برنامه‌نویسی پیشرفته

زمستان 1403 - بهار 1404

فاز سوم پروژه

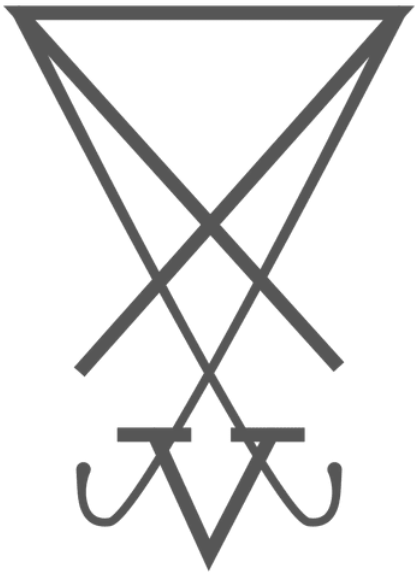
تیم طراحی و تعریف پروژه: آرین همتی، محمدرضا رحمانی، مهدی زاهدی

آیدی‌ها روی اسامی لینک شده‌اند

مقدمه‌ای بر فاز سوم (مثل همیشه حتماً بخوانید!)

به فاز سوم و پایانی پروژه درس برنامه‌نویسی پیشرفته خوش آمدید. در دو فاز اول پروژه (😰) شما یک بازی کامل شامل تعداد زیادی سیستم و پکت با پروتکل‌ها و کارکردهای حرکتی متفاوت پیاده‌سازی کردید و در طی این پروسه با فرایندهای پیشرفته گرافیکی، اصول معماری و طراحی و پارادایم‌های مختلف برنامه‌نویسی آشنا شدید و از آنها برای پیاده‌سازی یک کد توسعه‌پذیر و قابل‌فهم در راستای اصول SOLID استفاده کردید. همچنین شما در فاز دوم پروژه توانستید با طراحی‌های Event-Driven، کارکردهای مربوط به اجزای مجزای بازی را به یک‌دیگر مرتبط و متصل کنید. در ادامه این فاز، با استفاده از اصول OOP، رفتارها و پروتکل‌های متفاوتی برای اجزای مختلف بازی اعم از فریم‌ها و دشمن‌ها مختلف تعریف کرده و بدون ارتباط مستقیم آنها، رفتارهای متقابل بین این اجزا را کنترل کنید. در نهایت برای فرایندهای ذخیره‌سازی بازی و طراحی مراحل بازی از روندهای Event Queuing, Serialization استفاده کردید و کار خود را در این فاز به پایان رساندید. از شما انتظار می‌رود در طول فازهای اخیر با استفاده از دانش خود در زمینه معماری و طراحی روندهای اجرا و Workflow، نرم‌افزاری هم‌خوان با معماری Event-Driven و بالاخص هم‌سو با معماری MVC پیاده‌سازی کرده باشید که از توسعه‌پذیری مطلوبی برخوردار باشد. توصیه می‌شود با توجه به نسبت مهلت این فاز به محتوای مورد انتظار، در صورتی که کد شما در راستای این معماری‌ها پیاده‌سازی نشده است، برای پیاده‌سازی بدون اشکال این فاز زمان کافی برای اصلاح کد فازهای قبل خود اختصاص دهید. همچنین توجه کنید با توجه به ذات معماری‌های مبتنی بر شبکه، احتمال وقوع مشکلات Concurrency در روند اجرای برنامه بالا خواهد بود و توصیه می‌شود قبل از پیاده‌سازی این فاز، به رفع مشکلاتی از این قبیل هم توجه کافی داشته باشید. از آنجا که در این فاز به پروسه‌های پیشرفته‌تر ذخیره‌سازی احتیاج خواهید داشت، ممکن است برای پیاده‌سازی فرایندهای Serialization, Deserialization به Serialization Annotation ها نیاز پیدا کنید. در نهایت همانطور که از طریق ایمیل اطلاع پیدا کرده‌اید، یکی از سری‌های تمرین درس (در موضوع Reflection) از برنامه درسی حذف شده و نمره آن به این فاز انتقال پیدا کرده است. لذا از آنجا که در طراحی بازی‌ها، Responsive بودن همواره یکی از مهم‌ترین اصول پیاده‌سازی‌ست، خوب است برای جلوگیری از مشکلات احتمالی استفاده از Reflection در یک کد جاوا، با تغییراتی که این کار لحاظ زمانی و منابع برای اجرای بازی شما ایجاد می‌کند آشنا باشید. این لینک برای مطالعه درباره این دست از رفتارهایی که به علت استفاده از Reflection ایجاد می‌شوند منبع مفیدی خواهد بود. اهداف آموزشی این فاز شامل مباحثی اعم از Network, Data Stream, Reflection خواهند بود و از شما انتظار می‌رود (همانطور که در ادامه داک توضیح داده خواهد شد) با استفاده از دانشی که در طول درس درباره این مباحث کسب کرده‌اید، در قسمت‌های مختلف کد خود از پروتکل‌های ارتباطی صحیح و مناسب استفاده کنید و معماری‌های Stream متفاوتی برای رد و بدل کردن Request/Response های خود به کار بگیرید.

