در شبکه خارج شود. با ورود پکت‌های محرمانه به این سیستم‌ها، پکت به طور کامل از بین خواهد رفت. این سیستم‌ها نمی‌توانند روی عملکرد پکت‌های محافظت شده تاثیر بگذارند.

- ### سیستم‌های خراب‌کار

این سیستم‌ها برعکس سیستم‌های عادی، پکت‌ها را به پورت‌های ناسازگار ارسال می‌کنند و در صورتی که پکت ورودی به آنها هیچ نویزی نداشته باشد، یک واحد نویز به آن القا می‌کنند. به احتمال مشخصی، هر پکت ورودی به این سیستم به یک پکت تروجان تبدیل خواهد شد. این سیستم روی عملکرد پکت‌های محافظت شده اثری ندارد.

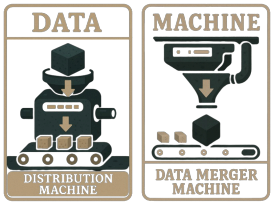


- ### سیستم VPN

هر پکت ورودی به یک سیستم VPN به شکل یک پکت محافظت شده از آن خارج خواهد شد. در صورتی که یک سیستم VPN از کار بیفتد، تمام پکت‌هایی که توسط این سیستم به پکت محافظت شده تبدیل شده‌اند به نوع اولیه خود باز می‌گردند و مجدداً نسبت به سیستم‌های خراب‌کار و سیستم‌های جاسوسی آسیب‌پذیر خواهند شد.

- ### سیستم آنتی‌تروجان

این سیستم‌ها سلامت پکت‌های شبکه را تا شعاع مشخصی بررسی می‌کنند. در صورتی که یک پکت تروجان در شعاع مشخصی از یک سیستم آنتی‌تروجان قرار داشته باشد، این سیستم آن را به یک پکت پیام‌رسان تبدیل می‌کند. بعد از هر عملکرد موفق، سیستم آنتی‌تروجان به مدت مشخصی از کار میفتد.



- ### سیستم‌های Distribute, Merge

این سیستم‌ها وظیفه تقسیم و بازیابی پکت‌های حجیم (که در ادامه به تفصیل توضیح داده شده‌اند) را دارند. این سیستم‌ها در عبور یک پکت عادی (غیر حجیم) مثل سیستم‌های عادی شبکه رفتار می‌کنند اما در صورتی که یک پکت حجیم وارد سیستم Distribution شود، به تعدادی بیت‌پکت تقسیم خواهد شد که بعدا (طبق توضیحاتی که در ادامه آمده است) توسط سیستم‌های Merger به پکت حجیم اولیه بازیابی می‌شوند.

در این فاز، همچنین پکت‌های شبکه نیز به چند دسته تقسیم می‌شوند و هر دسته، مکانیزم‌های حرکتی مخصوص به خود را خواهند داشت. همچنین همانطور که در فاز پیشین مشاهده کردید، نحوه حرکت هر پکت تنها به پورتی که از آن خارج می‌شد بستگی داشت اما همان‌طور که خواهید دید، در این فاز پورت مقصد یک پکت نیز در نحوه حرکت آن روی اتصالات موثر واقع خواهد شد. همچنین عملکرد پکت‌ها وابسته به نوع سیستمی که به آن وارد می‌شوند نیز متفاوت خواهد بود. همچنین در این فاز در صورتی که حرکت یک پکت روی یک سیم بیش از مدت زمان مشخصی به طول بینجامد، این پکت از بین خواهد رفت!

پکت‌های پیام‌رسان

این پکت‌ها که عادی‌ترین نوع پکت‌های موجود هستند، تنها وظیفه انتقال عادی پیام‌ها را در شبکه خواهند داشت و عبارتند از:

	اندازه: 2 واحد	با هر بار ورود به سیستم 2 سکه به شبکه اضافه می‌کند	سرعت حرکت با شروع از یک پورت سازگار نصف سرعت حرکت آن با شروع حرکت از یک پورت ناسازگار است. حرکت آن در هر یک از پورت‌ها با سرعت ثابت انجام می‌گیرد. (مگر در اثر **Impact**)
	اندازه: 3 واحد	با هر بار ورود به سیستم 3 سکه به شبکه اضافه می‌کند	حرکت آن با شروع حرکت از پورت‌های سازگار با سرعت ثابت است (مگر در اثر **Impact**) اما حرکت آن در عبور از یک پورت ناسازگار شتاب‌دار خواهد بود
	اندازه: 1 واحد	با هر بار ورود به سیستم 1 سکه به شبکه اضافه می‌کند	حرکت آن با شروع از یک پورت سازگار، با شتاب ثابت است (مگر در اثر **Impact**) اما در صورت حرکت آن در عبور از یک پورت ناسازگار، شتاب حرکت نزولی خواهد بود. در صورتی که این پکت بعد از شروع به حرکت روی یکی از سیم‌ها، با پکت دیگری برخورد کند، جهت حرکت خود را تغییر می‌دهد تا به سیستم مبدا بازگردد و مجدداً تلاش می‌کند خود را به سیستم مقصد برساند

