**فصل اول:**

**مقدمه**

با توسعه فناوری و اینترنت، بازی‌های آنلاین دو بعدی و بازی‌های تحت شبکه روز به روز بیشتر مورد توجه قرار می‌گیرند. این بازی‌ها امکان ایجاد تعاملات چندنفره و همکاری بین بازیکنان را فراهم می‌کنند و تجربه‌ای اجتماعی و هیجانی را برای کاربران فراهم می‌کنند. بازی آنلاین دو بعدی به عنوان یک دسته‌بندی از بازی‌های تحت شبکه بررسی می‌شود.

**بازی آنلاین دو بعدی:**

بازی‌های آنلاین دو بعدی با استفاده از گرافیک‌ها و المان‌های دو بعدی، تجربه‌ای ساده و مفید از بازی را ارائه می‌دهند. این بازی‌ها معمولاً در محیط‌های ساده و شخصیت‌های کارتونی طراحی می‌شوند و به صورت آنلاین قابل بازی هستند.

****

تصویر 1-1 بازی‌های دو بعدی

از جمله مثال‌های معروف این بازی‌ها، می‌توان به بازی‌های منچ، شطرنج، دوز، پازل، بازی‌های آرکید و بازی‌های ماجراجویی اشاره کرد.

**متور بازی سازی (Game Engine):**

متور بازی سازی نرم افزاری هست که فضایی آسوده و راحتتر با قابلیت‌های پیشرفته جهت توسعه بازی را به طراح و توسعه‌دهنده بازی ارایه میکند. اما این پروژه از متور بازی سازی جهت توسعه و پیاده‌سازی بازی استفاده نمیکند و این موضوع باعث می‌شود که در توسعه مدل بازی‌های مختلف توسعه‌دهنده در انتخاب و روند و اجرا بازی و نرم‌افزار اختیار بیشتری داشته باشد.

**بازی تحت شبکه:**

بازی‌های تحت شبکه، با استفاده از قابلیت‌های شبکه و اتصال اینترنت، به بازیکنان امکان می‌دهند تا در یک محیط مجازی بازی کنند و بازیکنان دیگری را به چالش بکشند یا با آن‌ها همکاری کنند. این بازی‌ها معمولاً دارای عناصر چالش‌برانگیز، رقابت و همکاری هستند و در برخی موارد نیاز به تعامل زنده با دیگر بازیکنان دارند. بازی‌های تحت شبکه می‌توانند در انواع ژانرها از جمله بازی‌های ورزشی، استراتژیک و تیراندازی وجود داشته باشند.

**نرم افزارهای اجرای و مدیریت بازی‌ها (launcher):**

نرم افزارهای اجرا و مدیریت بازی‌های چند نفره مانند استیم، اپیک گیمز و یوپلی نرم‌افزارهایی هستند که امکان دسترسی به بازی‌های تحت شبکه را فراهم می‌کنند.



تصویر 2-1 نرم‌افزار های اجرا و مدیریت بازی‌های

این نرم‌افزارها به کاربران امکان می‌دهند تا بازی‌های خود را مدیریت کنند، بازی‌های جدید را خریداری کنند، با دوستان خود در تعامل باشند و در اتاق‌های/بازی های چندنفره شرکت کنند. این نرم‌افزارها علاوه بر ارائه امکانات برای بازی‌کنان، به توسعه‌دهندگان نیز امکان می‌دهند تا بازی‌های خود تحت قالب توسعه داده و در این بستر پیاده‌سازی کنند.

**فریم ورک (FrameWork):**

فریم ورک یم چارچوب به حساب می‌آید که معمولاً برنامه نویس ها برای توسعه و طراحی نرم‌افزار از آن استفاده می‌کنند. استفاده از فریم ورک به ساده شدن توسعه پردازش ها کمک می‌کند.

به عنوان مثال در توسعه یک نرم‌افزار در ویندوز با ماوس می‌توان المنت های فرم مورد نظر را چید و به سرعت برای آن کد نویسی کرد اما در این پروژه از فریم ورک این مدلی استفاده نشده و تنها از یک کتابخانه گرافیکی سطح متوسط استفاده شده است که برای قرار دادن یک دکمه یا ورودی متنی کاربر نیاز است کامپوننت آن را نوشته و سپس از آن استفاده کرد که این به این معنی است که از صفر تمامی مواد مورد نیاز را باید توسعه داد و بعد از آن‌ها استفاده کرد.



تصویر 3-1 کتابخانه گرافیکی

**کتابخانه گرافیکی SFML (Simple and Fast Multimedia Library):**

یکی از محبوب‌ترین کتابخانه‌های برنامه‌نویسی بازی‌ها و برنامه‌های گرافیکی در زبان C++ است. این کتابخانه قابلیت‌های گسترده‌ای برای رسم شکل‌ها، کنترل ورودی کاربر، پخش صدا و انیمیشن را فراهم می‌کند. این کتابخانه متن باز است و از سیستم عامل های مختلفی پشتیبانی میکند به عبارت دیگر قابلیت چند سکویی را دارد.

در زیر می‌توانید یک معرفی خلاصه‌ای از کتابخانه SFML را ببینید:

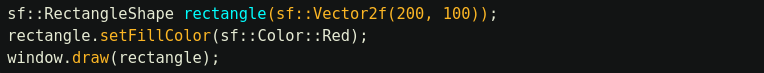
**ساخت پنجره**

با استفاده از SFML، می‌توانید به سادگی یک پنجره رسم کنید و آن را نمایش دهید. به این صورت می‌توانید یک پنجره با عرض 800 پیکسل و ارتفاع 600 پیکسل ایجاد کنید:

تصویر 4-1 کد ساخت پنجره

**رسم شکل‌ها:**

SFML امکان رسم اشکال ساده مانند مستطیل، دایره و خط را فراهم می‌کند. به عنوان مثال، می‌توانید یک مستطیل با ابعاد 200x100 پیکسل را رسم کنید:

تصویر 5-1 کد رسم شکل

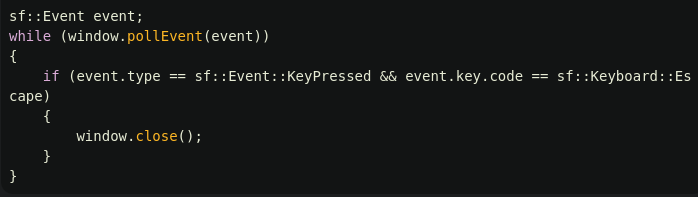
**پخش صدا:**

SFML قابلیت پخش فایل‌های صوتی را دارد. به عنوان مثال، می‌توانید یک فایل صوتی را بارگیری کنید و پخش کنید:

تصویر 6-1 کد پخش صدا

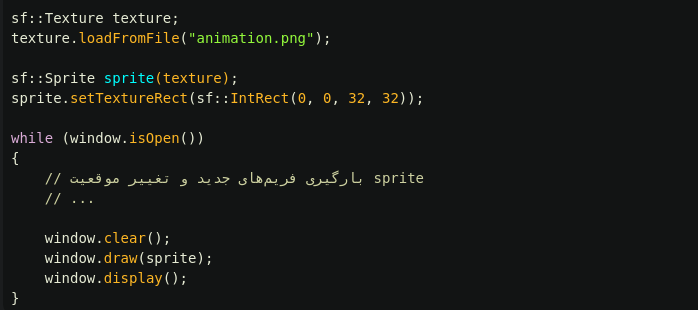
**کنترل ورودی کاربر:**

با استفاده از SFML، می‌توانید ورودی کاربر را دریافت کنید. به عنوان مثال، می‌توانید بررسی کنید که آیا کاربر دکمه Escape را فشرده است یا خیر:

تصویر 7-1 کد کنترول ورودی کاربر و رویداد ها

**انیمیشن:**

SFML امکان رسم انیمیشن‌ها را فراهم می‌کند. به عنوان مثال، می‌توانید یک تصویر را به عنوان فریم‌های متوالی بارگیری کنید و آن‌ها را به صورت انیمیشن نمایش دهید:

تصویر 8-1 کد انیمیشن و روح بازی

این معرفی شما را با برخیاز قابلیت‌های کلیدی کتابخانه SFML آشنا کند. با استفاده از این قابلیت‌ها، شما می‌توانید برنامه‌های گرافیکی متنوعی را با استفاده از SFML توسعه دهید. لازم به ذکر است که این شش تا ده خط برای معرفی کلیت کتابخانه SFML است و شما می‌توانید با نگاه به مستندات رسمی SFML و مثال‌های بیشتر، عملکرد و قابلیت‌های دقیق‌تر کتابخانه را بفهمید و در برنامه‌های خود استفاده نمایید.

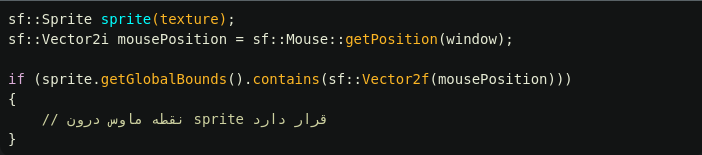
**تابع getGlobalBounds**

این تابع برای دریافت مستطیلی که شامل شکل مورد نظر (مانند sprite یا shape) استفاده می‌شود. به عنوان مثال، شما می‌توانید مستطیلی که شامل یک sprite است را با استفاده از `getGlobalBounds()` دریافت کنید:

تصویر 9-1 کد دریافت موقعیت های یک شی

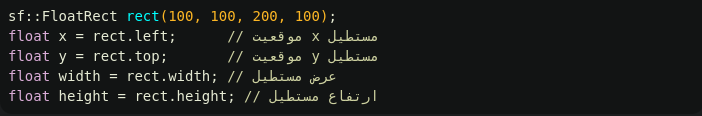
**تابع contains**

تابع `contains()` برای بررسی اینکه آیا یک نقطه درون یک مستطیل قرار دارد یا خیر استفاده می‌شود. به عنوان مثال، شما می‌توانید بررسی کنید که آیا نقطه ماوس درون یک sprite قرار دارد یا خیر:

تصویر 10-1 کد برسی مختصات و برخورد ماوس با شی

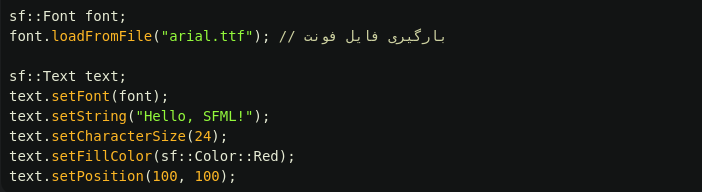
**موقعیت و ابعاد توسط نوع sf::FloatRect**

در این مثال، یک مستطیل ایجاد می‌شود و موقعیت و ابعاد آن در یک `sf::FloatRect` ذخیره می‌شود:

تصویر 11-1 کد موقعیت های یک شی

**نمایش متن sf::Text و sf::Font**

در این مثال، یک متن به کمک `sf::Text` ایجاد شده و فونت آن توسط `sf::Font` تنظیم می‌شود:

تصویر 12-1 کد ایجاد یک متن

**رنگ و نوع sf::Color**

در این مثال، یک شکل با استفاده از `sf::Color` و رنگ RGB قرمز ساخته می‌شود:

تصویر 13-1 کد رنگ دهی به شی

**شبکه sf::UdpSocket (ارسال و دریافت پیام با استفاده از UDP):**

در این مثال، یک `sf::UdpSocket` ایجاد می‌شود و از آن برای ارسال و دریافت پیام‌ها با استفاده از پروتکل UDP استفاده می‌شود:

تصویر 14-1 کد استفاده از شبکه با استفاده از کتابخانه

ولی مشکلاتی در این بخش کتابخانه وجود دارد برای مثال برای ارسال و دریافت اطلاعات با استفاده از UdpSocket درسته در یک Thread جداگانه موارد این متد را اجرا میکند ولی در دریافت و ارسال اطلاعات به مشکل خواهیم خورد در صورتی که اطلاعات کم باشند میتوان مورد استفاده قرار داد ولی برای مثال زمانی که از این متد در کد استفاده کردیم و نیاز بود هر ثانیه یک متد دیگر را اجرا کند یا اطلاعات را دریافت کند بصورت پیشفرض این متد کل برنامه را متوقف میکند تا اطلاعات دریافت کند سپس به روند برنامه باز میگردد و برای اینکه کل برنامه به اجرا ادامه دهد و همچنین این بخش از برنامه منتظر دریافت اطلاعات از سوی شبکه باشد نیاز است اقدامی انجام دهیم و با فعال کردن non blocking برای این بخش میتوان اطلاعات را بدون متوقف شدن برنامه دریافت کرد ولی نکته این است که در صورت عدم دریافت اطلاعات از شبکه بصورت خودکار مقدار های تصادفی به ما میدهد و بخش non blocking هم مشکلاتی دارد که نیاز است کد کتابخانه ویرایش شود برای مثال با زمانی که این بخش را مورد استفاده قرار دادیم با محدودیت ارسال رشته مواجه شدیم. درست است که این موارد به مقدار بافر مربوط است ولی این کتابخانه مدل ارسال اطلاعات را بصورت Packet استفاده میکند بنابر این برخی مشکلات میتواند از آن منشع کرفته شود. همچنین این کتابخانه از نظر Thread هم مواردی دارد که دسترسی توسعه‌دهنده را محدود میکند. بنابر این تصمیم گرفتیم که بخش شبکه پروژه را با استفاده کتابخانه ای دیگر و تخصصی تر توسعه دهیم.

کلاس `sf::RenderWindow` در کتابخانه SFML برای ایجاد پنجره‌ای برای رسم شیء‌ها و انجام رویدادهای مربوط به پنجره استفاده می‌شود. در ادامه، یک مثال برای استفاده از `sf::RenderWindow` و رسم یک شیء ساده را ارائه می‌دهیم:

تصویر 15-1 نمونه کد یک برنامه ساده با استفاده از کتابخانه SFML

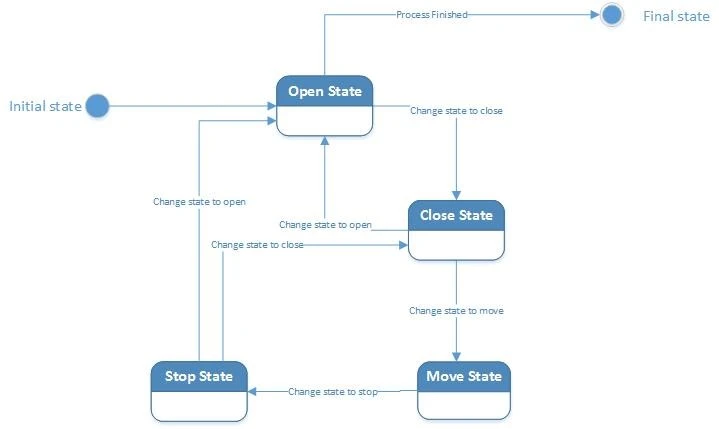
در این کد، یک شیء از کلاس `sf::RenderWindow` با سایز 800x600 پیکسل و عنوان "SFML Window" ایجاد می‌شود. سپس یک شیء مستطیل با ابعاد 100x100 پیکسل و رنگ قرمز ایجاد می‌شود و در موقعیت (200، 200) در پنجره قرار می‌گیرد.

سپس، در حلقه اصلی برنامه، رویدادهای مربوط به پنجره مانند بستن پنجره چک می‌شوند و در صورت بسته شدن پنجره، حلقه بسته می‌شود. در هر دوره از حلقه، پنجره پاکسازی شده و شیء مستطیل درون آن رسم می‌شود. در نهایت، پنجره نمایش داده می‌شود.

**الگوی طراحی (Design Pattern):**

راه حل‌هایی که برای مشکلات رایج در طراحی نرم‌افزار هستند که توسط مهندسین نرم‌افزار با تجربه تولید شده و میتوان برای حل مشکلات از آن‌ها در پروژه های مختلف استفاده کرد.

بنابر این مهم نیست با چه زبانی کد نویسی می‌کنید میتوانید از این الگو ها استفاده کنید. در این پروژه از الگوی طراحی بازی استفاده شده (Game Design Pattern). الگوی بازی یا برنامه تک صفحه‌ای شباهاتی با ماشین‌های وضعیت در طراحی کامپایلر دارد برای نمونه میتوان به تصویر زیر اشاره کرد که برنامه از یک وضعیت به یک وضعیت دیگر بستگی به ورودی و اطلاعات تغییر وضعیت صورت گیرد و در نتیجه ی آن تمامی برنامه بخش بخش شده و همینطور کد های آن تمیز و جداگانه نوشته شده‌اند و قابلیت استفاده مچدد در سایر پروژه ها و نیاز ها را دارد.



تصویر16 -1 الگوی طراحی وضعیت

**کتابخانه Boost:**

کتابخانه Boost.Asio یکی از اجزای اصلی کتابخانه Boost است که برای برنامه‌نویسی شبکه و ورود/خروج همروند طراحی شده است.

****

تصویر 17-1 کتابخانه بوست

Boost.Asio ابزارهایی را ارائه می‌دهد که برنامه‌نویسان را قادر می‌سازد برنامه‌های شبکه را پیاده‌سازی کنند، از جمله TCP، UDP، SSL، DNS و سایر پروتکل‌های شبکه.

در ادامه، یک کد مثال UDP Socket با استفاده از Boost.Asio را برای شما آورده‌ایم:

این کد یک سوکت UDP ایجاد می‌کند و یک پیام را به یک سرور با آدرس IP و پورت مشخص ارسال می‌کند. سپس منتظر پاسخ از سرور می‌ماند و پاسخ را دریافت و چاپ می‌کند.