نکته بسیار مهم: در تعریف فاز آخر پروژه، با توجه به مهلت کمتر آن نسبت به فازهای قبلی عمدتاً سعی در این راستا بوده که محتوای فاز، غالباً حول محور اهداف آموزشی باشند و از تعریف آیتم‌های جدید در این فاز جلوگیری شود. لذا بدیهی‌ست که بخش قابل توجهی از نمره این فاز، به صورت مستقیم یا غیرمستقیم وابسته به پیاده‌سازی آیتم‌های فازهای قبل است. با اینکه سعی بر این است که این وابستگی کمینه باشد، در حدود یک نمره از 3.5 نمره این فاز (با احتساب نمره اضافه شده از تمرین Reflection) مربوط به آیتم‌های فاز دوم خواهد بود و سعی بر این است که باقی نمره این فاز با استفاده از ابزارهای فاز اول پروژه قابل پیاده‌سازی باشد. در حدود 20 درصد نمره فاز این فاز نیز مشابه فاز اخیر به موضوعات مربوط به کلین‌کدینگ و رعایت صحیح و بجای معماری‌ها و دیزاین‌پترن‌های مختلف اختصاص خواهد داشت.



بازی آنلاین!

یک از اهداف اصلی این فاز، تبدیل بازی‌ای که در فازهای قبلی پیاده کرده‌اید به یک بازی شبکه‌محور است. در این فاز شما باید کد خود را در قالب دو برنامه مجزای جاوا تحویل دهید که هر یک به طور مجزا مسئولیت کارکردهای Server, Client را بر عهده می‌گیرند. اگر در فازهای قبلی به درستی از معماری MVC بهره گرفته باشید، این گام برای شما بسیار ساده خواهد بود. دقت کنید از آنجا که هر یک از برنامه‌های مربوط به کدهای ارسالی Server, Client باید مستقلاً و به درستی قابل اجرا باشند، هیچ‌گونه Dependency مستقیمی نباید بین این دو کد وجود داشته باشد. اما بدیهیست برای فعالیت هم‌زمان و Synchronized بین این دو قسمت، ارتباطاتی بین Sever, Client مورد نیاز است تا:

1. فعالیت ورودی کاربر را از سمت Client به سمت Server مخابره کرده و باعث پیش‌روی بازی شوند
2. در هر فریم اطلاعات جدید بازی را (پس از پیش‌روی) از سمت Server (جهت نمایش فریم جدید) به Client مخابره کنند

به این منظور از شما انتظار می‌رود در قسمت Server,Client کد ارسالی خود وظایف زیر را پیاده‌سازی کنید:

1. در شروع اجرای بازی از سمت Client، تلاش کنید به آدرس از پیش تعیین شده‌ای مربوط به Server متصل شوید
2. در صورت موفقیت در اتصال به این آدرس، پیامی به کاربر نمایش دهید و منوی اصلی بازی را (مانند شروع فاز دوم) نمایش دهید
3. در صورت عدم موفقیت در اتصال به آدرس Server، این مشکل را به همراه گزینه‌ای برای تلاش مجدد به کاربر اطلاع دهید. همچنین باید گزینه‌ای برای اجرای بازی بدون اتصال به Server در نظر بگیرید. (توضیحات لازم در ادامه آمده است) در هر صورت (اجرای بازی با اتصال به Server یا بدون آن) شما باید گزینه‌ای برای نمایش وضعیت اتصال و قطع/وصل کردن ارتباط با Server در منوی اصلی بازی در نظر بگیرید. همچنین در صورت قطع شدن ارتباط با Server در هر نقطه‌ای از بازی، باید آن را با در قالب یک پیام به کاربر اطلاع دهید
4. قسمت Server کد ارسالی شما باید همواره آماده دریافت اتصال از سمت یک Client جدید بوده و لیست کاملی از Client های متصل به Server داشته باشد. همچنین Server شما باید همواره آماده دریافت Request های هر Client متصل و پاسخ دادن به آنها باشد.