اگر یک پیام متنی از یک پورت ناسازگار وارد یک سیستم از شبکه شود، با سرعتی دو برابر حالت عادی از سیستم خارج خواهد شد

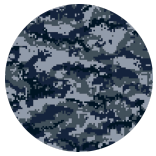
پکت‌های محافظت شده

این پکت‌ها در واقع از جنس یکی از پکت‌های پیام‌رسان هستند اما نوع آنها برای شبکه مشخص نیست و از لحاظ ظاهری هم پنهان خواهند بود. نحوه حرکت یک پکت محرمانه روی هر سیم به طور تصادفی از نحوه حرکت یکی از پکت‌های پیام‌رسان انتخاب می‌شود.

	اندازه: مقدار دو برابر پکت اولیه	با هر بار ورود به سیستم 5 سکه به شبکه اضافه می‌کند	این پکت در طی عبور یک پکت پیام‌رسان از یک سیستم VPN به وجود می‌آید. نحوه حرکت این پکت به طور تصادفی از نحوه حرکت یکی از پکت‌های پیام‌رسان انتخاب می‌شود. این پکت در طی عبور از یک سیستم خراب‌کار و یا جاسوسی، به نوع اولیه خود باز می‌گردد.

پکت‌های محرمانه

این پکت‌ها برای ارتباط امن بین دو سیستم در شبکه آزاد می‌شوند و کارکرد اصلی آنها در فاز بعد معرفی خواهد شد. در این فاز، در صورتی که چنین پکتی در مسیر حرکت به سمت یک سیستم از شبکه قرار داشته باشد، در صورتی که یک پکت دیگر در این سیستم ذخیره شده باشد، سرعت خود را حد مشخصی کاهش می‌دهد تا همزمان با پکت دیگری در این سیستم حضور نداشته باشد.

	اندازه: 4 واحد	با هر بار ورود به سیستم 3 سکه به شبکه اضافه می‌کند	این پکت با سرعت ثابت روی اتصالات حرکت می‌کند (مگر در اثر **Impact**) اما در صورتی که چنین پکتی در مسیر حرکت به سمت یک سیستم از شبکه قرار داشته باشد و پکت دیگری در این سیستم ذخیره شده باشد، سرعت خود را حد مشخصی کاهش می‌دهد تا همزمان با پکت دیگری در این سیستم حضور نداشته باشد
	اندازه: 6 واحد	با هر بار ورود به سیستم 4 سکه به شبکه اضافه می‌کند	این پکت در طی عبور یک پکت محرمانه عادی (مورد قبلی) از یک سیستم VPN به وجود می‌آید. این پکت در هر لحظه تلاش خواهد کرد فاصله مشخصی را (با حرکت به سمت جلو یا عقب روی اتصالات شبکه) با تمام پکت‌های دیگر موجود روی سیم‌های شبکه حفظ کند

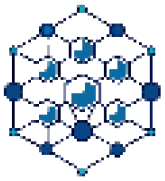
توجه کنید سازگاری برای پکت‌های محرمانه معنادار نیست و طبعاً پورت‌هایی متناظر با این پکت‌ها وجود نخواهد داشت