لطفاً توجه داشته باشید که برای اجرای این کد، باید کتابخانه Boost را نصب و به پروژه خود اضافه کنید و همچنین در برخی از سیستم‌عامل‌ها، لازم است که کتابخانه Boost.Asio را به صورت جداگانه نصب کنید.



تصویر 18-1 کد نمونه ارسال داده به شبکه

**انگیزه:**

بعد از توضیحات بالا شاید سؤال برای شما سؤال پیش بیایید که چرا از یک فریم ورک یا متور بازی سازی مناسب برای ایجاد پروژه استفاده نکرده‌ایم. هدف از انتخاب این مسیر این بوده که یک نرم‌افزار تک صفحه‌ای با استفاده از Game Design Pattern ها تولید کنیم و همچنین با انجام این پروژه به یادگیری و کسب تجربه‌ عمیق‌تری نسبت به برنامه نویسی شی گرایی پیدا کنیم و در تمرین زبان برنامه نویسی ++C قدمی برداشته باشیم.

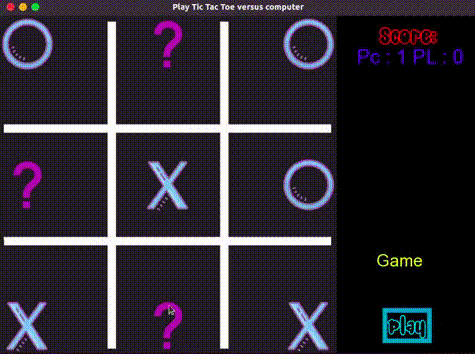
**فصل دوم**

**کارهای مرتبط و ابزار های مورد نیاز**

**کارهای مشابه که قبلاً صورت گرفته:**

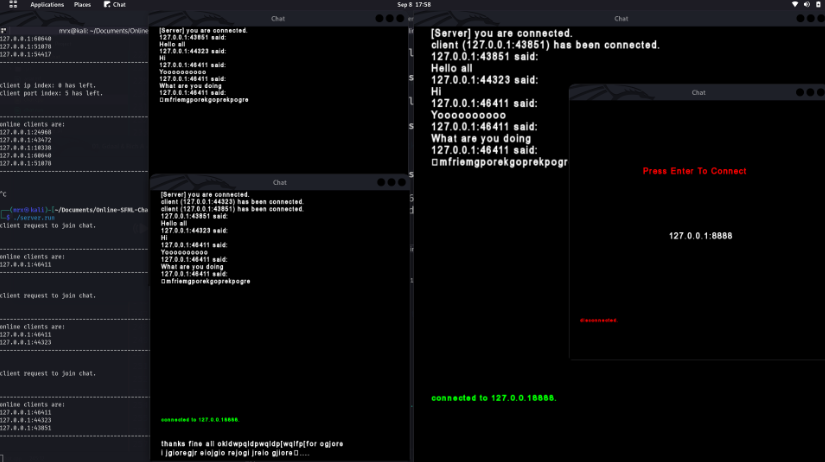
در گذشته چند پروژه ی کوچک با استفاده از این زبان و این کتابخانه توسعه داده شده است.

برای مثال میتوان به بازی دوز یا همان XO که کامپیوتر در مقابل کاربر بازی میکند:



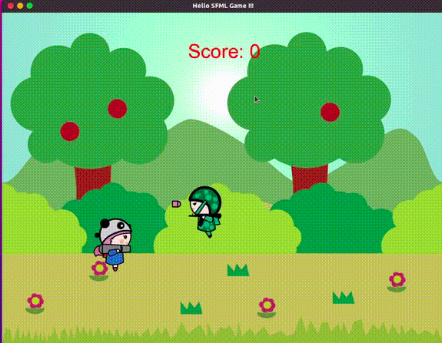
تصویر 1-2 بازی دوز

و به یک برنامه چت ساده تحت شبکه:



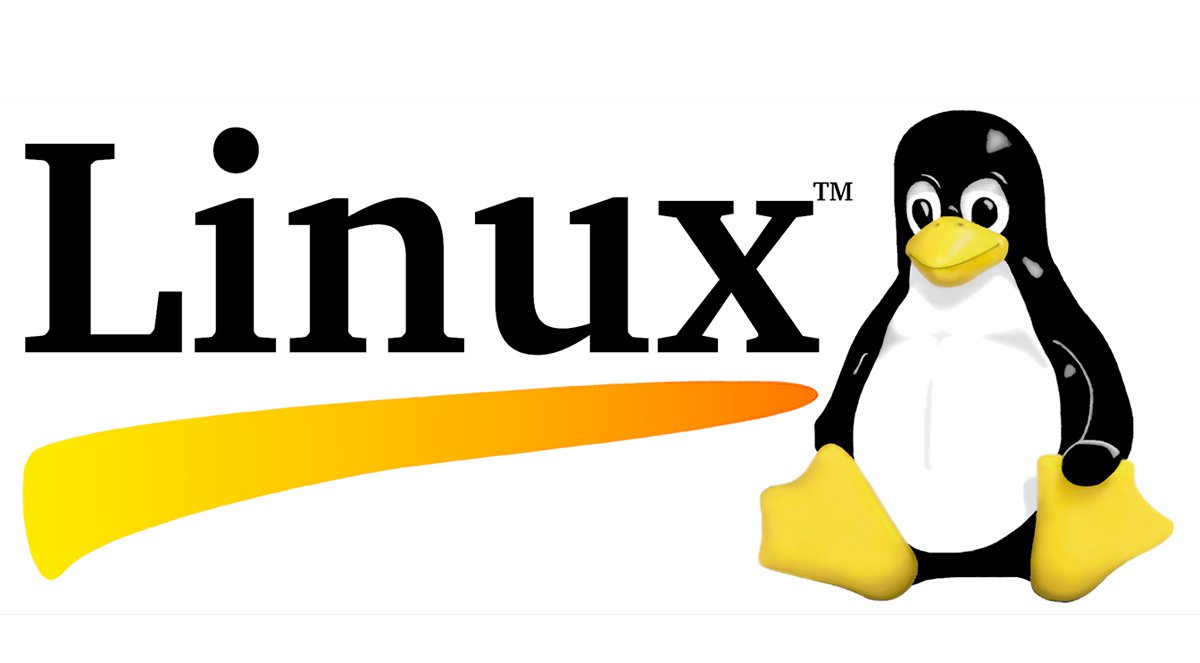
تصویر 2-2 برنامه چت آنلاین

و همچنین به یک بازی که با راهنمایی قدم به قدم کتاب (C++ Game Development By Example) اشاره داشت.



تصویر 3-2 بازی بازوکای کوچک

**راه اندازی و ابزار های مورد نیاز:**



تصویر 4-2 سیستم عامل لینوکس

**سیستم عامل:**

برای توسعه نرم‌افزار حرفه‌ای سیستم عامل های مختلفی وجود دارند. اما یکی از آن‌ها لینوکس است و تقریباً تمامی سیستم عامل ها قابلیت‌های یکسان و قابل مقایسه‌ای دارند و درصورت عدم وجود آن قابلیت میتوان از شبیه ساز هم استفاده کرد. این مورد بستگی به علاقه و علم فردی دارد و ما در توسعه این پروژه از سیستم عامل لینوکس استفاده کرده‌ایم به دلیل اینکه نیاز به کرک نرم‌افزار ها نیست و همچنین راحت شدن از آپدیت های و سرویس های ویندوز و استفاده از ابزار ها بدون نیاز به شبیه سازها.



تصویر 5-2 مدیریت نسخه گیتهاب

**مدیریت نسخه:**

برای اینکه پروژه بخش‌های مختلفی دارد و همینطور تعداد فایل‌ها کد زیاد میباشد و میخواهیم یک توسعه حرفه‌ای را تجربه کنیم. نیازمند یک برنامه مدیریت نسخه (Version Controller) هستیم ابزار های مدیریت نسخه نرم‌افزار قابلیت‌های زیر را به ما ارايه می‌کند:

* برای اینکه از پروژه یک نسخه پشتیبان در فضایی بجز فضای سیستم داشته باشیم به یک سرور نیازمند هستیم که هنگام توسعه آخرین تغییرات را به آن ارسال کرده و درصورت بروز مشکل به آن تغییرات دسترسی داشته باشیم.
* درصورتی که پروژه بصورت تیمی و چند نفره درحال توسعه باشد میتوان از این ابزار برای هماهنگی توسعه اعضای تیم استفاده کرد.
* ذخییره تغییرات با عنوان مناسب میتواند کمک بسیار زیادی در دسترسی به تغیرات آن کند.
* تگ و نسخه زدن به بخشی از کد.
* بازگشت به تغییرات قبل درصورت بروز مشکل.
* برسی تاریخچه تغییرات.

با استفاده از دستور زیر می‌توان ابزار گیت را نصب کرده:

sudo apt install git

سپس با چند دستور ابتدایی گیت آشنا خواهید شد:

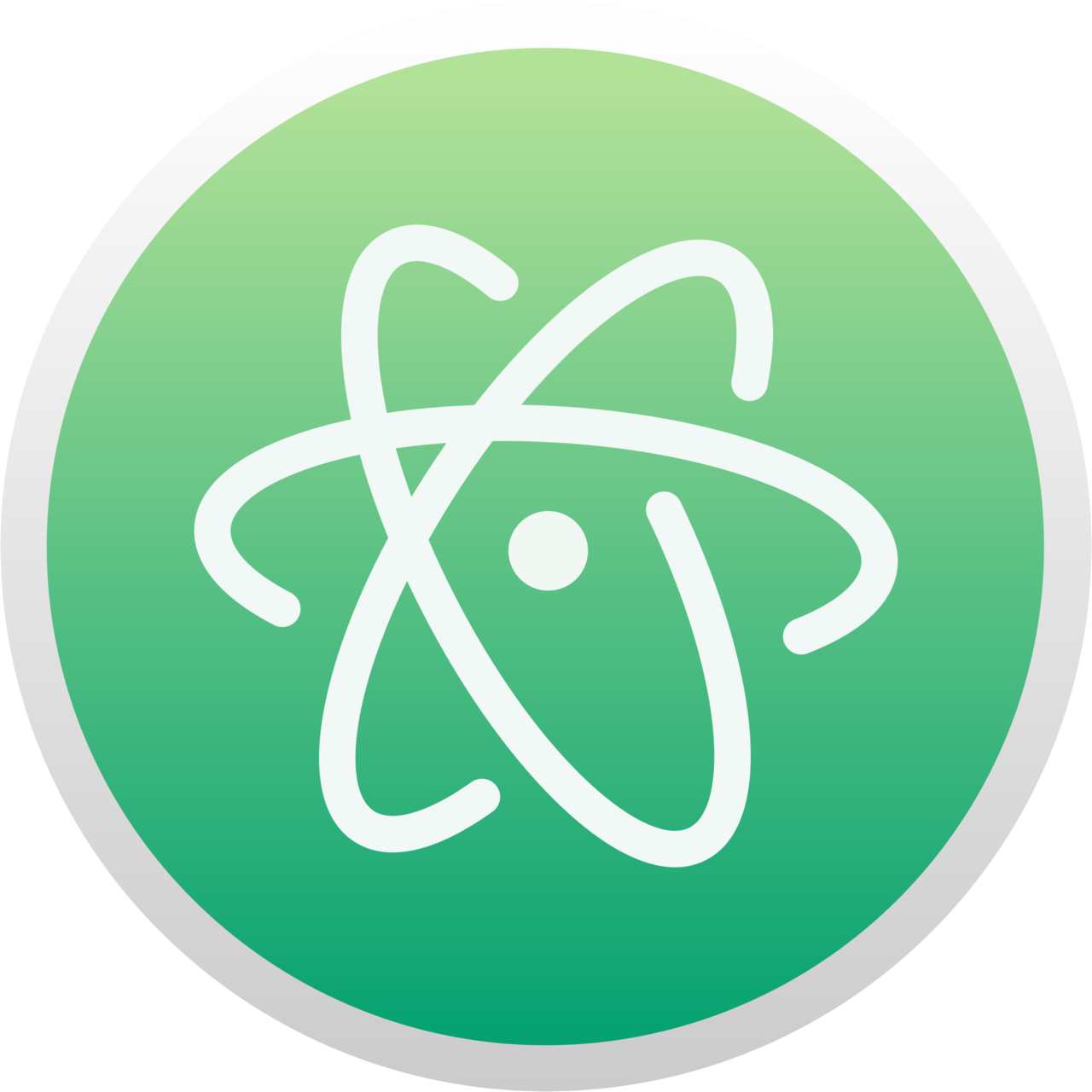
git add ./example.file

git pull

git push origin master

git tag “10.0.0” | git push –tags origin

و به یک حساب کاربری گیت هاب نیاز دارید و با وصل کردن آن به ابزار گیت می‌توانید کد ها را در مخزن ارسال و دریافت کنید.



تصویر 6-2 ویرایشگر Atom

**ویرایشگر کد و متن:**

ویرایشگر های زیادی هستن که قابلیت‌های مختلفی دارند ولی اکثر آن‌ها نیاز به کرک دارند و همچنین سلیقه فردی هم تأثیر زیادی دارد و ما در توسعه این پروژه از برنامه ی Atom استفاده کرده‌ایم. تقریباً میشه گفت این برنامه حرفه‌ای برای کد نویسی نیست مخصوصاً اگر زبان برنامه نویسی و محیط مربوط به وب نباشد. زیر ا نیاز به دیباگر و کنسول و خیلی از ابزار های دیگر است.

میتوانید با دستور زیر آن را در سیستم عامل لینوکس نصب کنید:

git snap install atom --classic



تصویر 7-2 کامپایلر

**نصب کامپایلر ++G و استفاده از آن:**

همانطور که گفته شد ویرایشگر متن ما قابلیت‌هایی را ندارد مثل کامپایلر.

برای نصب کامپایلر میتوانید با دستور زیر اقدام کنید:

sudo apt install g++-9

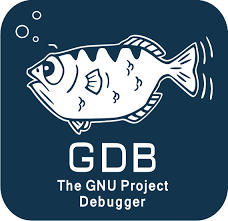
با دستور زیر میتوان از کد نوشته شد یک خروجی گرفت:

g++ -o hello.run hello.cpp

و برای اجرای خروجی:

hello.run/.

**دیباگر و نصب آن (Debugger):**

برخی موارد برنامه خروجی مورد نظر را نمی‌دهد و نیاز است خط به خط برنامه را

اجرا کرد و اشکال برنامه را پیدا کرد و یا ممکنه برنامه بسته بشود یا به اصطلاح

Crash کند. با استفاده از دیباگر می‌توان علت کرش کردن برنامه را دید البته برای

استفاده از این قابلیت نیاز است هنگام کامپایل گرفتن یک گزینه را فعال کرد.

تصویر 8-2 دیباگر

برخی از محیط های توسعه نرم‌افزار مجهز به دیباگر هستند ولی دیباگر های جدا از محیط توسعه هم میتوان یافت و استفاده کرد که ما GDB استفاده میکنیم.

برای نصب دیباگر در سیستم عامل لینوکس از دستور زیر استفاده میکنیم:

sudo apt-get install gdb

به نحوه کار با این دیباگر آشنا خواهیم شد. قبل از اینکه با دستورات و کارکردن با دیباگر آشنا شوید نیاز است بدانید که برای استفاده راحتتر و دقیق‌تر نیاز است هنگام کامپایل گرفتن از کد باید از -g استفاده کرد. این کار بخشی از کد خروجی را باز میگذارد تا دیباگر بتواند دسترسی بیشتری داشته باشد. برای اجرای برنامه توسط آن می‌توان با استفاده از دستور زیر اقدام کرد:

gdb ./hello.run

برای اجرای برنامه:

Run <

برای خروج از محیط دیباگر:

quit <

پس از وارد کردن دستور زیر به محیط دیباگر وارد می‌شوید برای اجرای برنامه می‌توانید عبارت run را وارد کرده و برای خروج میتوانید از quit استفاده کنید. برای توقف کردن برنامه وقتی که به خطی از کد یا تابعی که مورد نظر داریم رسید می‌توان از دستور زیر استفاده کرد. برای مثال وقتی برنامه به اجرای خط ۲۲ فایل مورد نظر رسید از اجرا دست برمی‌دارد و منتظر دستورات ما می‌شود.

break hello.cpp:22

برای اجرای یک خط بعدی از برنامه این دستور را وارد می‌کنیم:

next <

برنامه را به اجرا دربیاور تا وقتی که دوباره به آن نقطه مورد نظر برسد:

continue <

چاپ مقدار یک متغیر در دسترس (یعنی در خط های قبل که اجرا شده بودند این متغییر وجود داشته).

print myVariable

در این دیباگر دستورات و قابلیت‌های بیشتر هم وجود دارد که می‌توانید با تحقیق درباره آن اطلاعات بیشتری کسب کنید.