به منظور انجام این ارتباطات از شما انتظار می‌رود در هر مورد از پروتکل ارتباطی مناسب (QUIC,UDP,TCP,...) برای انتقال داده‌های موردنیاز استفاده کنید. همچنین استفاده صحیح از دیزاین‌پترن‌های ارتباطی (Request-Response, Pulling,...) موردانتظار است. برای مطالعه بیشتر درباره انواع دیزاین‌پترن‌های ارتباطی می‌توانید این منبع را مطالعه کنید. همچنین مطالعه این لینک برای یادگیری بهتر QUIC سودمند خواهد بود.
توجه کنید کد Server شما باید بتواند پاسخگوی چند Client همزمان باشد و پیام‌های دریافتی/ارسالی را بدون تداخل دریافت و پردازش کند

Leaderboard

در این بخش از شما انتظار می‌رود در پایان هر بازی یک جدول امتیازات کامل برای کاربر نمایش دهید که در آن رکورد کمترین مدت زمان به پایان رساندن هر مرحله و تمام بازی و XP به دست آمده او در آن بازی و همچنین رکورد بیشترین XP کسب شده در یک بازی در بین همه بازیکنان دیگر (با احتساب خودش) با نام کاربری آنها نمایش داده شده است. همچنین می‌توانید آمارهای دیگری به سلیقه خود در این بخش نمایش دهید.

قابلیت بازی آفلاین!

با اینکه عمده هدف این فاز از پروژه پیاده‌سازی بازی به صورت آنلاین است، از شما انتظار می‌رود بتوانید بازی را طوری طراحی کنید که کاربر کماکان قادر به بازی آفلاین و بدون اتصال به Server باشد. به این منظور شما باید نتایج بازی‌های آفلاین کاربر را برای ارسال به Server در اولین فرصت، در فایلی ذخیره کنید. همچنین تبعاً قابلیت‌های آنلاین بازی در این حالت برای کاربر غیر فعال خواهند بود.

Data Integrity Validation (DIV)

یکی از مشکلات همیشگی پیاده‌سازی بازی‌های Multiplayer آنلاین، اطمینان حاصل کردن از درستی داده‌های غیر مطمئن است. برای مثال یک کاربر صرفاً با کشف Response های ارسالی به سرور پس از پایان بازی می‌تواند امتیاز دلخواه و غیرواقعی از نتیجه بازی‌های آفلاین خود را به سمت Server مخابره کند. برای جلوگیری از این مشکلات، شما باید متناسب با معماری و نحوه پیاده‌سازی خود روش‌هایی برای اطمینان حاصل کردن از دیتای ارسالی به سمت Server طراحی و اجرا کنید. برای آشنایی با یکی از راه‌حل‌های معمول برای این مشکل می‌توانید این لینک را مطالعه کنید.

دیتا هندلینگ سمت Server

در این فاز، همانطور که منطقی و واضح بنظر می‌رسد، شما جز در حالت بازی آفلاین، تمام داده‌های همه بازیکنان را در سمت Server کد خود ذخیره می‌کنید و سپس با استفاده از روندهای اجرای مختلف (اعم از On-call, Consistent) دیتای مورد نیاز برای اجرای هم‌روند دو بخش مجزای بازی خود را از Server دریافت و یا به آن ارسال می‌کنید. از این رو شما نیازمندید تمام اطلاعات مربوط به هر کاربر شامل نام کاربری، Squad مربوطه، مقدار XP، قابلیت‌های باز شده در بازی و قابلیت فعال، تاریخچه امتیازات و غیره را در قالب یک فایل به ازای هر کاربر ذخیره و دائماً آپدیت کنید. برای تشخیص فایل سیو مربوط به هر کاربر می‌توانید Mac Address سیستم کاربر را در بدو اتصال به سرور فراخوانی کرده و فرایند شناسایی کاربر در روند ذخیره‌سازی اطلاعات را با استفاده از این مقدار انجام دهید. همچنین اطلاعات مربوط به بازی شامل تمام مقادیر ثابت مطلق، (مقادیر ثابتی که از ثوابت دیگر بازی محاسبه نمی‌شوند و ذاتاً ثابت هستند) اطلاعات مربوط به Squad ها اعم از XP جمع‌آوری شده و اعضا و نبردهای بین Squad ها و تمام موارد دیگر که نیاز به ذخیره‌سازی دارند را مطابق روند ذخیره‌سازی در فاز اخیر در قالب چند فایل ذخیره کنید. پیاده‌سازی موارد امتیازی مربوطه فاز دوم در این فاز توصیه می‌شود. در این قسمت می‌توانید قسمت امتیازی این بخش را نیز پیاده‌سازی کنید.