پکت‌های حجیم

این نوع از پکت ها حمل‌کننده داده‌های حجیم هستند و در نتیجه به سادگی نمی‌توانند از اتصالات عادی عبور کنند. بعد از رسیدن به یک سیستم، پکت حجیم تمام پکت‌های ذخیره شده در این سیستم را از بین می‌برد. همچنین، هر سیم در شبکه حداکثر تحمل سه بار عبور پکت‌های حجیم را خواهد داشت و **پس از سه بار عبور پکت‌های حجیم این سیم از بین خواهد رفت**! در نتیجه شما باید علاوه بر قرار دادن سیستم‌های Distributor و Merger در (شبکه) ابتدا و انتهای شبکه، شبکه‌ای تحمل‌پذیر طراحی کنید که در صورت از بین رفتن برخی از اتصالات، همچنان قابلیت عبور دادن پکت‌های حجیم را داشته باشد. اهمیت این مسئله در بخش‌های بعد مبرهن‌تر خواهد بود. یک پکت حجیم پس از عبور از یک سیستم Distributor به تعدادی (به مقدار اندازه پکت حجیم مربوطه) بیت‌پکت (پکت‌های پیام‌رسان از اندازه 1) تقسیم می‌شود. توجه کنید بیت‌پکت‌های پکت‌های حجیم متمایز را با رنگ‌های مختلف از هم تمییز دهید. این بیت‌پکت‌ها سپس به صورت عادی در شبکه به گردش در می‌آیند و شما به عنوان اپراتور شبکه موظفید طوری برنامه‌ریزی کنید که تمام بیت‌پکت‌های مربوط به یک پکت حجیم در نهایت در یک سیستم Merger جمع شوند و به یک پکت حجیم تبدیل شوند. توجه کنید انتقال یک بیت‌پکت به سیستم مرجع مفید نیست و مشمول Packet Loss خواهد بود. در صورتی که یک پکت حجیم از اندازه $N = n_1 +\cdots+ n_k$ در نهایت به صورت چند پکت حجیم از اندازه‌های $n_1, \cdots, n_k$ به سیستم‌های مرجع بازگردد، اندازه آن‌ها مجموعاً $\lfloor k\sqrt[k]{n_1 \cdots n_k} \rfloor$ خواهد بود و در نتیجه اندازه Packet Loss برابر با مقدار $n_1 +\cdots+ n_k - \lfloor k\sqrt[k]{n_1 \cdots n_k} \rfloor$ است.

پکت‌های حجیم با ورود به هر سیستم از طریق یک پورت، این پورت را به صورت تصادفی به یک پورت دیگر تبدیل می‌کند

توجه کنید سازگاری برای پکت‌های حجیم معنادار نیست و طبعاً پورت‌هایی متناظر با این پکت‌ها وجود نخواهد داشت

	اندازه: 8 واحد	با هر بار ورود به سیستم 8 سکه به شبکه اضافه می‌کند	حرکت آن بر روی سیم های صاف با سرعت ثابت و بر روی انحناها، با شتاب ثابت است. (مگر در اثر **Impact**)
	اندازه: 10 واحد	با هر بار ورود به سیستم 10 سکه به شبکه اضافه می‌کند	حرکت این پکت روی تمام سیم‌ها با سرعت ثابت است اما به ازای طی مسافت مشخصی روی سیم‌ها، مرکز آن به مقدار مشخصی از روی سیم منحرف می‌شود. (مشابه اثر **Impact** روی حرکت پکت‌های دیگر در حال حرکت)

باز هم به منو باز می‌گردیم!

مشابه فاز اخیر، در فروشگاه تعبیه شده در فاز اول کاربر باید بتواند در ازای مقداری از سکه‌هایش، قابلیت‌هایی را **تنها برای همان مرحله** خریداری کند تا همان لحظه شروع به اثر کنند. در این فاز باید قابلیت‌های زیر نیز در این فروشگاه قابل خریدن باشند:

- **Scroll of Aergia:** بازیکن می‌تواند با پرداخت 10 سکه و انتخاب یک نقطه از یکی از اتصالات شبکه، شتاب حرکت پکت‌های گذرنده از این نقطه را برای مدت زمان مشخصی (مثلا 20 ثانیه) صفر می‌کند و پکت ها (تا زمانی که مجدداً شتابی به آنها اعمال نشده باشد) با سرعت ثابت روی سیم ها حرکت می‌کنند. توجه داشته باشید که استفاده از این قابلیت با cooldown همراه است یعنی بازیکن برای استفاده مجدد از آن باید برای مدت زمان مشخصی صبر کند.
- **Scroll of Sisyphus:** با پرداخت 15 سکه، بازیکن می‌تواند یکی از سیستم‌های غیر مرجع را تا شعاعی مشخص جابه‌جا کند. دقت کنید در استفاده از این قابلیت، طول سیم‌ها نباید از طول سیم موجود در مرحله تجاوز کند و همچنین جابجایی سیستم‌ها، در صورتی که باعث عبور سیم‌ها از سیستم‌ها شود غیرمجاز است و نباید انجام‌پذیر باشد.
- **Scroll of Eliphas:** با پرداخت 20 سکه و انتخاب یک نقطه از یکی از اتصالات شبکه، این قابلیت برای مدت زمان مشخصی (مثلا 30 ثانیه) فعال می‌شود. تمام پکت‌هایی که از این نقطه از سیم عبور می‌کنند، مرکز ثقل خود را (که ممکن است تحت برخورد پکت‌ها و یا اثرات Impact جابجا شده باشد) به امتداد سیم باز می‌گردانند. توجه کنید این حرکت باید به صورت پیوسته انجام شود زیرا در غیر این صورت وقوع Tunneling و به هم خوردن منطق برخورد محتمل خواهد بود.