**نصب کتابخانه گرافیکی (SFML):**

برای توسعه نیاز به نصب کتابخانه گرافیکی است

و برای نصب آن می‌توان از دستور زیر استفاده کرد:

sudo apt-get install libsfml-dev

تصویر 9-2 کتابخانه گرافیکی

**نصب کتابخانه Boost:**

برای توسعه نیاز به نصب این کتابخانه است

و برای نصب آن می‌توان از دستور زیر استفاده کرد:



sudo apt-get install libboost-all-dev

تصویر 10-2 کتابخانه Boost

**نصب کتابخانه پایگاه داده Sqlite:**

برای توسعه نیاز به نصب این کتابخانه است

و برای نصب آن می‌توان از دستور زیر استفاده کرد:

sudo apt-get install sqlite3

تصویر 11-2 کتابخانه Sqlite

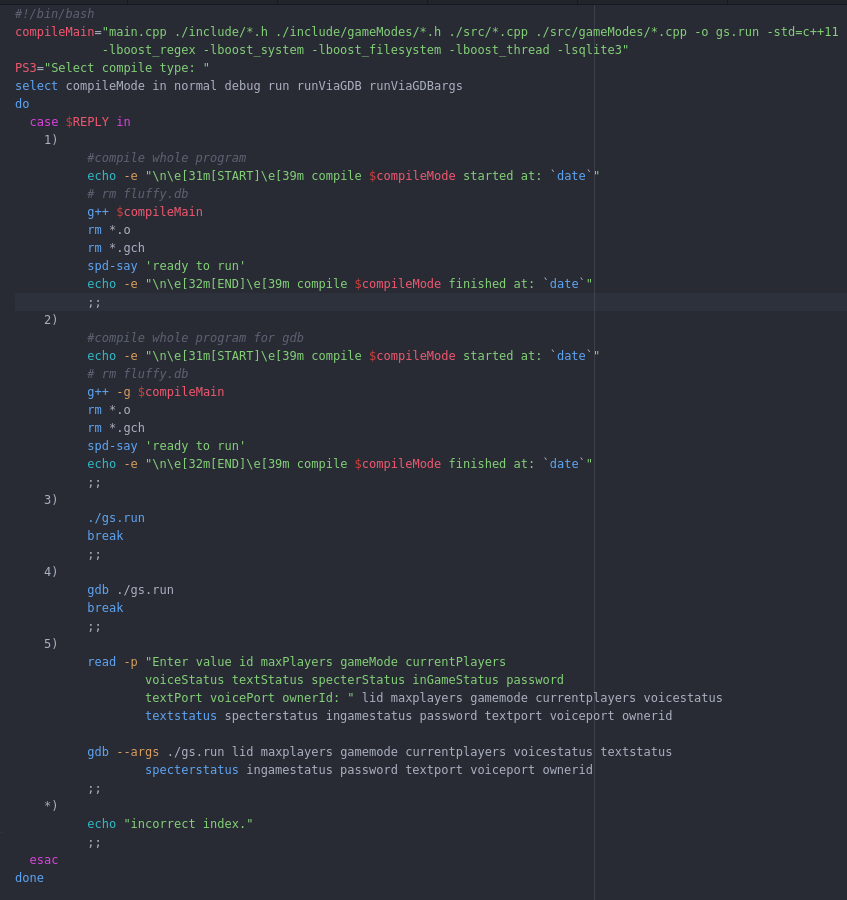
**فصل سوم**

**جزئیات روش و کارهای انجام شده**

**چگونگی خروجی گرفتن پروژه:**

با استفاده از bash script که نوشته‌ایم تمامی فایل‌ها و پوشه های آن بخش را در کامپایل یا اجرا قرار میدهد. پنج حالت در اختیار داریم که با وارد کردن مقادیر به کامپایلر و دیباگر دستور میدهد و برنامه اجرا کرده یا خروجی را آماده می‌کند و پس از کامپایل یک پیام مشخص را که می‌گوید‌ “آماده اجرا است” البته به زبان انگلیسی. با اجرا کردن این اسکریپت حالت‌های زیر را داریم و با وارد کردن عدد های زیر به حال مورد نظر وارد می‌شویم:

1. کامپایل ساده: فقط برنامه را کامپایل می‌کند.
2. کامپایل برای دیباگر: با فعال کردن پرچم یا flag دیباگ کامپایل می‌کند.
3. اجرای برنامه.
4. اجرای برنامه در دیباگر.
5. اطلاعاتی را می‌گیرد و سپس برنامه را اجرا می‌کند ولی با ارسال آن اطلاعات به عنوان ورودی یا Argument.



تصویر 0-3 کد اسکریپت کامپایل برنامه

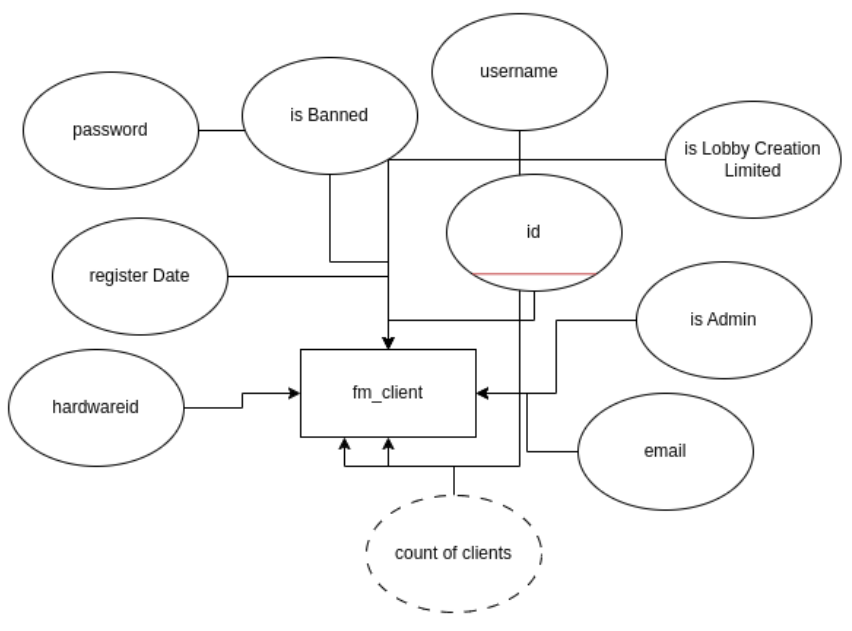
**پایگاه داده و نحوه ذخیره اطلاعات:**

این پایگاه داده شامل چهار جدول می‌باشد که توسط برنامه Main Server و Game Server ثبت و ویرایش و خوانده می‌شوند. بنابر این درصورتی که نیاز است پایگاه داده به API منتقل شود این موارد در نظر گرفته و اعمال شود.

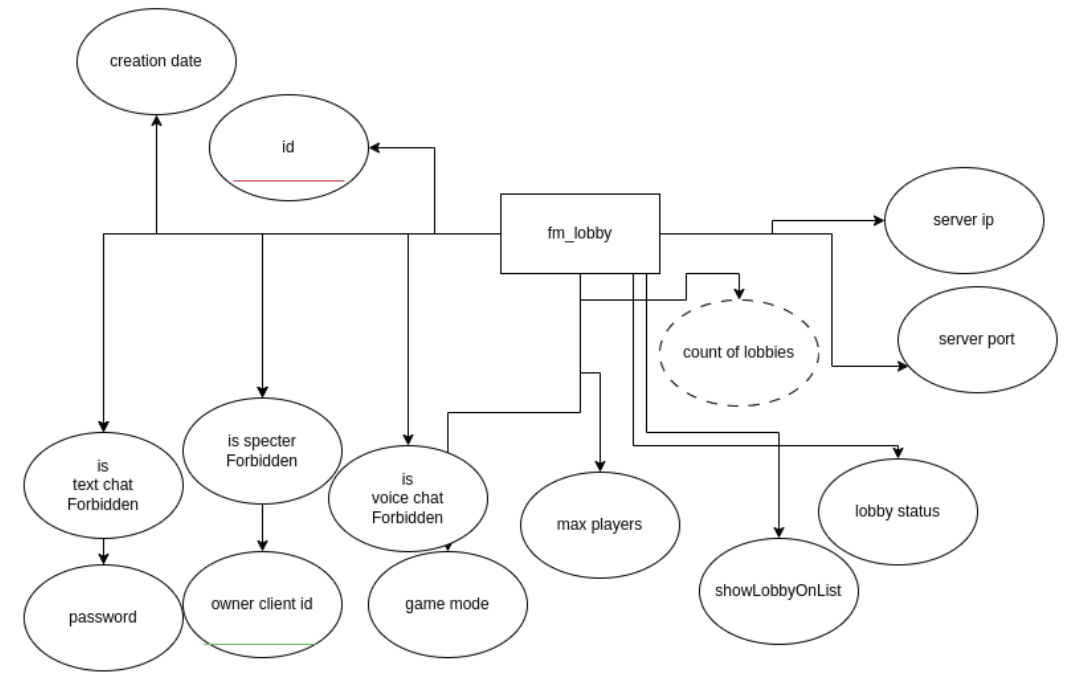
جدول ۱-۳ جدول های پایگاه داده

|  |  |  |
| --- | --- | --- |
| **نام جدول** | **کاربرد** | **کلید خارجی از** |
| fm\_client | ذخیره اطلاعات حساب‌های کاربری | - |
| fm\_lobby | ذخیره اطلاعات اتاق‌های بازی | fm\_lobby  شناسه مدیر اتاق |
| fm\_client\_in\_lobby | ذخیره شناسه کاربر و شناسه اتاق بازی که نشان دهنده ی این است که آن کاربر به آن اتاق متصل شده است | fm\_lobby  شناسه اتاق بازی  fm\_client  شناسه کاربر |
| fm\_client\_login | ذخیره کاربر هایی که وارد شده‌اند همراه با زمان ورود و شناسه منحصر به فرد (identity) | fm\_client  شناسه کاربر |

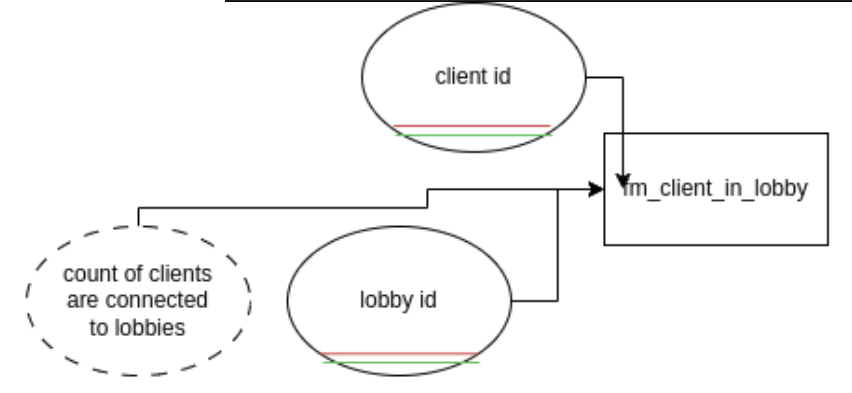
همچنین جزئیات جداول در تصاویر زیر آمده است همجنین می‌توانید در پروژه در پوشه docs به پوشه database رفته و اطلاعات کامل تری از پایگاه داده کسب کنید.

****

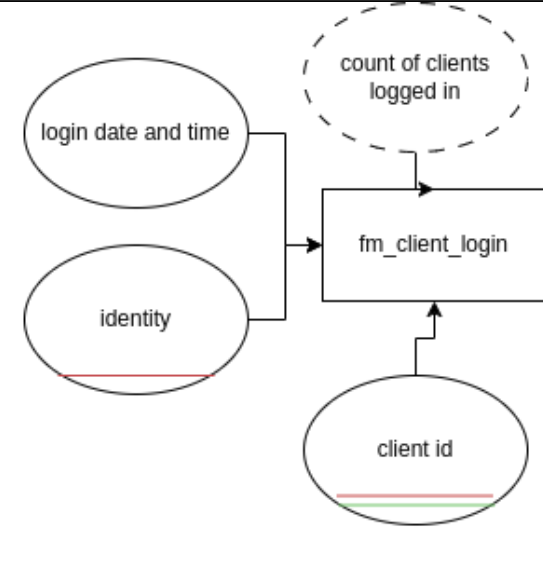
تصویر 1-3 پایگاه داده جدول حساب کاربر ها



تصویر 2-3 پایگاه داده جدول اتاق‌های بازی



تصویر 3-3 پایگاه داده جدول کاربر به کدام اتاق متصل شده

****

تصویر 4-3 پایگاه داده جدول کاربر های وارد شده

**برنامه Main Server:**

این برنامه برای اجرا در سمت سرور توسعه داده شده است. در یک سرور قرار می‌گیرد و اطلاعات را با پایگاه داده‌ای که در کنار خود می‌سازد قرار میدهد.



تصویر 5-3 نحوه کار کرد کلی برنامه Main Server

(درصورتی که تعداد کاربران زیاد شد این امکان هست که این برنامه را در چند سرور قرار داده و پایگاه داده را در یک سرور مرکزی قرار بدهید و با این کار می‌توان از شلوغی پهنای باند سرور اصلی جلوگیری کرد و تجربه بهتری برای کاربران رقم زد) پایه این قابلیت برای آینده قرار داده شده و توسط قابلیتی که در بخش Main Client خواهیم دید این امر به سادگی امکان‌پذیر خواهد بود. تنها نیاز است فایل پایگاه داده را در یک سرور مرکزی یا API قرار داد و به صورتی اطلاعاتی که میخوایم بخونیم یا بنویسیم در پایگاه داده را به آن درخواست بدهیم. بنابر این می‌توان این برنامه را در چندین سرور قرار داده به عنوان سرور های پشتیبان درصورتی که یک سرور مشکلاتی داشت کاربران با کلیک بر دکمه تلاش مجدد به سرور های دیگر (پشتیبان) متصل خواهند شد.

این برنامه وظایف زیر را بر عهده دارد:

1. ثبت حساب کاربری.
2. ورود به حساب کاربری.
3. ساخت اتاق بازی.
4. ارسال مشخصات اتاق بازی به کاربر (توسط شناسه ای که کاربر ارسال کرده است).
5. ارسال لیست اتاق بازی به کاربر.
6. ورود به حساب کاربری بدون نیاز به مجدد وارد کردن مشخصات حساب کاربری (درصورتی که زمان انقضای ورود تمام نشده باشد).

این نکته که این برنامه در اولین فاز توسعه یافته بنابر این معماری و ساختار این بخش زیاد جالب و دل‌نشین نیست ولی بخش‌های آینده بهتر و بهتر خواهند بود.

از آنجایی که فایل‌های زیادی در این بخش قرار دارد. حال با فایل‌ها و کد های درون این بخش آشنا خواهیم شد و توضیحاتی در مورد آن‌ها خواهیم داد.

جدول ۲-۳ معرفی فایل‌های Main Server

|  |  |  |
| --- | --- | --- |
| **نام فایل** | **کاربرد** | **هدف** |
| Config.h | نگهداری اطلاعات و مقادیر. | جداسازی و مرتب سازی مقادیر حیاتی پروژه و سرعت در تغییر و توسعه پروژه. برای مثال: درصورتی که این مقادیر در کد و فایل‌های مختلف نوشته می‌شد پیدا کردن این مقادیر سخت خواهد بود و خوانایی پروژه و کد پایین می‌آمد. |
| Database.h | کار با پایگاه داده محلی (همچنین query های ثبت و … اطلاعات مختلف در قالب توابع در این بخش قرار دارند). | نوشتن و یا خواندن اطلاعات به پایگاه داده توسط توابع موجود در آن. |
| DataSecurity.h | تابع رمز گزاری/گشایی اطلاعات.  تابع برسی و حذف کد های مخرب sql دریافت شده.  تابع تولید شناسه یکتا (identity) که جهت احراز هویت کاربر استفاده شده.  تابع برسی سختی و آسانی کلمه عبور وارد شده توسط کاربر. | امن نگه داشتن اطلاعات و برسی اطلاعات دریافتی توسط سرور و موارد امنیتی. |
| DataType.h | نگهداری و جمع آوری داده‌ها و نوع های پر تکرار و مورد استفاده توسط فایل‌های متفاوت. | در پروژه نوع ها (Type) های داده‌ای زیاد خواهند بود و درصورتی که هر بخش/کلاس نوع خود را کنار خود بنویسد در این امر خوانایی کد را کمتر خواهد کرد (برای مثال ویرایش سخت خواهد بود) همچنین در پروژه بخش‌های مختلف نوع ها را نیاز دارند با این روش جلوگیری خواهیم کرد از include کل کلاس یا فایل به فایل/تابع مورد نظر که این امر از مشکلات include های زیاد (مشکل circular dependencies را کمتر خواهد کرد). |
| ProcessData.h | پردازش و پاسخ دهی به درخواست/اطلاعات دریافتی از سوی کاربران. | برسی وجود ارتباط (connection) های و همچنین متصل کردن و جدا کردن یا پارس کردن اطلاعات دریافتی (seperate) و صدا زدن توابع مورد نیاز از بخش database برای اعمال یا خواندن اطلاعات از سوی پایگاه داده. |
| Server-request-response-list.h | نگهداری کد های درخواست و پاسخ با نام خوانا. | در ابتدای پیام دریافتی یا ارسالی از یا به سوی کاربر یک کد سه رقمی قرار دارد و برای تمیز بودن و توسعه راحتتر کد از نام های واضح و عدد ها استفاده شده و این فایل وظیفه نگهداری آن را دارد. این فایل با فایل مربوطه با همین نام در بخش Main Client باید یکسان باشند تا در ارتباط کاربر و سرور کد های دستوری یکدیگر را متوجه بشوند. |
| Main.cpp | تابع دریافت اطلاعات همراه با Thread آن  شی پایگاه داده  لیست اطلاعات دریافت شده  صدا زدن تابع process از کلاس ProcessData | اجرای همزمان توابع و متدها در کنار یکدیگر بصورتی که با لیست موجود در تعامل باشند.  (به اشتراک گذاشتن یا پاس دادن منابعی همچون لیست دریافتی ها و پایگاه داده به تابع receiveData و شئ process) |

حال که با فایل‌ها و کاربرد و اهداف آن‌ها آشنایی پیدا کردیم اطلاعات بیشتری از این نرم‌افزار خواهیم داد.

سناریو و روند اجرای و کارکرد این برنامه بدین صورت است که این برنامه اجرا می‌شود. ابتدا وجود داشتن فایل پایگاه داده را برسی می‌کند درصورتی که فایل پایگاه داده‌ای در کنار آن موجود نباشد. ابتدا تلاش میکند و فایل پایگاه داده را می‌سازد و سپس جدول های تعریف شده را در آن تولید می‌کند.