اپراتور در برابر اپراتور، قابلیت بازی Multiplayer

در این فاز از پروژه، شما باید یک بازی مولتی‌پلیر و دو نفره طراحی کنید که در آن دو اپراتور تلاش می‌کنند با ایجاد اتصالاتی بین سیستم‌های داده شده (که برای هر دو بازیکن بازی یکسان و مشترک است)، شبکه‌ای را طراحی کنند و با هم رقابت کنند تا تعداد بیشتری پکت را سالم به مقصد برسانند. در فاز پیش‌رو، برخی سیستم‌های مرجع می‌توانند توسط بازیکن‌های بازی Multiplayer کنترل شوند تا کار را برای رقیب سخت‌تر کنند! در این فاز پکت‌ها در دو رنگ قابل مشاهده خواهند بود. پکت‌های رنگ اول متعلق به بازیکن اول و پکت‌های رنگ دوم متعلق به بازیکن دوم خواهند بود. در نتیجه از دست رفتن یک پکت نوع اول در Packet Loss بازیکن اول (و مشابهاً برای نوع دوم) محاسبه خواهد شد.

فرصت اولیه ساخت شبکه

در ابتدا هر دو بازیکن بازی مدت مشخصی (مثلاً 30 ثانیه) فرصت دارند تا اتصالات دلخواه خود را بین سیستم های داده شده را ایجاد کنند، همچنین اگر بازیکنی به زمان بیشتری نیاز داشت تا 30 ثانیه دیگر با دریافت جریمه می تواند اتصالات خود را ایجاد کند. مکانیک‌های جریمه باید به نقش یک Negative Feedback Loop را در این بخش ایفا خواهند کرد. شما در این بخش باید دکمه‌ای طراحی کنید که با فشردن آن، بازیکن‌ها قادر باشند آمادگی خود را برای شروع بازی اعلام کنند. اعلام آمادگی برای شروع بازی برگشت‌پذیر نخواهد بود.

- Wrath of Penia : در 10 ثانیه اول زمان اضافه، هر 2 ثانیه یک پکت تصادفی به یک سیستم مرجع کنترل‌پذیر حریف اضافه می‌شود
- Wrath of Aergia : در 10 ثانیه دوم زمان اضافه، به ازای هر ثانیه Cooldown بازیکن 1 درصد افزایش می‌یابد
- Wrath of Penia : در 10 ثانیه سوم زمان اضافه، به ازای هر ثانیه سرعت تمام پکت‌ها 3 درصد نسبت به قبل افزایش پیدا می‌کند