ذخیرهٔ بازی

یکی از موارد مهم که همواره در پیاده‌سازی بازی‌ها (بالاخص بازی‌های fast-paced) ذخیره کردن وضعیت بازیست، تا بتوانیم در زمان‌های تعیین شده پیشروی بازی را ذخیره کنیم. البته از آنجا که همیشه زندگی بر وفق مراد نیست، ممکن است بر اثر کرش در بازی یا سیستم عامل، خاموش شدن اتفاقی سیستم و یا حتی وجود تروجانی مثل 💀YouAreAnIdiot، بازی در لحظه‌ای پیش‌بینی نشده متوقف شود! از این رو برای جلوگیری از از دست دادن پیشروی کاربر در این لحظات بغرنج، به Real-time progress save یا فرایند سیو بازی در لحظه نیاز داریم. در نتیجه، یکی از موارد مورد انتظار از شما در این فاز، فرآیندی‌ست که بتواند در حین بازی، اطلاعات مهم آن را در یک فایل (با فرمت و روش دلخواه)، ثبت و ذخیره کند و بعداً آنها را مجدداً اجرا کند. استیت ذخیره شده باید شامل **تمام** اجزای وضعیت ذخیره شده بازی باشد (اعم از همه سیستم‌ها و پکت‌ها، جهت و سرعت حرکت همه پکت‌ها و غیره) بدین معنا که لود شدن مجدد فایل سیو با ادامه بازی تفاوتی نکند. دقت کنید در این بازی مکانیک سیو شما شامل سیو در هنگام خروج از مرحله و همچنین سیو مرتب بازی در بازه‌های زمانی مختلف است.

امتیازی (Save Validation): در این بخش شما باید ساز و کاری امن برای لود کردن بازی ذخیره‌سازی شده طراحی کنید. برای مثال فرض کنید که شما در فایل سیو بازی، برای هر پکت یک فلگ تعبیه کرده باشید که سالم به مقصد رسیدن آن پکت را مشخص می‌کند. در این صورت یک بازیکن طمع‌کار با دسترسی به فایل سیو و جایگزین کردن آن با فایلی که در آن تمام این فلگ‌ها true هستند، می‌تواند تمام مراحل شما را به سادگی دور بزند. برای جلوگیری از تقلب در بازی خود، شما باید یا دسترسی به فایل سیو لحظه‌ای را غیر ممکن کنید و یا در خوانش این فایل سیو تقلبی به عنوان سیو چک‌پوینت مشکلی ایجاد کنید تا بازیکن‌ها موفق به دور زدن مکانیزم سیو شما نشوند. هر راهکار دیگری در راستای این مشکل قابل قبول خواهد بود.

قبل از شروع یک بازی جدید، (در صورتی که فایل سیو لحظه‌ای از بازی قبل باقی مانده باشد) شما باید پیغامی به کاربر نشان داده و از او بپرسید که آیا مایل است بازی سیو شدهٔ قبلی (که قبل از پایان موفقیت‌آمیز مرحله قطع شده است) را ادامه دهد یا خیر. در صورتی که کاربر این پیشنهاد را رد کند، این فایل حذف می‌شود و بازی جدیدی شروع می‌شود اما در صورتی که کاربر این پیشنهاد را قبول کند، شما باید ابتدا به مدت چند ثانیه وضعیت بازی را بی حرکت به او نشان دهید و سپس بازی را ادامه دهید. دقت کنید در صورت خروج از بازی از طریق منوی تعبیه شده در بازی، شما نباید این فایل سیو لحظه‌ای را ذخیره کنید و این فایل سیو صرفاً برای خروج‌های پیش‌بینی نشده در فرایند بازی خواهد بود. بنابراین شما باید در طول اجرای بازی، آن را با نرخ ثابتی (هر چند ثانیه یک بار) ذخیره کنید تا مانع از از دست رفتن اطلاعات بازی در این مواقع شوید.