سپس یک لیست را برای نگهداشتن اطلاعات دریافتی تولید می‌کند. سپس سوکتی از نوع udp تولید می‌کند و پورت را از فایل config.h می‌خواند و سپس آن را bind می‌کند. یک نخ (thread)‌ تولید می‌کند و تابع receiveData را به عنوان تابع اجرایی و همچنین سوکت و لیست اطلاعات دریافت شده را بصورت رفرنس به آن پاس می‌دهد. سدس شئ از نوع ProcessData تولید می‌کند و در یک حلقه بی نهایت متد process آن شئ را صدا می‌زند و این کار باعث می‌شود تا برنامه هم‌زمان تابع دریافت اطلاعات و پردازش اطلاعات را اجرا کند.

تابع دریافت اطلاعات از سوکت اطلاعات را می‌خواند و درصورتی که شرط های مورد نظر (برسی اندازه اطلاعات دریافتی) صحت یافت اطلاعات را در قالب نوع داده‌ای SocketDataQueue قرار می‌دهد و آن را در لیست اطلاعات دریافتی اضاف می‌کند.

مشخصات نوع SocketDataQueue در جدول زیر آمده است:

جدول ۳-۳ نوع داده‌ای SocketDataQueue

|  |  |  |
| --- | --- | --- |
| **نوع داده** | **مقدار** | **کاربرد** |
| int | code | کد درخواست عددی بین ۱۰۰ تا ۹۹۹ که در یک فایل این کد ها تعریف شده‌اند. |
| boost::asio::ip::address | ip | آدرس آی پی کاربر درخواست دهنده |
| unsigned short | port | آدرس پورت کاربر درخواست دهنده |
| std::string | data | اطلاعات دریافت شده (این اطلاعات بصورت پشت هم هست و جدا سازی نشده است و اطلاعات با استفاده از جداکننده ها (Delimiters) جدا خواهند شد. |

همچنین هم‌زمان آن متد process درحال برسی اندازه لیست اطلاعات دریافتی است درصورتی که اندازه آن بیشتر از صفر شد آن اطلاعات را در برسی قرار می‌دهد. اطلاعات دریافتی را رمز گشایی می‌کند سپس امنیت اطلاعات را توسط تابع isSQLCodeIncluded برسی می‌کند درصورت وجود کد مخرب sql تابع removeSQLCodeFromData اجرا خواهد شد که مقدار های اضافی را از اطلاعات حذف کند و سپس با استفاده از یک switch کد درخواست را مورد برسی قرار می‌دهد.

کد و توضیحات مربوط به درخواست و پاسخ در جدول ۵-۳ آمده است. سپس بعد از پردازش (جدا سازی مقادیر با استفاده از تابع dataSeparator) و اعمال تغییرات در پایگاه داده در صورت نیاز با استفاده از تابع sendData و منتقل کردن مقادیری همچون کد پاسخ و اطلاعات مورد نظر جهت ارسال به کاربر اقدام می‌کنیم.

نکته: برای کوتاه شدن نوع در جدول با عدد نشانه گذاری می‌شوند:

جدول ۴-۳ نشانه گذاری و توضیحات کد های درخواست و پاسخ

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **شناسه/کد** | **نوع** | **فرستنده** | **گیرنده** | **محدوده کد** |
| ۱ | درخواست | کاربر | سرور | ۱۰۰ تا ۱۹۹ |
| ۲ | پاسخ موفق | سرور | کاربر | ۲۰۰ تا ۳۹۹ |
| ۳ | پاسخ ناموفق | سرور | کاربر | ۴۰۰ تا ۴۹۹ |
| ۴ | پاسخ ناموفق  (به دلیل مشکل داخلی سرور) | سرور | کاربر | ۵۰۰ تا ۵۹۹ |
| - | - | - | - | ۶۰۰ تا ۹۹۹ |

برای زیبایی نام و کوتاه شدن آن در جدول زیر بخش‌هایی از نام حذف شده‌اند. برای مثال عبارت MS به معنای MainServer و MC به معنای MainClient می‌باشد که از نام حذف شده‌اند همچنین خط‌های زیر “ \_ ” هم حذف شده اند.

شناسه سخت‌افزار و identity که در بخش اطلاعات قرار گرفته است توسط برنامه ارسال انتخاب و پر می‌شود و نیازی نیست کاربر اقدامی برای آن انجام بدهد.

کد های درخواست و پاسخ‌ها با محدوده متفاوت از یکدیگر مقدار دهی شده است. برای اینکه به سادگی با دیدن محدوده بفهمیم مشکل از کجا است. برای مثال اگر کد در محدوده ۵۰۰ بود میتوان گفت این پاسخ است و مشکل داخلی در سرور هنگام پردازش درخواست ما رخ داده است.

جدول ۵-۳ کد های درخواست و پاسخ‌ها به همراه توضیحات

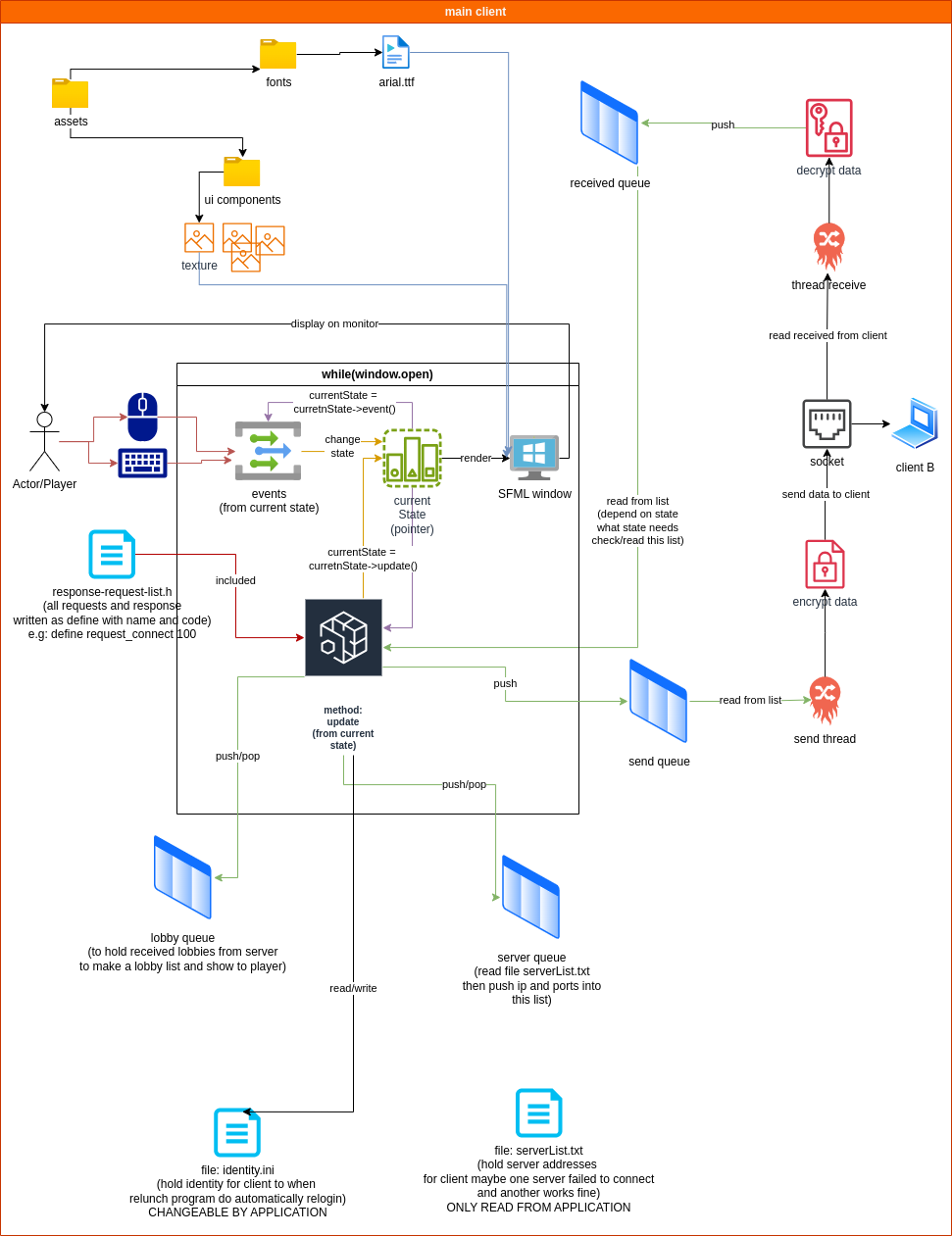
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **نوع** | **کد** | **نام** | **توضیح** | **اطلاعات دریافتی/ارسالی** |
| 1 | 100 | REQUEST CONNECT | درخواست متصل شدن به سرور | - |
| 1 | 101 | REQUEST DISCONNECT | درخواست قطع اتصال از سرور | - |
| 1 | 102 | REQUEST  LOGIN | درخواست ورود به حساب کاربری | نام کاربری  کلمه عبور  شناسه سخت‌افزار |
| 1 | 103 | REQUEST REGISTER | درخواست ثبت حساب کاربری جدید | ایمیل  نام کاربری  کلمه عبور  شناسه سخت افزار |
| 1 | 104 | REQUEST  CREATE  LOBBY | درخواست ساخت اتاق بازی | Identity  شناسه مدل بازی  حداکثر بازیکن های اتاق  وضعیت ارتباط صوتی  وضعیت ارتباط متنی  وضعیت وارد شدن تماشاگر  کلمه عبور اتاق |
| 1 | 105 | REQUEST GET LOBBY LIST | درخواست دریافت لیست اتاق‌های بازی | - |
| 1 | 106 | REQUEST  GET LOBBY INFO | درخواست دریافت اطلاعات اتاق بازی توسط شناسه آن | شناسه اتاق بازی |
| 1 | 107 | REQUEST  LATEST VERSION | درخواست دریافت ورژن جاری سرور | - |
| 1 | 108 | REQUEST  RELOGIN | درخواست ورود به حساب کاربری بدون وارد کردن نام کاربری و کلمه عبور | Identity  شناسه سخت افزار |
|  | | | | |
| 2 | 200 | RESPONSE CONNECTION ACCEPTED | موفقیت در اتصال به سرور | - |
| 2 | 201 | RESPONSE DISCONNECTED | موفقیت در قطع اتصال از سرور | - |
| 2 | 202 | RESPONSE SUCCESS LOGIN | ورود با موفقیت انجام شده است | identity |
| 2 | 203 | RESPONSE SUCCESS REGISTER | ثبت نام با موفقیت انجام شده است. | identity |
| 2 | 204 | RESPONSE SUCCESS LOBBY CREATED | ساخت اتاق بازی با موفقیت انجام شده است. | آدرس سرور : آدرس پورت  سروری که ساخته شده |
| 2 | 205 | RESPONSE SUCCESS GET LOBBY LIST | درخواست دریافت لیست اتاق‌های بازی با موفقیت انجام شده است. | اطلاعات ۱۰ اتاق اول را بصورت پشت سر هم و با کارکتر های جدا کننده تفکیک شده‌اند.  (جدا کننده در فایل config.h با نام MS\_DATA\_DELIMITER)  (این تعداد در فایل config.h با نام MSGET\_LOBBY\_LIST\_COUNT\_OF\_RESULTS ذخیره شده و قابل ویرایش است.  اطلاعات به ترتیب زیر می باشد:  به ترتیب:  وضعیت کلمه عبور (۰ یا ۱)  وضعیت ارتباط صوتی (۰ یا ۱)  وضعیت ارتباط متنی (۰ یا ۱)  وضعیت ورود تماشاگر (۰ یا ۱)  وضعیت اتاق بازی بدین معنی که در بازی هستند یا خیر (۰ یا ۱)  وضعیت نمایش در لیست اتاق بازی‌ها (۰ یا ۱)  شناسه اتاق  تعداد حداکثر کاربر هایی که می‌توانند وارد شوند.  تعداد کاربر های حاری در اتاق  مدل بازی  آدرس سرور (آدرس و پورت یک‌جا با عبارت : جدا شده اند) |
| 2 | 206 | RESPONSE SUCCESS GET LOBBY INFO | با دریافت اطلاعات اتاق توسط شناسه اتاق با موفقیت انجام شده است. | اطلاعات آن اتاق را بصورت پشت سر هم با استفاده از تفکیک کننده جدا شده.  ترتیب اطلاعات مثل  RESPONSE SUCCESS GET LOBBY LIST  می‌باشد |
| 2 | 207 | RESPONSE SUCCESS LATEST VERSION | پاسخ موفقیت دریافت عدد آخرین نسخه برنامه | ورژن سرور را از config.h  با نام : MS\_VERSION |
|  | | | | |
| 3 | 400 | ERROR CONNECTION EXISTS | خطای اتصالی با این آدرس و پورت وجود دارد. | - |
| 3 | 401 | ERROR CONNECTION REJECTED USING VPN | خطای اتصال رد شده است به دلیل استفاده از فیلترشکن | - |
| 3 | 402 | ERROR CONNECTION REJECTED | خطای اتصال رد شده است. | - |
| 3 | 403 | ERROR CONNECTION NOT EXISTS | خطای اتصالی با این آدرس و پورت یافت نشد. | - |
| 3 | 404 | ERROR FAILED TO LOGIN INCORRECT | خطا اطلاعات ورود به حساب کاربری درست نیست. | - |
| 3 | 405 | ERROR FAILED TO LOGIN NOT EXISTS | خطا حساب کاربری وجود ندارد | - |
| 3 | 406 | ERROR\_FAILED\_TO\_LOGIN\_BANNED | خطا حساب کاربری ممنوع شده است. | - |
| 3 | 407 | ERROR\_FAILED\_TO\_LOGIN\_BAD\_DATA\_SYNTAX | خطا ساختار درخواست ورود به حساب کاربری مشکل دارد. | - |
| 3 | 408 | ERROR\_FAILED\_TO\_REGISTER\_USERNAME\_EXISTS | خطا در ثبت نام حساب کاربری: نام کاربری وجود دارد. | - |
| 3 | 409 | ERROR\_FAILED\_TO\_REGISTER\_EMIAL\_EXISTS | خطا در ثبت نام حساب کاربری: پست الکترونیکی وارد شده قبلاً ثبت شده. | - |
| 3 | 410 | ERROR\_FAILED\_TO\_REGISTER\_EASY\_PASSWORD | خطا در ثبت نام حساب کاربری: کلمه عبور وارد شده آسان است | - |
| 3 | 411 | MS\_ERROR\_FAILED\_TO\_REGISTER\_BAD\_DATA\_SYNTAX | خطا ساختار درخواست ساخت حساب کاربری مشکل دارد. | - |
| 3 | 412 | MS\_ERROR\_FAILED\_TO\_LOBBY\_CREATION\_INVALID\_IDENTITY | خطا در ساخت اتاق بازی:  (مقدار identity) هویت نامعتبر است. | - |
| 3 | 413 | MS\_ERROR\_FAILED\_TO\_LOBBY\_CREATION\_CANT\_OWN\_TWO\_LOBBY | خطا در ساخت اتاق بازی:  امکان ساخت دو اتاق توسط یک نفر نیست. | - |
| 3 | 414 | MS\_ERROR\_FAILED\_TO\_LOBBY\_CREATION\_FORBIDDE  N\_FOR\_YOU | خطا در ساخت اتاق بازی:  ساخت اتاق بازی برای این حساب محدود شده است. | - |
| 3 | 415 | MS\_ERROR\_FAILED\_TO\_LOBBY\_CREATION\_BAD\_DATA\_SYNTAX | خطا در ساخت اتاق بازی:  ساختار درخواست نامعتبر است. | - |
| 3 | 416 | MS\_ERROR\_FAILED\_TO\_GET\_LOBBY\_LIST\_NO\_LOBBY\_AVAIABLE | خطا: اتاق بازی وجود ندارد که به عنوان لیست اتاق‌ها ارسال شود. | - |
| 3 | 417 | MS\_ERROR\_FAILED\_TO\_GET\_LOBBY\_INFO\_LOBBY\_NOT\_FOUND | خطا در دریافت اطلاعات اتاق بازی:‌ اتاقی با چنین شناسه ای یافت نشد. | - |
| 3 | 418 | MS\_ERROR\_FAILED\_TO\_GET\_LOBBY\_INFO\_BAD\_DATA\_SYNTAX | خطا در دریافت اطلاعات اتاق بازی:‌ ساختار درخواست نامعتبر است. | - |
| 3 | 419 | MS\_ERROR\_FAILED\_TO\_RELOGIN\_IDENTITY\_INVALID\_OR\_NOT\_EXISTS | خطا در ورود مجدد به حساب کاربری: (مقدار identity) هویت نامعتبر است. | - |
| 3 | 420 | MS\_ERROR\_FAILED\_TO\_RELOGIN\_IDENTITY\_EXPIRED | خطا در ورود مجدد به حساب کاربری:زمان ورود مجدد منقضی شده است | - |
|  | | | | |
| 4 | 500 | MS\_ERROR\_FAILED\_TO\_INSERT\_CLIENT\_IDENTITY | خطا داخلی سرور:هنگام ثبت هویت کاربر در پایگاه داده | - |
| 4 | 501 | MS\_ERROR\_FAILED\_TO\_REGISTER\_CLIENT | خطا داخلی سرور:هنگام ثبت حساب کاربری جدید | - |
| 4 | 502 | MS\_ERROR\_FAILED\_TO\_CREATE\_LOBBY | خطا داخلی سرور:هنگام ساخت اتاق بازی | - |
| 4 | 503 | MS\_ERROR\_FAILED\_TO\_GET\_LOBBY\_LIST | خطا داخلی سرور:هنگام دریافت لیست اتاق‌های بازی | - |
| 4 | 504 | MS\_ERROR\_FAILED\_TO\_GET\_LOBBY\_INFO | خطا داخلی سرور:هنگام دریافت اطلاعات اتاق بازی توسط شناسه اتاق | - |
| 4 | 506 | MS\_ERROR\_FAILED\_TO\_RELOGIN | خطا داخلی سرور:هنگام ورود مجدد به حساب کاربری | - |

**برنامه Main Client:**

این برنامه همان اجرا کننده یا Launcher هست که در سمت کاربر قرار می‌گیرد و با استفاده از Game Design

Pattern وضعیت ماشین رابط کاربری تک صفحه‌ای را پیاده‌سازی می‌کند.