سیستم‌های مرجع کنترل‌پذیر، سلاح مخفی اپراتورها

در فازهای قبل انتشار پکت‌ها از سیستم‌های مرجع به صورت از پیش تعیین شده و برنامه‌ریزی شده اتفاق می‌افتاد اما در بازی مولتی‌پلیر، کنترل انتشار پکت‌ها از برخی از سیستم‌های مرجع در دست بازیکن‌ها خواهد بود. در این فاز بعضی از سیستم‌های مرجع "کنترل‌پذیر" خواهند بود. بازیکن اول با استفاده از یک سیستم مرجع کنترل‌پذیر می‌تواند پکت‌هایی را در شبکه بازیکن دوم منتشر کند و همچنین بالعکس، بازیکن دوم قادر خواهد بود با استفاده از این سیستم‌ها پکت‌هایی را در شبکه بازیکن اول منتشر کند. هر سیستم مرجع کنترل‌پذیر تعداد مشخصی "مهمات" (که شامل پکت‌هایی از انواع مختلف هستند) در خود ذخیره می‌کند. دقت کنید مهمات نوع اول ذخیره شده در سیستم مرجع کنترل‌پذیر برای بازیکن دوم و مهمات نوع دوم ذخیره شده در سیستم مرجع کنترل‌پذیر برای بازیکن اول قابل استفاده است. هر بازیکن با انتخاب هر کدام از سیستم‌های مرجع باید بتواند در پنجره‌ای کوچک (ترجیحاً Hovering Panel) تصویر هر نوع پکت و تعداد باقی مانده آن را ببیند و در صورت تمایل یکی از آنها را برای انتشار در شبکه بازیکن حریف انتخاب کند. سپس پکت انتخابی وارد شبکه بازیکن حریف می‌شود و هم سیستم و هم پکت انتخابی وارد حالت Cooldown خواهند شد. دقت کنید Cooldown انواع پکت تنها برای بازیکنی که از این سیستم برای انتشار پکت استفاده کرده وضع خواهند شد و اثری روی استفاده بازیکن مقابل نخواهند داشت اما Cooldown سیستم برای هر دو بازیکن وضع می‌شود. این قابلیت از انتشار متوالی چندین پکت در شبکه توسط بازیکن‌ها جلوگیری می‌کند. بازیکن باید بتواند Cooldown باقی‌مانده هر پکت و سیستم‌های مرجع کنترل‌پذیر را مشاهده کند. در صورتی که Cooldown یک سیستم به پایان برسد، بازیکن‌ها می‌توانند مجدداً از این سیستم برای انتشار پکت‌ها استفاده کنند.

دقت کنید در این فاز همچنین برخی از سیستم‌های مرجع برای هر دو بازیکن کنترل‌ناپذیرند و عملکرد آن‌ها مشابه فاز اخیر و به صورت خودکار خواهد بود
این سیستم‌های مرجع به صورت یکی در میان پکت‌هایی از نوع اول و دوم را در شبکه منتشر می‌کنند تا تعادل بازی طی این مکانیک بر هم نخورد

اگر یک پکت در حالت Cooldown قرار بگیرد، تصویر آن در پنل مهمات سیستم مرجع کنترل‌پذیر مربوطه و برای بازیکن مربوطه باید تا زمان اتمام Cooldown خاکستری دیده شود. اگر تعداد یکی از پکت ها در مهمات یکی از بازیکن‌ها صفر شد، باید تصویر آن با رنگ قرمز نمایش داده شود. همچنین شما باید اتمام Cooldown یک پکت را (مشابهاً در پنل مهمات سیستم مرجع کنترل‌پذیر مربوطه و برای بازیکن مربوطه) با یک جلوه بصری (مانند لرزش یا هر جلوه دیگری به انتخاب خودتان) مشخص کنید. همچنین توصیه می‌شود یک شمارش معکوس برای Cooldown هر سیستم و هر پکت داشته باشید.برای متعادل شدن بازی توجه کنید Cooldown سیستم‌های مرجع کمترین Cooldown پکت‌ها بیشتر باشد.

توجه کنید پکت‌های نوع اول تنها از طریق اتصالات شبکه بازیکن اول و پکت‌های نوع دوم تنها از طریق اتصالات بازیکن دوم قابلیت حرکت دارند

همچنین دقت کنید شما باید قابلیت مشاهده شبکه طراحی شده توسط بازیکن مقابل را **پس از اتمام فرصت اولیه ساخت شبکه** برای هر بازیکن فراهم کنید. برای این کار می‌توانید شبکه بازیکن حریف را به صورت کم‌رنگ‌تر روی صفحه بازی هر بازیکن نمایش دهید. این قابلیت در کنار کنترل برخی از سیستم‌های مرجع باعث می‌شود بازیکن‌ها بتوانند از ضعف‌های طراحی یک‌دیگر برای برد بهره ببرند.

بازیکن‌ها در فرصت اولیه ساخت شبکه باید بتوانند با استفاده از Temporal Progress، وضعیت پکت‌های شبکه‌های مرجع کنترل‌پذیر را در وضعیت فعلی شبکه خودشان و شبکه بازیکن حریف بررسی کنند. طبیعتاً این گزینه پس از شروع بازی برای هر دو بازیکن غیرفعال خواهد بود

اکنون زمان شروع بازی است!