امتیازی (Save Synchronization): در فرایند سیو یک بازی fast-paced با تعداد زیادی فیلد و آبجکت، علاوه بر مشکلات Concurrency معمول که در فرایند سیو بازی رخ می‌دهد، یکی از مسائل مهم همزمانی داده‌های رونوشت شده است. به طور خلاصه ممکن است شما در لحظه‌ای در کد خود تصمیم به ذخیره‌سازی بازی بگیرید اما چون فرایند سیو به ترتیب روی آبجکت‌های مختلف در حال انجام است، در طول این فرایند تغییری در وضعیت بازی (حتی در حد حرکت یک انتیتی به اندازه یک پیکسل) رخ دهد و بعضی از آبجکت‌های شما قبل از این تغییر و مابقی پس از این تغییر ذخیره شده باشند. در این صورت دیتای ذخیره شده شما لزوما با هم همخوانی نخواهد داشت. (برای مثال ممکن است در دیتای ذخیره شده شما دو پکت که در طی بازی با هم برخورد نداشته‌اند، در فایل سیو شما با هم برخورد کرده باشند) شما در این بخش باید با طراحی یک روند سیو مناسب، این مشکل را از سر راه بردارید و مطمئن شوید دیتای ذخیره‌سازی شده شما synchronized شده است.

گریزی مجدد به SOLID

یکی از مشکلات بنیادی در روند پیاده‌سازی، ایجاد Cyclic Dependency در ساختار کد است. برای اینکه شهود مناسبی از این مشکل داشته باشید، تصور کنید که سیستم برای لود یک فایل سیو، به ترتیب مشخصی serialization های تولید شده در فرایند سیو را به آبجکت‌هایی در کد شما تبدیل می‌کند. در نتیجه نیاز است قبل از ایجاد هر آبجکت، تمام فیلدهای آن آبجکت‌هایی شناخته شده باشند. حال فرض کنید شما آبجکت‌های X_1, X_2,...,X_n را در ساختار کد خود دارید که در آن، X_1 به عنوان فیلدی از X_2، X_2 به عنوان فیلدی از X_3 و... و X_n به عنوان فیلدی از X_1 معرفی شده است. در این صورت سیستم برای لود فایل شما، مستقل از اینکه از کدام آبجکت آغاز به deserialization کند، با آبجکت ناشناخته‌ای برای معرفی به عنوان یک فیلد مواجه خواهد شد که باعث ایجاد Compilation Error در کد شما می‌شود. با اینکه حل این مشکل معمولاً به تغییرات بنیادی در کد نیاز خواهد داشت، یکی از راه‌های جلوگیری از این مشکل در ساختار کد، رعایت اصول SOLID است. در نتیجه توصیه می‌شود اصول پایه‌ای SOLID را به درستی در کلاس‌هایی که نیاز به ذخیره شدن اطلاعات دارند، رعایت کنید و همانطور که در مقدمه نیز اشاره شد، کلاس‌های خود را تا حد امکان به ساختار POJO نزدیک نگه دارید تا از این ایراد معماری مصون بمانید!

طراحی مراحل مختلف بازی

در این قسمت شما باید مجموعاً 5 مرحله مختلف برای آزمایش جنبه‌های مختلفی که در طی این دو فاز پیاده‌سازی شده‌اند طراحی کنید. نکته بازی در اینجاست که **مراحل طراحی شده شما باید متوالی باشند!** این بدین معناست که اگر با یک شبکه مشخص، یکی از مراحل را به پایان رساندید، در مرحله بعدی سیستم‌های جدید به همان شبکه (با همان اتصالات انتخاب شده) اضافه خواهند شد و در نتیجه بازیکن برای بقا در مراحل بعدی بازی باید سعی کند شبکه‌ای بهینه و تمیز و به دور از درهم‌تنیدگی ایجاد کند. همچنین اتصالات ایجاد شده در مراحل قبلی در جدیدترین مرحله قابل حذف کردن نخواهند بود و در صورتی که شبکه تولید شده در مراحل پیشین بیش از حد درهم‌تنیده باشد، بازیکن مجبور خواهد بود بازی را از ابتدا آغاز کند! انتظار می‌رود مراحل طراحی شده از کیفیت مناسب برخوردار باشند به نحوی که مشخصه‌های متفاوت بازی در آن قابل مشاهده باشد و بتوان قوانین بازی را در آنها آزمایش کرد.