دو لیست و نخ Thread در این برنامه وجود دارد که وظیفه نگهداری و ارسال و دریافت اطلاعات از سوکت را بر عهده دارند. سپس متد های مختلفی از اشاره گر وضعیت صدا زده می‌شوند و رابط کاربری مربوطه را به نمایش می‌گذارند و رویداد ها (event) را پشتیبانی خواهند کرد. در ادامه توضیحاتی درمورد بخش‌های ذکر شده داده خواهد شد.



تصویر 6-3 نحوه کار کرد کلی Main Client

کاربر با استفاده از برنامه کار های زیر را می‌تواند انجام دهد:

* ثبت نام حساب کاربری جدید.
* ورود به حساب کاربری.
* ورود به حساب کاربری بدون نیاز به مجدد وارد کردن مشخصات حساب کاربری (درصورتی که زمان انقضای ورود تمام نشده باشد).
* خروج از برنامه.
* دیدن لیست اتاق‌های بازی.
* تازه کردن لیست اتاق‌های بازی.
* خروج از حساب کاربری.
* ساخت اتاق بازی.
* ورود به اتاق بازی.
* دیدن اطلاعات اتاق بازی.
* گرفتن اطلاعات اتاق بازی با وارد کردن شناسه اتاق مورد نظر.

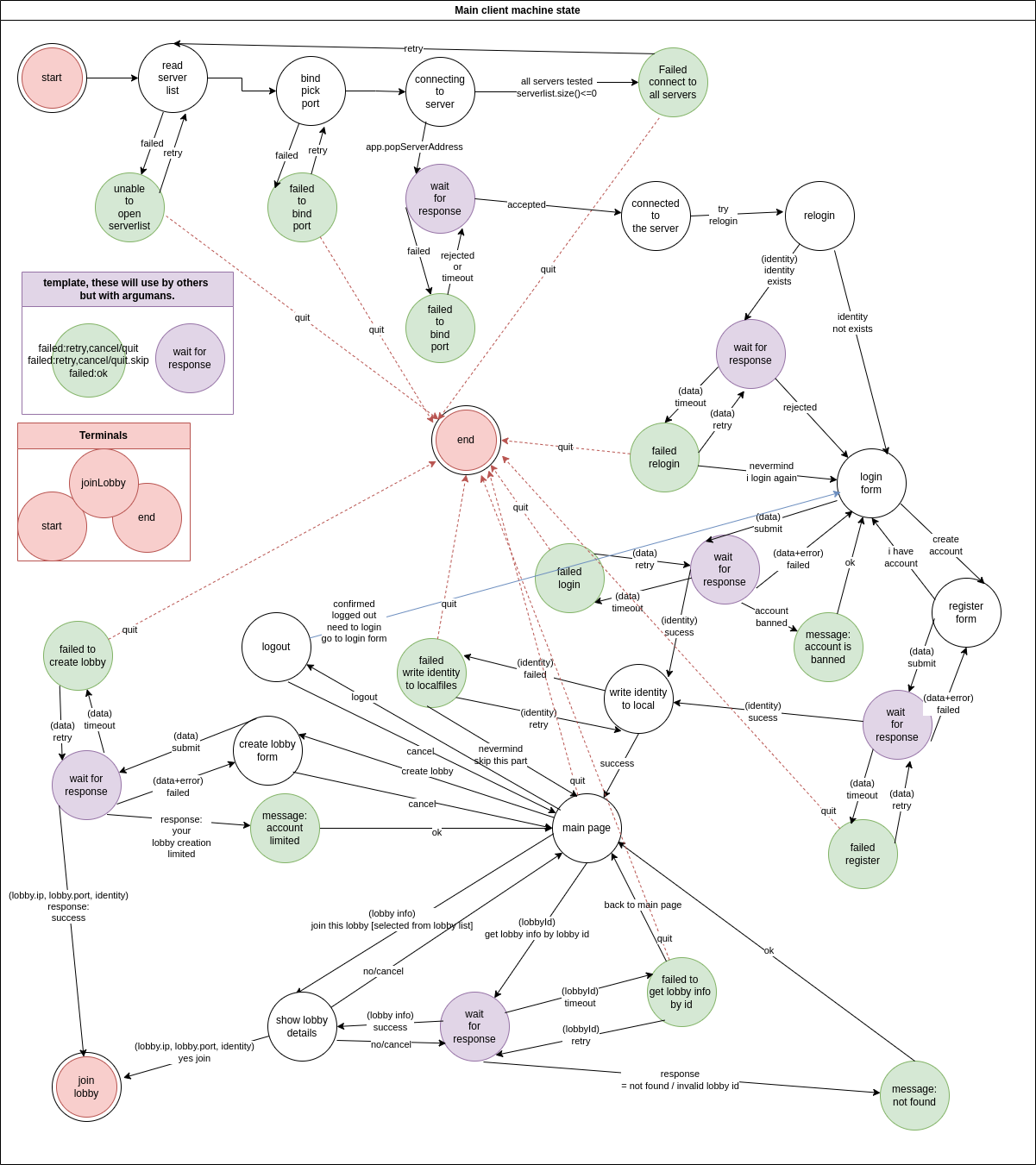
در این بخش به معرفی فایل‌ها و پوشه های درون این نرم‌افزار خواهیم پرداخت:

جدول 6-3 معرفی فایل ها و پوشه های Main Client

|  |  |  |
| --- | --- | --- |
| **نام فایل یا پوشه** | **کاربرد** | **هدف** |
| assets | نگهداری محتوای برنامه و در کنار برنامه در سمت کاربر قرار خواهد گرفت. | جدا سازی محتوا (صوت . تصویر. فونت و..) از کد ها |
| assets/font | نگهداریFont برنامه و در کنار برنامه در سمت کاربر قرار خواهد گرفت. | نگهداری و جدا سازی فایل Font  از سایر محتواها |
| assets/uiComponents | نگهداری Texture برنامه و در کنار برنامه در سمت کاربر قرار خواهد گرفت. | تعداد تصاویر ممکنه زیاد باشند بنابر این نیاز به دسته بندی آن است. |
| Include/states | نگهداری وضعیت‌های توسعه داده شده. | تعداد وضعیت‌ها زیاد است و ممکن است خوانایی برنامه و فایل‌ها را کمتر کند و پیدا کردن آن‌ها سخت باشد. بنابر این تمامی وضعیت‌های توسعه داده شده در این پوشه قرار داده شده است. |
| Include/uiComponents | نگهداری مولفه های توسعه داده شده | تعداد مولفه های توسعه داده شده زیاد است بنابر این نیاز است آن‌ها را دسته بندی کنیم برای خوانایی و تمیزی پروژه. |
| App.h | نگهداری و راه اندازی  سوکت  لیست اتاق بازی  آدرس سرور  مقدار identity  نخ های ارسال و دریافت  لیست سرور های یافت شده از فایل serverList.txt  لیست اطلاعات دریافتی  لیست اطلاعات ارسالی  پنجره اصلی برنامه  لیستی از پیام‌ها (Notification) | تمیز نگه داشتن فایل main همچنین سازماندهی کردن و گذاشتن محدودیت دسترسی به متغیر ها.  ارسال راحتتر شئ و تابع ها با ارسال یک شئ از نوع App |
| AppState.h | قالب و چهارچوبی برای توسعه وضعیت ها | از آنجایی که نیاز است با اشاره گر به وضعیت اشاره کرد و توابع آن را صدا زد نیاز است یک ساختار یکپارچه و قالبی وجود داشته باشد که توابع ضروری وضعیت‌ها را حین توسعه کنترول و مدیریت کند و توسط این فایل و کلاس که به عنوان پدر به تمامی وضعیت‌ها ارث داده شده است. |
| Config.h | نگهداری مقدار های حیاتی برنامه و همچنین مقدار های فاصله نام فایل‌های Texture و … این برنامه و رابط کاربری آن. | تمیز و خوانا نگهداشتن متغیر ها و مقدار های آن‌ها در زمان پیشرفض یا زمانی که متغیر در فایل‌های زیادی قرار داده شده است و همچنین از آنجایی که در این برنامه رابط کاربری داریم تمامی مقدار و عدد های مربوط به فاصله Padding و امثال آن در این فایل با پیشوند نامگذاری شده‌اند برای تغییر و دسترسی سریعتر. |
| DataSecurity.h | رمز گذاری و رمز گشایی  اطلاعات ارسال شده توابع آن | برقراری امنیت و جدا سازی کد های مربوطه به آن از سایر بخش ها. |
| DataType.h | نگهداری و جمع آوری داده‌ها و نوع های پر تکرار و مورد استفاده توسط فایل‌های متفاوتنگهداری و جمع آوری داده‌ها و نوع های پر تکرار و مورد استفاده توسط فایل‌های متفاوت | در پروژه نوع ها (Type) های داده‌ای زیاد خواهند بود و درصورتی که هر بخش/کلاس نوع خود را کنار خود بنویسد در این امر خوانایی کد را کمتر خواهد کرد (برای مثال ویرایش سخت خواهد بود) همچنین در پروژه بخش‌های مختلف نوع ها را نیاز دارند با این روش جلوگیری خواهیم کرد از include کل کلاس یا فایل به فایل/تابع مورد نظر که این امر از مشکلات include های زیاد (مشکل circular dependencies را کمتر خواهد کرد). |
| Server-request-response-list.h | نگهداری کد های درخواست و پاسخ با نام خوانا | در ابتدای پیام دریافتی یا ارسالی از یا به سوی سرور یک کد سه رقمی قرار دارد و برای تمیز بودن و توسعه راحتتر کد از نام های واضح و عدد ها استفاده شده و این فایل وظیفه نگهداری آن را دارد. این فایل با فایل مربوطه با همین نام در بخش Main Server باید یکسان باشند تا در ارتباط کاربر و سرور کد های دستوری یکدیگر را متوجه بشوند. |
| States.h | تمامی وضعیت‌ها را در این بخش نگهداری می‌کنیم بصورت include و این فایل را در بخش source وضعیت‌ها اضاف می‌کنیم برای کد هر وضعیت نیاز به داشتن تعریفی از سایر وضعیت‌ها داریم به جهت ساخت شئ و ارسال آن به عنوان خروجی | جلوگیری از مشکلات include های زیاد مشکل  (circular dependencies) |
| UdpSocket.h | تعریف سوکت و مقدار buffer و دریافت کنند. | با جدا سازی سوکت از سایر بخش‌های برنامه امکان افزایش خوانایی و ماژولار کردن برنامه خواهد بود. |
| ServerList.txt | یک فایل که آدرس سرور های Main Server را در خود ذخیره می‌کند. این فایل در کنار فایل‌های سمت کاربر قرار خواهد گرفت. | برای آینده پیش‌بینی شده است که به راحتی بتوان درصورت مشکل در اتصال به یک سرور به سایر سرور ها دسترسی داشت. |
| Identity.ini | یک فایل که مقدار identity دریافتی از سوی سرور را در خود نگهداری می‌کند.  این فایل در کنار فایل‌های سمت کاربر قرار ساخته خواهد شد. | جهت افزودن قابلیت ذخیره و ورود به حساب کاربری بدون وارد کردن نام کاربری و کلمه عبور در مدت محدود |

سناریو استفاده و اجرای این برنامه بدین صورت است که در ابتدا پس از اجرا شئ تولید خواهد شد با نوع App و اشاره گر مقادیری دارد که با استفاده از تابع init آن‌ها را مقداری دهی خواهد کرد و سپس برنامه با صدا زدن تابع run آن شئ شروع به اجرای اصلی و نمایش رابط کاربری خواهد کرد.

هر وضعیت یک وظیفه ی خاص دارد بنابر این نیاز است آن‌ها را توضیح دهیم و لیست کنیم و در جدول بالا می‌توانید توضیحات مربوط به وضعیت را ببینید. در ابتدا اشاره گر وضعیت برنامه به وضعیت ReadServerList اشاره می‌کند و سپس آن وضعیت نسبت به شرایط و تصمیم خود وضعیت جدیدی را به عنوان خروجی باز می گرداند. و این تصمیم های وضعیت‌ها در قالب یک نقشه (که به آن ماشین وضعیت می‌گویند) طراحی و ترسیم شده است که می‌توان در تصویر زیر جزئیات بیشتری از آن را ببینید که هر وضعیت ممکن است به کدام وضعیت اشاره کند و با این روش می‌توان روند اجرای برنامه را حدس زد.



تصویر 7-3 وضعیت ماشین Main Client

در تصویر بالا که وضعیت ماشین را مشاهده کردید. این نرم‌افزار با استفاده از یک اشاره گر یا همان (Pointer) وضعیت را نگهداری می‌کند و درصورتی که وضعیت جاری تمام شد و یا صلاح دید آن این بود که وضعیت را نیاز است تغییر کند. مقدار وضعیت جدید را باز می گرداند و آن در اشاره گر مقدار دهی می‌شود و سپس در حلقه اصلی توابع مربوط به آن وضعیت صدا زده خواهد شد.

جدای از توضیح وضعیت‌ها و ماشین وضعیت‌ها برنامه بدین صورت کار خواهد کرد که در ابتدا یک فایل به نام serverList.txt در کنار فایل اجرای برنامه قرار دارد که حاوی آدرس آیپی و پورت سرور های اصلی است. هدف قرار دادن این فایل و وضعیت آینده بینی این نرم‌افزار است که ممکن است در آینده با زیاد شدن کاربر ها و زیاد شدن ترافیک شبکه و پردازش آن‌ها تصمیم بگیریم که بجای یک Main Server چندین سرور داشته باشیم برای پاسخ‌گویی به نیاز های این بخش و به راحتی با جابجایی پایگاه داده به یک سرور مرکزی و متصل کردن Main Server ها و قرار دادن آدرس سرور ها بصورت لیست در این فایل امکان‌پذیر خواهد بود. درصورتی که کاربر نتواسنت به سرور اول متصل شود برنامه به سراغ سرور دوم و بعدی.. تلاش‌های خود را برای اتصال خواهد کرد.

بدین صورت بعد از خواندن فایل آدرس سرورها آن‌ها را در یک لیست قرار خواهد داد و درخواست مورد نظر را به سرور ارسال خواهد کرد و منتظر پاسخ می‌ماند. لیست درخواست و پاسخ‌های این بخش با لیست درخواست و پاسخ Main Server که در جدول ۵-۳ آمده شده یکی است و بنا به پاسخ گرفته شده پیام یا کار مورد نیاز توسط توابع انجام داده خواهد شد.

از آنجایی که وضعیت‌ها در فایل‌های مختلف با وظایف مختلف توسعه داده شده‌اند ما نیاز به یک چهار چوب و قالب داریم که تحت آن‌ها توسعه یابند (از آنجایی که با اشارگر به وضعیت اشاره می‌کنیم و توابع آن را صدا می‌زنیم نیاز است توابع یکسانی داشته باشند بنابر این یک کلاس با نام AppState تولید می‌کنیم و تمامی وضعیت‌ها تحت قالب این توابع Pure و اجباری توسعه و نوشته می‌شوند. در جدول – توابع اجباری که هر وضعیت باید آن‌ها را پیاده‌سازی کند را توضیح داده‌ایم.