در این بازی دو بازیکن پس از طراحی شبکه‌های خود در مهلت تعیین شده بازی را آغاز می‌کنند. مهمات موجود در هر سیستم مرجع کنترل‌پذیر برای هر دو بازیکن باید برابر باشد. سپس شبکه آغاز به کار می‌کند و بازیکنان تلاش می‌کنند بیشترین تعداد پکت سالم را به مقصد برسانند. هر پکت از بین رفته (و مشمول Packet Loss) به اندازه 1.5 واحد تاثیر منفی روی تعداد پکت‌های موفق خواهد داشت. این مکانیزم ساده در Economy بازی باعث می‌شود که بازیکنان علاوه بر ساخت یک شبکه موفق، تلاش کنند شبکه‌ای طراحی کنند که پکت‌های بازیکن حریف را در خطر قرار دهد.

مروری بر پکت تروجان و پکت‌های محرمانه

شما در فاز قبل با خاصیت تروجان بودن یک پکت آشنا شدید. یک پکت تروجان با عبور یک پکت از یک سیستم خراب‌کار به وجود می‌آید و توسط سیستم‌های آنتی‌تروجان به نوع اولیه خود باز می‌گردد. اما ضرر اصلی تروجان شدن یک پکت در فاز قبل توضیح داده نشده است. با تروجان شدن یک پکت از نوع اول، این پکت به یک پکت مشابه از نوع دوم تبدیل خواهد شد! اگر این پکت تروجان توسط سیستم آنتی‌تروجان به نوع اولیه خود بازگردد، مجدداً به یک پکت نوع اول تبدیل می‌شود. (مشابهاً برای پکت‌های نوع دوم)

همچنین در فاز اخیر با **خواص پکت‌های محرمانه** آشنا شدید. این پکت‌ها در عبور از سیستم جاسوسی از بین می‌رفتند و مکانیزم حرکت آن‌ها به صورت تصادفی از پکت‌های پیام‌رسان انتخاب می‌شد. این پکت‌ها باید (مشابه فاز اخیر) از سیستم‌های مرجع قابل انتشار باشند اما انتشار آنها تنها از طریق سیستم‌های کنترل‌ناپذیر انجام می‌گیرد و قابلیت انتشار یک پکت محرمانه از طریق سیستم کنترل‌پذیر توسط بازیکن‌ها وجود ندارد.

یک Feedback Loop برای جذابیت بازی

استفاده از یک Feedback Loop می‌تواند در افزایش سرعت بازی‌ها نقش بزرگی ایفا کند. در این فاز، با رسیدن هر پکت از نوع اول/دوم به مقصد (به معنای رسیدن این پکت به یک سیستم مرجع)، یک واحد از این پکت از نوع دوم/اول به پکت‌هایی که بازیکن اول/دوم می‌تواند برای حریفش در شبکه منتشر کند اضافه خواهد شد. این قابلیت اهمیت سرعت واکنش را (به علت احتمال Monopoly) در بازی افزایش می‌دهد.

خودکارسازی حملات Wave ها

در این فاز از شما انتظار می‌رود روند انتشار پکت‌ها در سیستم‌های مرجع کنترل‌ناپذیر را به صورت خودکار پیاده‌سازی کنید و از فراخوانی دستی دشمن‌ها در مناطق مشخص بپرهیزید. در گام اول از شما انتظار می‌رود با داشتن لیستی از پکت‌های موجود در بازی (شامل پکت‌های خاص) و

حداقل زمان مورد نیاز برای ظاهر شدن آنها، (برای مثال نباید پکت‌های حجیم را در ابتدای شروع بازی در شبکه منتشر کنید) و همچنین میزان Cooldown پکت مربوطه، صرفاً با دریافت تعداد پکت‌های مورد انتشار به طور خودکار سلسله مراتبی از پکت‌ها برای منتشر شدن در شبکه ایجاد کنید. این کار به منحصر به فرد بودن تجربه بازیکن حتی بعد از چندین بار پایان بازی کمک کرده و کاربر را مجبور به استفاده از استراتژی‌های متفاوت می‌کند. در گام دوم از شما انتظار می‌رود با استفاده از ابزارهای Reflection، به طور خودکار پکت‌های موجود در بازی را به عنوان Subclass های یک کلاس اصلی پکت (که طبق اصول OOP باید کلاس تمام پکت‌های دیگر خود را از آن به ارث برده باشید) شناسایی کرده و فرایند انتشار پکت‌های مختلف در بازی را تقریباً به طور کامل خودکار کنید. با این کار، در صورتی که در طی توسعه بازی، پکت جدیدی به بازی اضافه شود، نیازی به اضافه کردن دستی آنها در فرایند کار سیستم‌های مرجع کنترل ناپذیر نخواهد بود که اصطلاحاً هزینه توسعه برنامه را کاهش می‌دهد. همانطور که در مقدمه فاز اشاره شد، از شما انتظار می‌رود با تسلط به روندهای اجرا در متدهای پکیج Reflection، از ایرادهای Workflow ناشی از استفاده از متدهای این پکیج در Coupling Point های کد خود جلوگیری کنید و لذا رعایت صحیح این موارد در نمره این بخش موثر خواهند بود.