مختومه‌ای بر Game Design

همانطور که در این دو فاز دیدید، بازی تعریف شده تفاوت‌های بزرگی با مکانیک‌های بازی اصلی دارد که عمده این تغییرات برای متوازن‌تر کردن مکانیزم‌های بازی هستند. در طی تولید و طراحی یک بازی، نکات تئوری زیادی وجود دارند که می‌توانند در عین ریز و کم اهمیت بودن، بازی‌ها را بسیار جذاب یا خیلی حوصله سر بر و تکراری کنند. چنین نکات ریزی در مکانیک‌ها، Economy و فیزیک بازی‌ها باعث می‌شوند بازیکن بدون اینکه حتی خودش متوجه باشد، مستقل از اینکه بازی در ژانر موردعلاقه‌اش باشد، درگیر مکانیک‌های بازی شود. از این رو در نهایت بازی‌هایی بسیار ساده مثل Stardew Valley, Minecraft موفق‌تر از بازی‌هایی با گرافیک بسیار پیچیده‌تر مثل Demon's Souls و Final Fantasy ظاهر شوند.

فاکتورهایی مثل Death Design, Game Economy, Combat System, Camera Design و مواردی از این دست تفاوت‌های شگرفی در درگیر شدن پلیرها در بازی ایجاد می‌کنند و در نتیجه بازی‌هایی که در این موارد به نحو احسن عمل می‌کنند، در نهایت هم بسیار موفق‌تر از دیگر بازی‌ها خواهند بود. لینک‌های زیر برای علاقمندان به این موضوعات قرار داده می‌شود:

1. GMTK: Design of games' economy
2. GMTK: Design of combat systems
3. GMTK: How physics affect platformer games
4. GMTK: Analysis of video game designs
5. Designing better game economies

و باز هم بخش‌های امتیازی!

Save Validation

Save Synchronization

استفاده از روش‌های مبتنی بر PCA

همانطور که تا در این داک دیدید، در این بازی بعضاً به بررسی برخورد بین انتیتی‌هایی که لزوما چندضلعی نیستند، نیازمندیم. به طور کلی در این بازی برای انتیتی‌های غیر چندضلعی تنها بررسی برخورد با هیت‌باکس‌های مستطیلی از شما انتظار خواهد رفت. اما دقیق کردن این برخوردها، علاوه بر جلوه بصری معقول‌تر، جنبه استراتژیک بازی را پررنگ‌تر خواهد کرد. در این بخش، دقیق کردن برخوردها برای انتیتی‌های دلخواه با روش‌های Polygonization نمره امتیازی به همراه خواهد داشت. مطالعه محتوای این کارگاه می‌تواند مفید باشد.

پیاده‌سازی گشتاور دورانی

در این بخش، شما باید گونه‌ای از گشتاور دورانی را پیاده‌سازی کنید. وضعیتی را متصور شوید که یکی از پکت‌ها به پکت دیگری در شبکه برخورد می‌کند. طبق مکانیک Impact شما باید یک موج ضربه دقیقا از این نقطه برخورد ساطع کنید که همه پکت‌های دیگر (حاضر روی اتصالات شبکه) را به عقب براند. اما در این بخش شما همچنین باید گشتاور دورانی حاصل از این ضربه را نیز اعمال کنید که باعث می‌شود هر یک از پکت‌های تحت اثر این موج Impact تا مدت زمان معینی در حین جابجایی خود روی سیم‌ها حول مرکز ثقل خود نیز در دوران باشد. مثال و توضیحات کامل در این زمینه در این لینک آمده است. همچنین می‌توانید قسمت‌های مربوطه در کد پیشنهادی برای فاز اول پروژه سال اخیر را مطالعه کنید.

Log

در این بخش از شما انتظار می‌رود با استفاده از لایبرری‌هایی اعم از Slf4j, Logback, Log4j,... تمام وقایع کد خود را به صورت لاگ در یک فایل قابل خواندن ذخیره کنید. لاگ‌های شما باید شامل تمام حرکت‌ها، برخورد ها، خرید های فروشگاه، قابلیت های فعال، packet loss و غیره باشد. برای بخش شما باید دسته بندی مناسبی از لاگ‌ها ارائه دهید و آنها را به درستی در سطوح SEVERE, WARNING, INFO, DEBUG طبقه‌بندی کنید. نوشتن یک لاگ کامل همچنین می‌تواند برای دیباگ کد، بررسی کرش‌ها و (در سطوح بالاتر) حتی مطالعه رفتار پلیرها به شما کمک کند.


با آرزوی موفقیت و سربلندی شما
تیم درس برنامه‌نویسی پیشرفته
دانشگاه صنعتی شریف