جدول 3-7 توابع قالب AppState

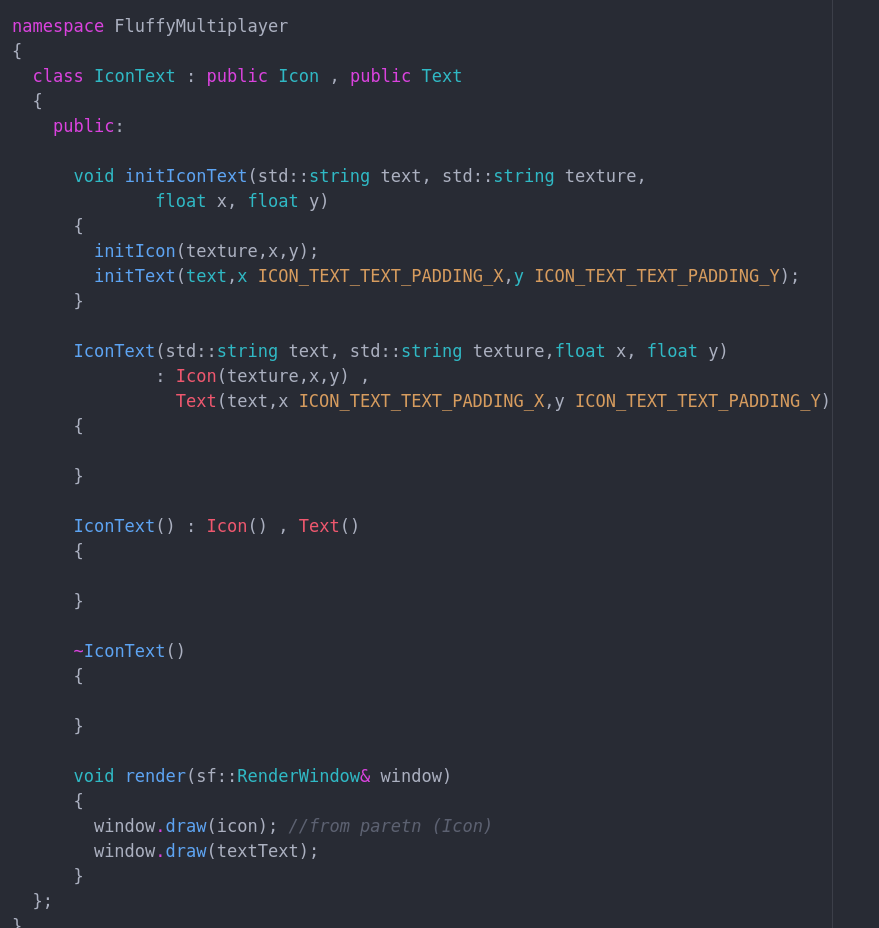
|  |  |
| --- | --- |
| **کاربرد و هدف** | **تابع** |
| در این کلاس یک متن بصورت پیشفرض قرار گرفته شده است که برای دسترسی سریع وضعیت‌ها به یک متن جهت نمایش به کاربر (حداقل هر وضعیت یک متن دارد)  کاربرد این تابع این است که راه اندازی و مقدار دهی به آن شئ متن. | initSimpleText |
| تنظیم و جابجایی مختصات شئ. | setSimpleTextPosition |
| جهت ترسیم مولفه های آن وضعیت. | virtual void render=0 |
| جهت قرار دادن منطق و کد های هر وضعیت. | virtual FluffyMultiplayer::AppState\* update=0 |
| جهت ارسال رویداد به وضعیت تا بتواند نسبت به رویداد کاربر واکنش مناسب بدهد. | virtual FluffyMultiplayer::AppState\* eventHandle=0 |

توضیحات بیشتر درمورد وضعیت‌های توسعه داده شده:

جدول 3-8توضیح درباره وضعیت ها

|  |  |  |  |
| --- | --- | --- | --- |
| **نام وضعیت** | **وظیفه** | **نام وضعیت** | **وظیفه** |
| State  Read  ServerList | خواندن فایل  serverList.txt و جدا سازی آدرس و پورت ها و افزودن آن‌ها به لیست سرور که در App تعریف شده | State  Register  Form | نمایش فرم ثبت نام و برسی رویداد های و واکنش به آن‌ها |
| State  Bind  Pick  Port | انتخاب یک عدد تصادفی و قرار دادن آن به عنوان پورت برنامه و برسی آزاد بودن آن پورت همچنین آن پورت را به سوکت اصلی بدهد | State  WriteIdentity  To  Local | نوشتن مقدار  identity در  فایل identity.ini |
| State  Connecting  To  Server | ارسال درخواست اتصال اولیه به سرور انتخاب شده | State  Main  Page | نمایش صفحه اصلی برنامه و همچنین ارسال یک درخواست برای دریافت لیست اتاق‌های بازی و نمایش لیست اتاق‌های بازی با چند دکمه جهت انجام درخواست و عملیات توسط کاربر. |
| State  Wait  For  Response | پس از وارد شدن به این وضعیت یک شمارنده وضعیت TimeOut را برسی می‌کند درصورتی که لیست دریافت اطلاعات خالی بود به سایر وضعیت‌هایی که گرفته است منتقل خواهد شد. درصورت تأیید و دریافت اطلاعات به وضعیت داده شده منتقل خواهد شد. | State  Logout | نمایش پیام تأیید برای خروج از حساب کاربری و صدا زدن تابع clearIdentity از شئ app |
| State  Connected  To  The  Server | پس از اتصال به سرور به این وضعیت منتقل می‌شویم و درصورتی که در آینده پس از اتصال اقدامی نیاز است در این بخش نوشته خواهد شد. | State  Create  Lobby  Form | نمایش فرم ساخت اتاق بازی و برسی رویداد های و واکنش به آن‌ها |
| State  Relogin | تلاش برای ورود مجدد به حساب کاربری درصورت وجود فایل identity.ini | State  Join  Lobby | نمایش پیغام اتاق بازی اجرا شده به همراه دو دکمه بازگشت و خروج همچنین صدا زدن تابع openGame از شئ app |
| State  Login  Form | نمایش فرم ورود به حساب کاربری و برسی رویداد های و واکنش به آن‌ها | State  Failed | نمایش پیغام ناموفق بودن در عملیات یا دریافت اطلاعات و با زدن دکمه تلاش مجدد به وضعیت مورد نظر انتقال دهد. |
| State  End | نمایش پیغام تأیید خروج همراه با دو دکمه تأیید و بازگشت | State  Show  Lobby  Details | نمایش اطلاعات اتاق بازی انتخاب شده همراه با دو دکمه که یکی بازشگشت و دیگری به نام ورود به اتاق |

همانطور که در فصل یک اشاره شد این برنامه توسط کتابخانه گرافیکی سطح متوسط توسعه یافته است بنابر این قابلیت‌های آن کتابخانه جهت توسعه رابط کاربری و فرم‌ها نیازمند اقداماتی می‌باشد برای مثال: گرفتن ورودی از کاربر با استفاده از یک المان همچون Text Input وجود ندارد و باید آن‌ها را بسته به نیاز خود توسعه داد. درست است با استفاده از فریم ورک ها و متور های بازی سازی مناسب این مشکلات را نخواهیم داشت و توسعه روند سریعتر و بهتری خواهد داشت ولی بنا بر انگیزه ما این روش‌ها را استفاده و پیاده‌سازی خواهیم کرد.

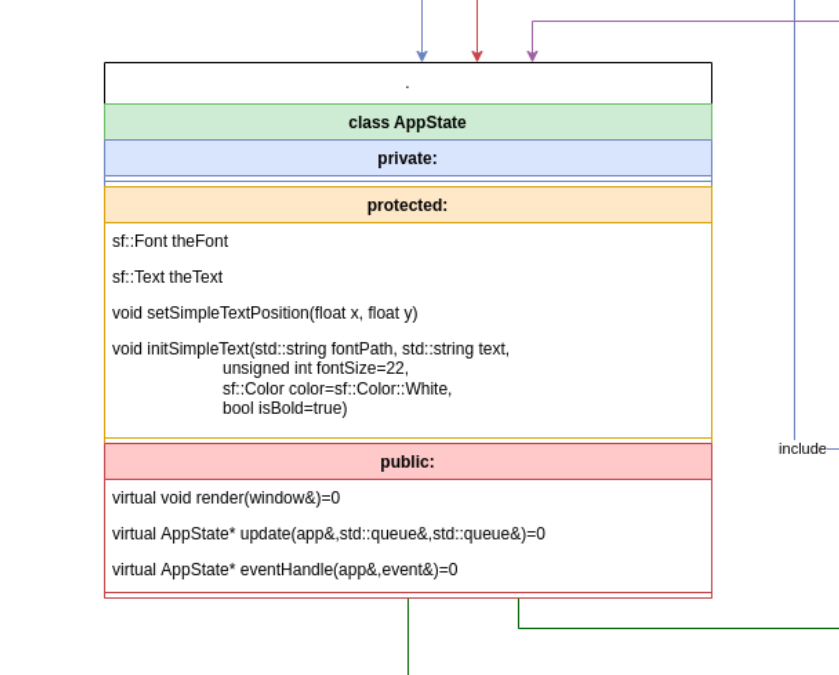


تصویر 8-3 کد نمونه تولید مولفه IconText

با استفاده از مولفه ها (Component) های توسعه داده شده فرم‌ها و بخش‌های رابط کاربری را توسعه داده‌ایم.

جدول 9-3 جزئیات مولفه های (Component) توسعه یافته

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **نام** | **کاربرد** | **تابع و**  **کاربرد آن‌ها** | **تأثیر گرفته از** |
| ۱ | Button | یک تصویر را به عنوان پیش‌زمینه قرار می‌دهد و سپس یک متن را در وسط آن قرار داده و متن درخواستی ما را در آن جایگزاری می‌کند و مختصات و ابعاد را در یک متغییر sf::FloatRect قرار می‌دهد و با یک تابع آن را اعلام می‌کند همچنین رنگ بندی و سایر هم قابل شخصی سازی برای هر شئ می‌باشد. | |  |  | | --- | --- | | تنظیم و جابجایی مختصات شئ | setPosition | | اعلام مختصات و ابعاد شئ | getButtonBound | | تنظیم رنگ متن دکمه | setForgroundColor | | تنظیم رنگ زمینه دکمه | setBackgroundColor | | تنظیم ابعاد دکمه | setSize | | راه اندازی دکمه با مقادیر اولیه و یا مقادیر داده شده. | init | | تابع سازنده که صورت‌های مختلفی دارد جهت استفاده های مختلف و مقادیر دریافتی را به تابع init ارسال خواهد کرد | Button | | پنجره را بصورت رفرنس دریافت می‌کند و دو شئ (متن و تصویر زمینه) را با دستور draw ترسیم خواهد کرد. | render | | UiComponent |
| ۲ | Check  Box | این مولفه شامل یک دکمه با تغییر تصویر زمینه آن می‌باشد که با یک تابع تصویر زمینه دکمه قابل تغییر است همچنین شامل یک متن می‌باشد جهت نمایش در کتار دکمه که هر دو آن‌ها با بر طبق یک نقطه مختصات دهی می‌شوند و مقدار Padding آن‌ها در فایل Config.h قرار دارد. | |  |  | | --- | --- | | به عنوان ورودی یک مقدار ۰ یا ۱ را می‌گیرد و وضعیت دکمه را تغییر می‌دهد  (دو وضعیت دارد Checked و Unchecked) | setStatus | | تنظیم و جابجایی مختصات شئ | setPosition | | اعلام مختصات و ابعاد شئ | getBound | | راه اندازی CheckBox با مقادیر اولیه و یا مقادیر داده شده | init | | مقدار متغیر isChecked را باز می گرداند | getStatus | | پنجره را بصورت رفرنس دریافت می‌کند و دو شئ (متن و تصویر زمینه) را با دستور draw ترسیم خواهد کرد | render | | تابع سازنده که صورت‌های مختلفی دارد جهت استفاده های مختلف و مقادیر دریافتی را به تابع init ارسال خواهد کرد | CheckBox | | UiComponent |
| ۳ | Icon | یک تصویر را به نمایش می‌گذارد و با کلیک بر روی آن هیچ اتفاقی رخ نخواهد داد. | |  |  | | --- | --- | | تغییر تصویر شئ | changeIconTexture | | تنظیم و جابجایی مختصات شئ | setIconPosition | | راه اندازی Icon با مقادیر داده شده. | initIcon | | پنجره را بصورت رفرنس دریافت می‌کند و دو شئ (تصویر) را با دستور draw ترسیم خواهد کرد | render | | تابع سازنده که صورت‌های مختلفی دارد جهت استفاده های مختلف و مقادیر دریافتی را به تابع initIcon ارسال خواهد کرد. | Icon | | UiComponent |
| ۴ | Icon  Text | همانند مولفه Icon یک تصویر را دریافت می‌کند.  همچنین یک متن را در زیر آن تصویر نمایش خواهد داد و قابلیت کلیک ندارد | |  |  | | --- | --- | | مقادیر گرفته شده را به توابع initIcon و initText ارسال خواهد کرد | initIconText | | تابع سازنده که صورت‌های مختلفی دارد و مقادیر دریافت شده را با صدا زدن تابع های سازندهای Icon و Text ارسال می‌کند. | IconText | | ترسیم دو شئ که در مولفه های پدر موجود دارن اقدام می‌کند | render | | بصورت غیر مستقیم از  UiComponent  مستیم از  Icon  و  Text |
| ۵ | Text  Input | یک تصویر را به عنوان پیش‌زمینه ورودی متنی و یک متن را در وسط آن قرار می‌دهد دو رشته را نگهداری می‌کند که یکی مقدار وارد شده توسط کاربر می‌باشد د دیگری به عنوان PlaceHolder درصورت خالی بودن متن ورودی آن را نمایش خواهد داد و قابلیت کلیک بر روی آن هم وجود دارد. دو متن در بالای ورودی قرار خواهد گرفت و یکی به عنوان متن خطا  یکی به عنوان Title نمایش داده خواهند شد. | |  |  | | --- | --- | | اعلام مختصات و ابعاد شئ | getInputBoxBound | | تنظیم و جابجایی مختصات شئ | setPosition | | تنظیم ابعاد | setSize | | راه اندازی TextInput با مقادیر اولیه و یا مقادیر داده شده | init | | تنظیم متن عنوان بالای TextInput | setTitles | | تنظیم متن خطای بالای TextInput | setError | | تنظیم متن درون (متن وارد شده) | setText | | تنظیم متن placeHolder | setPlaceholder | | تابع سازنده که صورت‌های مختلفی دارد و مقادیر دریافت شده را با صدا زدن تابع های سازندهای init ارسال می‌کند | TextInput | | حذف یک عبارت از رشته وارد شده توسط کاربر | removeFromText | | افزودن یک عبارت به رشته وارد شده توسط کاربر | appendToText | | فعال کردن PlaceHolder و نمایش آن | enablePlaceholder | | غیرفعال کردن PlaceHolder و نمایش آن | disablePlaceholder | | ترسیم تصویر زمینه. درصورت فعال بودن وضعیت hideTitleAndError دو عنوان بالای مولفه ورودی و همچنین یک متن در وسط را ترسیم می کند. | render | | یک تابع که بصورت مجازی تعریف شده برای زمانی که می‌خواهیم از این مولفه ارث ببریم و مولفه جدیدی تولید کنیم برای مثال می‌توان در آن یک دکمه جدید در کنار تمامی موارد ترسیم کرد با استفاده از این تابع این امر امکان‌پذیر خواهد بود. | specificRender | | رشته وارد شده توسط کاربر را باز می گرداند. | getEnteredText | | UiComponent |
| ۶ | Text | برای نمایش متن ساده یا همان Label در فرم‌ها استفاده می‌شود. | |  |  | | --- | --- | | تنظیم و جابجایی مختصات شئ | setTextPosition | | تنظیم متن | setText | | تنظیم اندازه متن | setFontSize | | فایل فونت را برای متن جاگزاری می‌کند با استفاده از تابع componentLoadAndSetFont  و سپس رشته ورودی را در متن و مختصات قرار گیری آن را تنظیم می‌کند. | initText | | تابع سازنده که صورت‌های مختلفی دارد جهت استفاده های مختلف و مقادیر دریافتی را به تابع initText ارسال خواهد کرد. | Text | | پنجره را بصورت رفرنس دریافت می‌کند و متن را با دستور draw ترسیم خواهد کرد. | render | | UiComponent |
| ۷ | Spin  Box | برای انتخاب یک مورد از یک لیست داده از این مولفه استفاده می‌کنیم.  که یک متن و دو دکمه در جلوی متن (هر دو دکمه بالا و پایین قرار دارند).  برای مثال انتخاب مدل بازی توسط این مولفه انتخاب می‌شود. یک لیست بصورت رشته ای دریافت می‌کند و سپس یک عدد صحیح را به عنوان مورد انتخاب شده (نگهداری اندیس) و با صدا زدن. | |  |  | | --- | --- | | تنظیم و جابجایی مختصات شئ | setPosition | | اعلام مختصات و ابعاد شئ | getBound | | تنظیم مقادیر موجود (لیستی از رشته) و یک عدد صحیح به عنوان مورد انتخاب شده پیش‌فرض | setItems | | گرفتن اندیس یا مقدار انتخاب شده. | getCurrentIndex | | مورد بالایی را انتخاب می‌کند | spinUp | | مورد پایین را انتخاب می‌کند | spinDown | | راه اندازی SpinBox با مقادیر اولیه و یا مقادیر داده شده. | init | | تابع سازنده که صورت‌های مختلفی دارد برای دریافت و انتقال مقادیر به تابع init بعد از تولید شئ صدا زده می‌شود | SpinBox | | ترسیم دو دکمه (که حاوی تصویر می‌باشند) و یک متن | render | | UiComponent |
| ۸ | Picture  Button | یک دکمه است با تصویر که یک متن در زیر آن قرار دارد. | |  |  | | --- | --- | | تنظیم و جابجایی مختصات شئ | setPosition | | اعلام مختصات و ابعاد شئ | getButtonBound | | راه اندازی PictureButton با مقادیر اولیه و یا مقادیر داده شده | init | | ترسیم متن و دکمه شئ | render | | تابع سازنده که صورت‌های مختلفی دارد برای انتقال داده‌ها به تابع init هنگاه ساخت شئ | PictureButton | | UiComponent |
| ۹ | Lobby  Cell | برای لیست اتاق بازی نیاز به این مولفه داریم که اطلاعات هر اتاق بازی را جداگانه نمایش دهد و بتوان آن را در به تعداد زیاد تولید کرد و کنار هم قرار داد.  مشخصات را با استفاده از متن و تصویر به نمایش می‌گذارد و قابلیت کلیک را داراست.  اطلاعات اتاق را با استفاده از یک شئ از نوع  LobbyData  نگهداری می‌کند. | |  |  | | --- | --- | | اعلام مختصات و ابعاد شئ | getButtonBound | | مشخصات اتاق بازی که به نمایش در آورده را باز  می گرداند. | getLobbyData | | راه اندازی LobbyCell با مقادیر اولیه و یا مقادیر داده شده و تصمیم برای قرار دادن تصویر مورد نیاز برای هر مولفه برای مثال وقتی وضعیت ارتباط متنی خاموش است نیاز است تصویر متناسب را در Icon ساخته شده قرار دهد. | init | | تابع render شئ های درون خود را صدا میزند و پنجره را به آن‌ها متنقل می‌کند که آن‌ها موارد مورد نیاز را ترسیم کنند. | render | | تابع سازنده که صورت‌های مختلفی دارد و مقادیر دریافت شده را به تابع init انتقال می‌دهد. | LobbyCell | | UiComponent  ولی  از اشیا‌ء بالا استفاده شده  مثل  Icon  و  Text  و  IconText |
| ۱۰ | Notification  Box | برای نمایش یک popup نیازمند لیست کردن آن‌ها هستیم که فقط حاوی متن نیستند بنابر این یک لیست از این شئ خواهیم ساخت و مقادیر را در آن اضافه می‌کنیم. این مولفه حاوی تصویر زمینه و عنوان و متن وسط و همچنین دو دکمه که با فشردن آن متن خطا کپی می‌شود و دیگری این popup را می‌بندد.  اطلاعات در یک متغیر از نوع NotificationData نگهداری می‌کنیم. | |  |  | | --- | --- | | راه اندازی NotificationBox با مقادیر داده شده | init | | اعلام مختصات و ابعاد شئ | getBackofBackgroundBound | | وضغیت boxClosed را به خروج تغییر می‌دهد | closeBox | | جهت کپی کردن متن خطا قرار داده شده. (این بخش به صورت کامل کار نمی‌کند و نیاز است کتابخانه مورد نیاز برای CopyClipBoard وارد و اعمال شود). | copyError | | تابع سازنده که هیچ کاری نمی‌کند و برای مقدار دهی این مولفه نیاز است حتماً init صدا زده و مقدار دهی شود. | NotificationBox | | درصورت True بودن وضعیت متغیر boxClosed مولفه ها و شئ های درون را با انتقال پنجره به تابع render ترسیم خواهد کرد. | render | | UiComponent  ولی  از اشیا‌ء بالا استفاده شده  مثل  Icon  و  Text |



تصویر 9-3 نمونه UML کلاس AppState

فایل‌های مستندات مربوط به این نرم‌افزار در پوشه docs قرار داده شده‌اند می‌توانید با مراجعه به آن اطلاعات بیشتری کسب کنید.