یادداشت پایانی!

واقعاً خسته نباشید! امیدوارم این ترم و این پروژه براتون خوب و آموزنده بوده باشه و در بین این همه مطالب رندوم و عجیب غریب به چیزهایی علاقمند شده باشید که باعث بشه برید دنبالشون رو بگیرید و شاید دربارشون بیشتر مطالعه کنید! من و تیم درس براتون عمیقاً آرزوی موفقیت و سربلندی می‌کنیم و امیدواریم با هر نمره‌ای هم که این درس رو به پایان رسوندید، از یادگیریش لذت برده باشید و این درس براتون یه تجربه جالب (البته با وجود پر زحمت بودن 🔥) بوده باشه. امیدواریم سال‌های بعد هم شما توی تیم درس باشید و یه ورودی دیگه رو با چیزای جالب درباره برنامه‌نویسی آشنا کنید! برای همتون آرزوی موفقیت می‌کنیم و امیدواریم همیشه از یاد گرفتن لذت ببرید. با آرزوی موفقیت، تیم درس برنامه‌نویسی پیشرفته...

آخرین سری بخش‌های امتیازی!

API

در این فاز شما موظفید ارتباطات بین سرور و کاربر را با استفاده از یک API انجام دهید. بدین منظور یک فایل خام در این لینک در اختیار شما قرار داده شده است. شما باید از این پکیج‌ها در بالاترین لایه کد خود استفاده کنید و متدهای آن را تکمیل کنید. تاکید می‌شود که به هیچ عنوان اسم و نوع متدها، فیلدها، enumها و یا کلاس‌ها و یا پکیج‌ها را تغییر ندهید تا فرایند ارزیابی کد شما با اشکال مواجه نشود. می‌توانید در صورت لزوم فیلد یا متدی به کلاس‌های فایل اضافه کنید اما از کاستن آنها و یا تغییر نام کلاس‌ها و متدهایی که در فایل وجود دارند اکیداً بپرهیزید. در ادامه توضیحاتی درباره فایل الحاق شده ارائه می‌دهیم تا بتوانید به درستی با استفاده از آن، اتصال سرور و کاربرهای بازی را برقرار کنید:

Enum

- **PacketType**: انواع مختلف پکت‌های بازی را شامل می‌شود
- **SystemType**: انواع مختلف سیستم‌های بازی را شامل می‌شود
- **AbilityType**: انواع مختلف قابلیت‌های بازی را شامل می‌شود
- **PortType**: انواع مختلف پورت‌های یک سیستم را شامل می‌شود

DTO (Data Transfer Object)

- **PacketDTO**: آبجکت طراحی شده برای انتقال اطلاعات مربوط به پکت‌های بازی می‌باشد. فیلد اول (id) یک آیدی برای هر پکت ایجاد خواهد کرد که به شناسایی این پکت برای عملیات‌های آتی کمک خواهد کرد. فیلد packetType نوع دقیق پکت را به همراه خواهد داشت و فیلد Position موقعیت مکانی این پکت روی صفحه بازی را نشان خواهد داد.
- **PortDTO**: برای انتقال اطلاعات درباره پورت‌های هر سیستم طراحی می‌شود و حاوی یک آیدی و سیستم مربوط به آن پورت خواهد بود.
- **SystemDTO**: آبجکت طراحی شده برای انتقال اطلاعات مربوط به سیستم‌های بازی می‌باشد. فیلد اول (id) یک آیدی برای هر سیستم ایجاد خواهد کرد که به شناسایی این سیستم برای عملیات‌های آتی کمک خواهد کرد. فیلد systemType نوع دقیق سیستم را به همراه خواهد داشت و فیلدهای inputPorts, outputPorts لیستی از پورت‌های ورودی و خروجی این سیستم را به همراه خواهند داشت.
- **WireDTO**: این آبجکت برای کنترل و انتقال اطلاعات مربوط به سیم‌ها طراحی می‌شود. فیلد color رنگ سیم را مشخص خواهد کرد. فیلد sourceId آیدی پورت مبدا سیم را ذخیره می‌کند و فیلد destinationId نگهدارنده آیدی مربوط به پورت مقصد خواهد بود.
- **GameInfoDTO**: برای انتقال اطلاعات مربوط به بازی اعم از مدت زمان بازی (در فیلد spentTime) استفاده خواهد شد.

Database

یکی از ابزارهای مناسب برای ذخیره‌سازی داده‌های دسته‌بندی شده، استفاده از پایگاه‌های داده یا Database است. برای آشنایی مقدماتی با دیتابیس‌ها مطالعه این منبع سودمند خواهد بود. با اینکه ارتباط مستقیم مدل‌های در حال اجرا در Runtime با پایگاه داده از طریق JDBC,JPA و ابزارهای مشابه امکان‌پذیر است، استفاده از ابزارهای ORM به علت هم‌سویی با پارادایم‌های برنامه‌نویسی OOP و معماری MVC توصیه می‌شود. برای مطالعه درباره ORM ها می‌توانید این لینک را دنبال کنید. برای استفاده از ORM در جاوا Hibernate ORM شاید یکی از بهترین گزینه‌ها باشد. برای این بخش موارد زیر مورد انتظار خواهد بود:

1. استفاده صحیح و بجا از جدول‌ها و تایپ درست برای ستون‌های دیتابیس
2. تعریف روابط صحیح بین جدول‌های استفاده شده در ذخیره‌سازی داده‌های بازی
3. ذخیره‌سازی کامل داده‌ها در دیتابیس و عدم ذخیره‌سازی دیتا در قالب فایل‌های JSON
4. تخصیص صحیح ستون‌های مربوط به Primary Key, Index های هر یک از جدول‌های دیتابیس
5. در صورت عدم اتصال به Server، داده‌ها به صورت Local ذخیره شده و انتقال به Server در اولین فرصت بعد از اتصال

Global Exception Handling

یکی از قابلیت‌های کارای فریم‌ورک Spring Boot نسبت به جاوا، تفاوت در نحوه Exception Handling آن است. همانطور که می‌دانید، فرایند Exception Handling در جاوا عمدتاً با بلوک‌های try-catch انجام می‌گیرد. این در حالی‌ست که در Spring Boot فرایند Exception Handling با قابلیتی به نام Global Exception Handling انجام می‌شود. در این رویه با استفاده از Annotate کردن یک کلاس با `@ControllerAdvice` تمام Exception هایی که در هر Controller اتفاق می‌افتد را به این کلاس Redirect می‌کنیم. در ادامه در این کلاس متدهایی با Annotation `@ExceptionHandler` ایجاد می‌کنیم که رویه‌های هندل کردن Exception های Redirect شده به این کلاس را در بر خواهند داشت. در اینجا شما می‌توانید با throw کردن انواع ExceptionMessage ها و Java Error ها، خروجی قابل قبولی را در صورت وقوع Exception به کاربر نمایش دهید. مرکزیت بخشیدن به فرایند Exception Handling باعث می‌شود هزینه توسعه‌پذیری کد شما را از لحاظ هندل کردن خودکار Exception ها کاهش می‌دهد و باعث می‌شود فرایندهای Exception Handling در کدتان پایدار و قابل پیش‌بینی باشند. از طرف دیگر این عملیات فرایند Logging را در سطوح Warning, Severe بهبود می‌بخشد و Log کردن این سطح از کد را بسیار ساده و مجتمع می‌کند. وظیفه شما در این قسمت این است که با استفاده از ابزارهای Reflection در جاوا، رویه‌ای مشابه Global Exception Handling در Spring Boot را در جاوا پیاده‌سازی کنید. پیاده‌سازی Annotation های مربوطه در این قسمت ضروری‌ست، لذا توصیه می‌شود پیش از پیاده‌سازی آن لینک‌های زیر را مطالعه کنید:

1. Java Annotations (Oracle Documentation)
2. Java Errors, Java Exceptions (Oracle Documentation)
3. Lesson on Java Exception Handling (Oracle Documentation)
4. Lesson on Spring Boot Global Exception Handling


به امید آنکه مشغول چیزی باشید
که از انجام دادن آن لذت می‌برید
دانشگاه صنعتی شریف