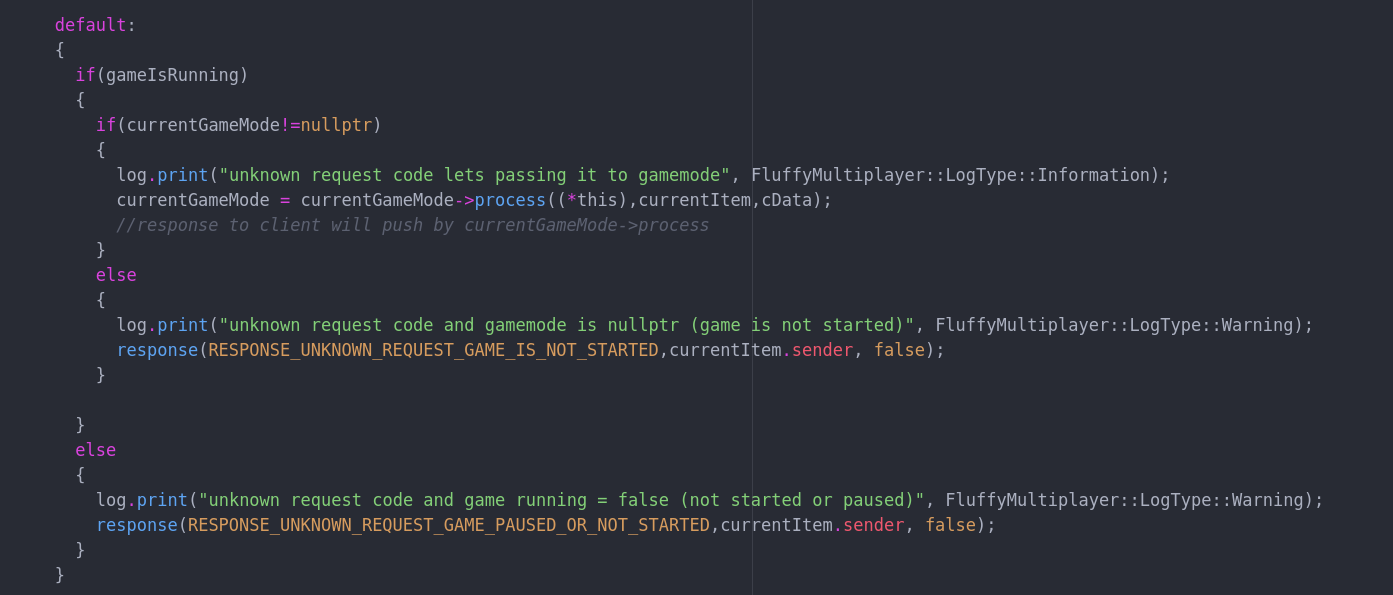
**برنامه Game Server:**

این برنامه جهت اجرا در سمت سرور توسعه یافته است و وظیفه پاسخ‌گویی به درخواست های بازی و اتاق بازی را داراست. در این برنامه مدل های بازی قرار میگیرند و بعد از هر تغییرات در مدل بازی یا خود بخش مدیریت اتاق بازی صورت گیرد ورژن این برنامه افزایش می‌یابد و اگر کاربر ورژنش کمتر از سرور باشد و یا کلمه عبور یا هویت نامعتبر باشد پیغام مورد نیاز را به کاربر می‌دهد و از ورود کاربر جلوگیری می‌کند.



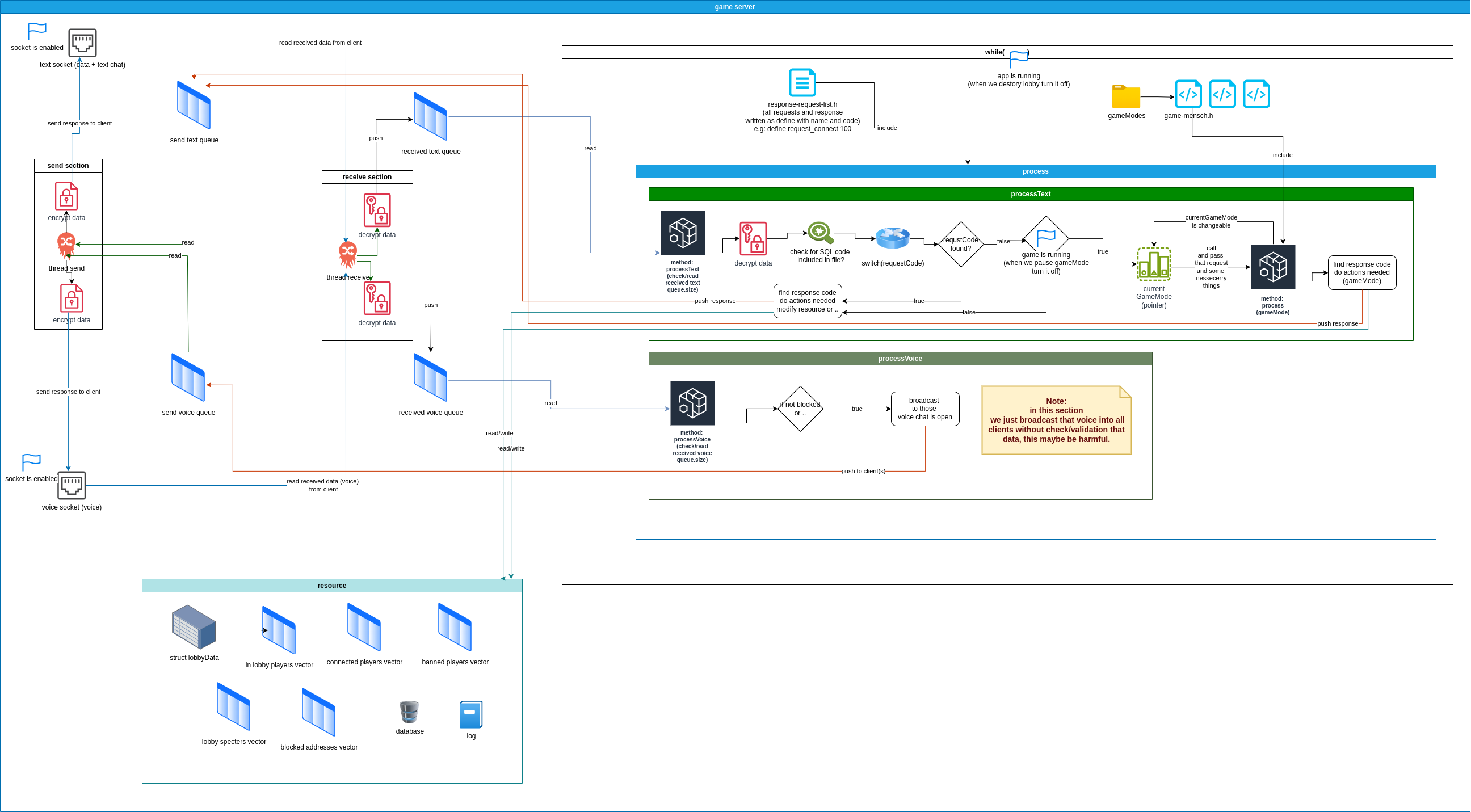
تصویر 10-3 کد پردازش درخواست ورود به اتاق بازی

این برنامه تفاوت‌هایی با برنامه‌های قبل دارد و آن هم این است که مدل بازی وارد خواهد شد و نیاز است ساختار طوری باشد که همزمان بجز پاسخ‌گویی به درخواست های اتاق بازی نیاز است نسبت به مدل بازی انتخاب شده به درخواست ها رسیدگی کند.

تصویر 11-3 کد ارسال درخواست جاری درصورت تعریف نشدن در درخواست های اتاق بازی

نگاهی به وظایف این برنامه داشته باشیم:

* تأیید هویت کاربر
* تأیید و برسی کلمه عبور اتاق
* تأیید و برسی یکی بودن نسخه کاربر و سرور
* متصل کردن کاربران به اتاق بازی (همچنین اعمال تغییرات در پایگاه داده)
* پاسخ به درخواست ورود و خروج از اتاق بازی و اعلام آن به سایر کاربران درون اتاق
* درصورت خروج مدیر از اتاق مدیریت را انتقال کند به سایر کاربران
* درصورت خروج تمامی کاربران اتاق بازی را تخریب کند
* شروع و توقف بازی
* ارسال پیام متنی
* ارسال پیام صوتی
* ارسال تنظیمات جدید (درصورتی که مدیر ویرایشی انجام دهد)‌ به کاربران
* ارسال پاسخ‌ها بصورت گروهی و انفرادی به کاربران اتاق و همچنین تماشاگران درون اتاق
* اجرای منطق و کد های یک مد بازی
* دریافت درخواست ها و ارسال آن‌ها به مد بازی جهت پردازش آن‌ها
* نمایش سمت های کاربرانی مثل مدیر اتاق. ادمین. تماشاگر. وضعیت ارتباط صوتی کاربر برای هر کاربر
* قابلیت مدیریت و تغییر تنظیمات اتاق توسط مدیر اتاق
  + تغییر مدل بازی
  + تغییر و انتقال مالکیت اتاق به سایر کاربران متصل شده
  + ویرایش کلمه عبور اتاق بازی
  + حذف اتاق بازی
  + ویرایش وضعیت مجاز بودن ارتباط صوتی
  + ویرایش وضعیت مجاز بودن ارتباط متنی
  + ویرایش وضعیت مجاز بودن ورود تماشاگر به اتاق
  + ویرایش تعداد حداکثر کاربران مجاز ورود به اتاق

تصویر 12-3 کار کرد کلی Game Server

معماری و ساختار این برنامه پیشرفته‌تر از برنامه‌ها Main Server و Main Client است. به این دلیل که تمامی بخش‌های برنامه بصورت ماژولار توسعه داده شده‌اند. فایل‌ها مشابه با موارد قبلی در Main Server و Main Client هستند ممکن است کمی تفاوت پیدا کرده باشند و برای تکراری نشدن مطالب فقط توضیحاتی درمورد فایل‌های جدید ارائه خواهیم داد:

جدول 10-3 معرفی فایل‌های Game Server

|  |  |  |
| --- | --- | --- |
| **نام فایل / پوشه** | **کاربرد** | **هدف** |
| GameMode.h | ایجاد قالب برای توسعه مدل های بازی و تولید اشاره گر با این نوع | از آنجایی که مدل های بازی متفاوتی خواهیم داشت توسعه آن‌ها بصورتی که بتوان با یک اشاره گر به آن اشاره کرد و توابع یکسانی داشته باشند نیاز به قالب است. |
| GameModes.h | نگهداری مدل های بازی توسعه داده شده بصورت Include | جلوگیری از مشکلات include های زیاد مشکل  (circular dependencies) |
| Lobby-request-response.h | نگهداری کد درخواست و پاسخ‌ها با نام مناسب | در ابتدای پیام دریافتی یا ارسالی از یا به سوی سرور یک کد سه رقمی قرار دارد و برای تمیز بودن و توسعه راحتتر کد از نام های واضح و عدد ها استفاده شده و این فایل وظیفه نگهداری آن را دارد. این فایل با فایل مربوطه با همین نام در بخش Game Client باید یکسان باشند تا در ارتباط کاربر و سرور کد های دستوری یکدیگر را متوجه بشوند. |
| Log.h | ثبت وقایع در فایل یا چاپ آن با ذکر تاریخ و ساعت و نوع در Console | برخی موارد نیاز است ازخطا ها یا اتفاقاتی که در سرور رخ می‌دهد دفترچه ثبت وقایعی داشته باشیم جهت عیب یابی. |
| include/gameModes | نگهداری فایل‌های مدل بازی | مدل های بازی ممکن است زیاد شوند بنابر این با استفاده از این پوشه آن‌ها را جدا می‌کنیم تا خوانایی و تمیزی پروژه حفظ شود. |

هدف از جدا کردن این برنامه از برنامه Main Server این بوده که پروژه بصورت توزیع شده باشد. به این دلایل زیر:

1. درصورتی که تمامی اتاق‌ها در یک سرور اجرا شوند زمانی که تعداد اتاق‌های بازی زیاد شوند یا بار پردازشی زیادی از سمت مدل بازی زیاد شود و یا پهنای باند سرور پر و اشغال شود کاربران تجربه بدی از بازی کردن خواهند داشت و یا ممکن است برنامه توسط سیستم عامل بسته شود به علت استفاده سنگین از منابع.
2. درصورتی که این برنامه با Main Server یکی می‌شد توسعه و تعمیر و نگهداری آن سخت و کار پیچیده‌ای خواهد شد. از حال برای آینده این پروژه پیش‌بینی شده است که در صورت افزایش کاربران به راحتی با تغییرات اندکی پروژه را مخصوصاً اتاق‌های بازی را در سرور های متفاوت پخش کنیم و کاربران مشکلی از سوی اینترنت و یا سرور اتاق بازی تجربه نکنند.

نحوه کارکرد این نرم‌افزار بدین صورت است که کاربر با استفاده از برنامه‌های Main Client و Main Server با پر کردن فرم درخواست ساخت اتاق را می‌دهد و سپس برنامه ی Main Server اطلاعاتی (مشخصات اتاق)‌ را به عنوان Argument به این برنامه هنگام اجرای آن می‌دهد و این برنامه با استفاده از اطلاعات دریافتی مقادیر خود را پر می‌کند و سپس آماده ی پاسخ‌گویی به نیاز ها و درخواست های کاربر ها می‌باشد. با دریافت دستور شروع بازی توسط مدیر اتاق شروع به ارسال درخواست های بازی به سمت مدل بازی در خود می‌کند.

از این رو که مدل بازی‌های مختلف خود کد هایی دارد به برای شناسایی و تفکیک درخواست ها و پاسخ‌های اتاق با مدل بازی نیاز است درخواست ها را تحت محدوده مدیریت کرد. در جدول زیر توضیحات بیشتری خواهیم داد:

جدول 11-3 محدوده کد ها درخواست ها و پاسخ‌های Game Server و Game Client

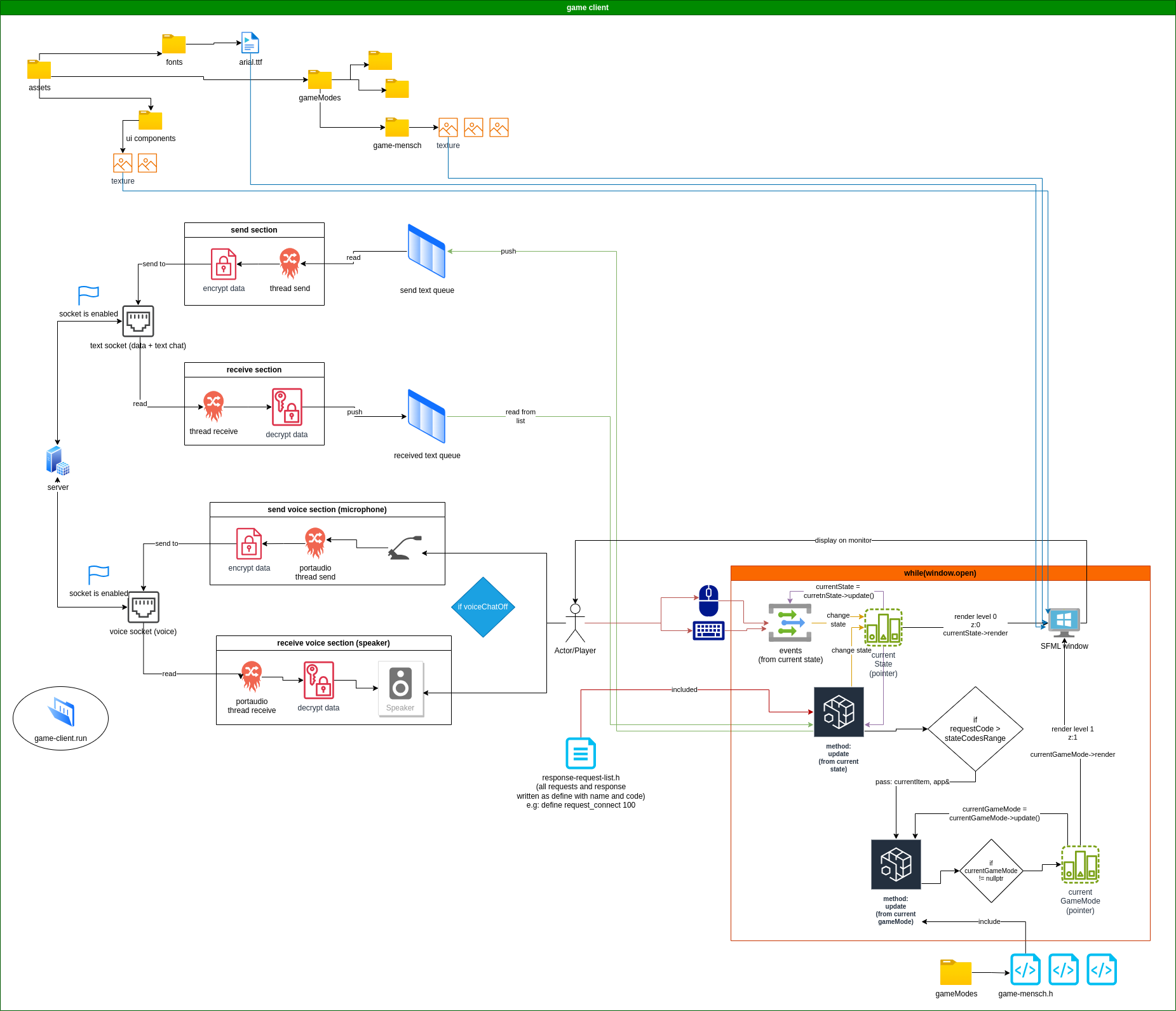
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **شناسه/کد** | **نوع** | **فرستنده** | **گیرنده** | **محدوده کد** |
| ۱ | درخواست به اتاق | کاربر | اتاق بازی | ۱۰۰ تا ۲۰۰ |
| ۲ | پاسخ موفق به درخواست اتاق | اتاق بازی | کاربر | ۲۰۱ تا ۳۰۰ |
| ۳ | پاسخ ناموفق به درخواست اتاق | اتاق بازی | کاربر | ۳۰۱ تا ۴۰۰ |
| ۴ | پاسخ ناموفق به درخواست اتاق  (به دلیل مشکل داخلی سرور) | اتاق بازی | کاربر | ۴۰۱ تا ۵۰۰ |
| ۵ | درخواست به مدل بازی | کاربر | مد بازی | ۵۰۱ تا ۷۰۰ |
| ۶ | پاسخ به درخواست مدل بازی | مد بازی | کاربر | ۷۰۱ تا ۹۹۹ |

جدول 12-3 درخواست و پاسخ Game Server و Game Client

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **نوع** | **کد** | **نام** | **توضیحات** | **حاوی اطلاعات** |
| 1 | 100 | REQUEST  CONNECT  TO LOBBY | درخواست اتصال به اتاق بازی | - |
| 1 | 101 | REQUEST RECONNECT TO LOBBY | درخواست بازیابیی اتصال اتاق بازی (جای گذاری شده برای آینده نرم‌افزار درصورتی که کاربر از اتاق به دلیل مشکلات اینترنتی خارج شده و پس از مدتی دوباره به اتاق متصل می‌شود). | - |
| 1 | 102 | REQUEST DISCONNECT FROM LOBBY | درخواست خروج از اتاق بازی | - |
| 1 | 103 | REQUEST JOIN TO LOBBY | درخواست ورود به اتاق بازی که اطلاعاتی ار به سرور ارسال می‌کند و سرور با برسی آن اطلاعات تصمیم می‌گیرد که این کاربر را متصل کند یا خیر. | Identity کاربر  کلمه عبور وارد شده  نسخه برنامه خود |
| 1 | 104 | REQUEST GET LOBBY SETTINGS | این درخواست مخصوص مدیر اتاق می‌باشد و زمانی که بر روی دکمه تنظیمات اتاق بازی کلیک می‌کند اطلاعات اتاق بازی را به او ارسال می‌کند. | - |
| 1 | 105 | REQUEST UPDATE LOBBY SETTINGS | این درخواست مخصوص مدیر اتاق می‌باشد. پس از تکمیل فرم ویرایش تنظیمات اتاق بازی مدیر اطلاعات جدید اتاق را توسط این درخواست به سرور ارسال خواهد کرد. | تعداد حداکثر کاربران در اتاق  شناسه مدل بازی  وضعیت ارتباط صوتی  وضعیت ارتباط متنی  وضعیت مجوز اتصال تماشاگر  کلمه عبور  شناسه مدیر |
| 1 | 106 | REQUEST\_STOP\_START\_GAME | این درخواست مخصوص مدیر اتاق می‌باشد که با آن باز را شروع یا پایان می‌دهد. | - |
| 1 | 107 | - | - | - |
| 1 | 108 | REQUEST\_DELETE\_LOBBY | این درخواست مخصوص مدیر اتاق می‌باشد که با آن اتاق بازی را تخریب می‌کند. | - |
| 1 | 109 | REQUEST\_KICK\_PLAYER | این درخواست مخصوص مدیر اتاق می‌باشد که برای آینده پیش‌بینی شده که یک کاربر را از اتاق بیرون کند. | - |
| 1 | 110 | REQUEST\_BAN\_PLAYER | این درخواست مخصوص مدیر اتاق می‌باشد که برای آینده پیش‌بینی شده که یک کاربر را از اتاق بیرون کند و دیگر اجازه ورود به آن ندهد و به اصطلاح آن را ممنوع کند.  (برای این مورد یک لیست بن در برنامه پیش‌بینی شده و قرار گرفته و هنگام اتصال کاربر شناسه آن و همچنین آدرس آیپی آن مورد برسی قرار می‌گیرد و از ورود فرد ممنوع شده جلوگیری می‌شود. | شناسه کاربر هدف  دلیل بصورت متنی  زمان ممنوعیت |
| 1 | 111 | - | - | - |
| 1 | 112 | REQUEST\_SEND\_TEXT\_CHAT | کاربران درون اتاق با استفاده از این درخواست می‌توانند پیام متنی را به اتاق ارسال کنند. | متن پیام |
| 1 | 113 | REQUEST\_SEND\_VOICE\_CHAT | کاربران درون اتاق درصورتی که وضعیت ارتباط صوتی آن‌ها فعال باشد می‌توانند پیام‌های صوتی خود را به اتاق ارسال کنند. | محتوای صوتی (به اندازی buffer که در نظر گرفته شده). |
| 1 | 114 | REQUEST\_ENABLE\_DISABLE\_VOICE\_CHAT | با این درخواست کاربران وضعیت جاری ارتباط صوتی خود را برعکس می‌کنند. | - |
| 1 | 115 | REQUEST\_GET\_LOBBY\_INFO | زمانی که مدیر اتاق تنظیمات اتاق را ویرایش می‌کند و به صفحه اصلی باز می‌گردد این درخواست بصورت خودکار به سرور ارسال می‌شود تا آخرین تنظیمات اتاق بازی را دریافت کند. | - |
|  | | | | |
| از این پس برای طولانی نشدن لیست پاسخ‌ها تنها پاسخ‌هایی که حاوی اطلاعات هستند و آن‌ها را به کاربر ارسال خواهد کرد در لیست قرار خواهند گرفت. | | | | |
| 2 | 203 | RESPONSE\_PLAYER\_DISCONNECTED | هنگامی که یک کاربر از اتاق خارج می‌شود این پیام به سایر کاربران ارسال خواهد شد | شناسه کاربری که خارج شده |
| 2 | 204 | RESPONSE\_PLAYER\_JOINT\_INTO\_LOBBY | هنگامی که یک کاربر به اتاق وارد می‌شود اطلاعات آن کاربر با این پیام به سایر کاربران ارسال خواهد شد. | شناسه کاربر  نام کاربری  وضعیت مدیر است (۰ یا ۱)  وضعیت ادمین است (۰ یا ۱)  وضعیت تماشاگر است (۰ یا ۱)  وضعیت ارتباط صوتی (۰ یا ۱) |
| 2 | 206 | RESPONSE\_PLAYER\_KICKED\_FROM\_LOBBY | هنگامی که یک کاربر از اتاق اخراج می‌شود این پیام به سایر کاربران ارسال خواهد شد. | شناسه کاربر هدف |
| 2 | 207 | RESPONSE\_PLAYER\_BANNED\_FROM\_LOBBY | هنگامی که یک کاربر از اتاق ممنوع می‌شود این پیام به سایر کاربران ارسال خواهد شد. | شناسه کاربر هدف |
| 2 | 210 | RESPONSE\_PLAYER\_VOICE\_CHAT\_ENABLED | هنگامی که وضعیت ارتباط صوتی یک کاربر فعال می‌شود این پاسخ به سایر کاربران ارسال خواهد شد. | شناسه کاربر هدف |
| 2 | 211 | RESPONSE\_PLAYER\_VOICE\_CHAT\_DISABLED | هنگامی که وضعیت ارتباط صوتی یک کاربر غیرفعال می‌شود این پاسخ به سایر کاربران ارسال خواهد شد. | شناسه کاربر هدف |
| 2 | 212 | RESPONSE\_PLAYER\_SENT\_VOICE\_MESSAGE | هنگامی که یک پیام صوتی دریافت شده باشد در آن را به سایر کاربران ارسال خواهد کرد. | محتوای پیام صوتی |
| 2 | 213 | RESPONSE\_PLAYER\_SENT\_TEXT\_MESSAGE | هنگامی که یک پیام متنی دریافت شده باشد در آن را به سایر کاربران ارسال خواهد کرد. | محتوای پیام متنی |
| 2 | 216 | RESPONSE\_LOBBY\_SETTINGS\_UPDATED | هنگامی که تنظیمات اتاق بازی ویرایش می‌شوند این پاسخ به تمامی کاربران ارسال خواهد شد. | شناسه اتاق  شناسه مدل بازی  حداکثر کاربران مجاز اتاق  تعداد کاربران جاری  پورت ارتباط صوتی سرور  وضعیت مجوز ارتباط صوتی  وضعیت مجوز ارتباط متنی  وضعیت مجوز ورود تماشاگر  شناسه مدیر |
| 2 | 217 | RESPONSE\_LOBBY\_SETTINGS\_IS | اطلاعات و مشخصات اتاق بازی را بدون فیلتر را به مدیر اتاق ارسال می‌کند. | شناسه اتاق  حداکثر کاربران مجاز  شناسه مدل بازی  تعداد کاربران جاری  وضعیت مجوز ارتباط صوتی  وضعیت مجوز ارتباط متنی  وضعیت مجوز ورود تماشاگر  کلمه عبور  نام و شناسه مدیر جاری |
| 2 | 218 | RESPONSE\_LOBBY\_OWNER\_CHANGED | زمانی که مدیر اتاق از اتاق خارج می‌شود این پیام به سایر کاربران ارسال خواهد شد. | شناسه مدیر جدید |
| 2 | 219 | RESPONSE\_LOBBY\_PLAYERS\_ARE | زمانی که یک کاربر به تازگی به اتاق وارد می‌شود اطلاعات تمامی کاربران قبلی که در اتاق بودند به آن کاربر ارسال خواهد شد  (نکته: هر کاربر شامل این اطلاعات زیر می‌باشد بنابر این ممکن است به تعداد n بار اطلاعات پشت سر هم قرار گرفت باشند). | شناسه کاربر  نام کاربری  وضعیت مدیر است (۰ یا ۱)  وضعیت ادمین است (۰ یا ۱)  وضعیت تماشاگر است (۰ یا ۱)  وضعیت ارتباط صوتی (۰ یا ۱) |
| 2 | 222 | RESPONSE\_LOBBY\_INFO\_IS | مشخصات اتاق بازی را به کاربران باز می گرداند. | شناسه اتاق  حداکثر کاربران مجاز  شناسه مدل بازی  تعداد کاربران جاری  وضعیت مجوز ارتباط صوتی  وضعیت مجوز ارتباط متنی  وضعیت مجوز ورود تماشاگر |
| 3 | 313 | RESPONSE\_ERROR\_UPDATE\_LOBBY\_SETTINGS\_NO\_PERMISSION | زمانی که کاربر درخواست ویرایش تنظیمات اتاق بازی را می‌کند و مجوز و دسترسی لازم را ندارد این پاسخ ارسال خواهد شد. | - |

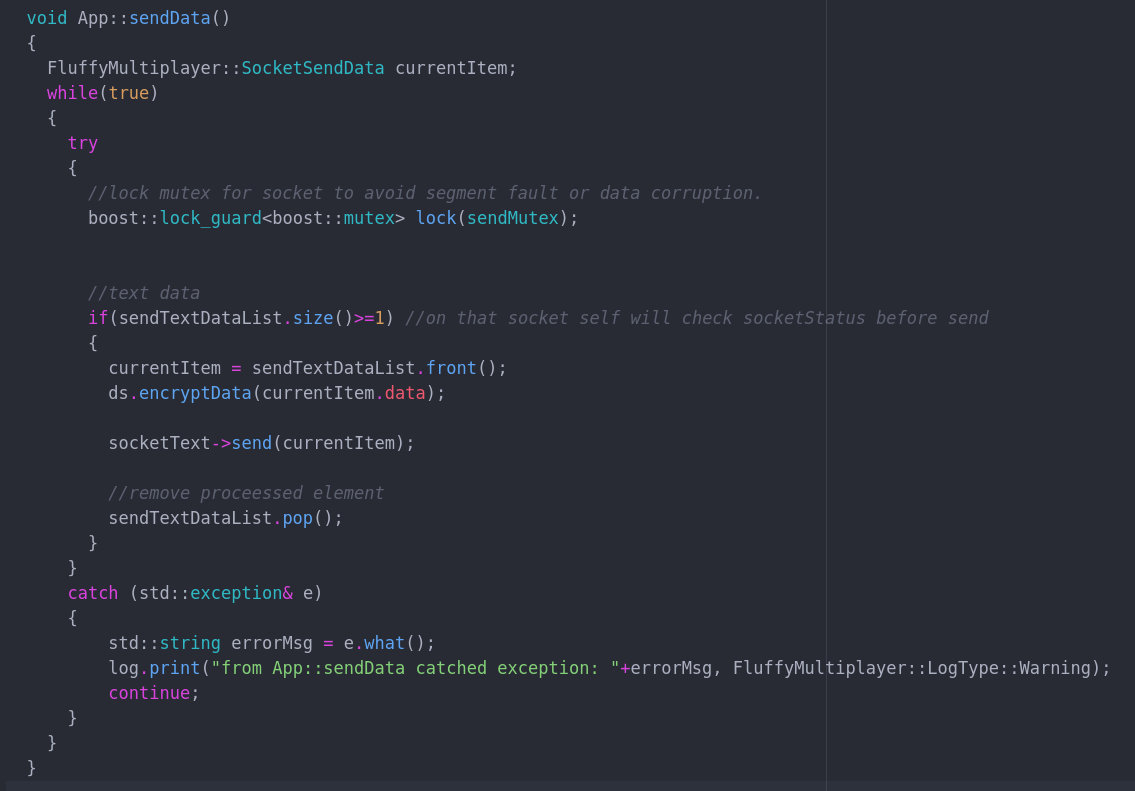
**برنامه Game Client:**

این برنامه مشابه برنامه Main Client است ولی با اصلاح ساختار برای مثال بخش‌های App و UdpSocket و Log مشابه با Game Server توسعه داده شده‌اند. همچنین این برنامه توسط دو سوکت ارسال و دریافت اطلاعات را انجام می‌دهد یکی جهت ارسال درخواست ها و پاسخ‌ها ی مربوط به اتاق و پیام متنی و دیگری برای ارتباط صوتی در نظر گرفته شده است. به این دلیل که ممکن است بعضی مد بازی‌ها اطلاعات زیادی را ارسال و دریافت کنند و سوکت مشغول بشود و این امر باعث افت کیفیت و تأخیر در ارسال و دریافت محتوای ارتباط صوتی بشود و برای کاربر تجربه خوبی نباشد. در این برنامه برای هر سوکت دو لیست در نظر گرفته شده است که یکی برای اطلاعات دریافت شده و دیگری برای ارسال اطلاعات که در صف قرار خواهند گرفت استفاده می‌شود. و همینطور از یک نخ Thread برای دریافت اطلاعات و قرار دادن آن‌ها در لیست اطلاعات دریافتی استفاده می‌شود. برای ارسال اطلاعات از آن رو که بخش‌های مختلفی مثل مدیریت اتاق بازی (درخواست های پایه) را شامل می‌شود و همینطور مدل بازی هردو از لیست ارسال اطلاعات استفاده می‌کنند.

تصویر 13-3 کار کرد کلی Game Client

زمانی که Thread درحال برسی و ارسال اطلاعات است ممکن است یک درخواست بصورت کامل در لیست نوشته نشده باشد (به دلایل مختلف سخت افزاری یا طولانی بودن پردازش و متن درخواست) بنابر این اینجا مشکلی رخ خواهد داد که آن نخ اطلاعات نصفه را خوانده و دسترسی مجاز به سایر اطلاعات را ندارد که در این مورد دو حال ممکن است رخ دهد خطای Segment Fault و یا داده ناقص باشد و درخواستی که به سرور ارسال می‌شود بصورت ناقص باشد بنابر این نیاز است از Mutex استفاده کنیم.

روش کار بدین صورت می‌شود که Mutex شبیه به یک پرچم Flag است که زمانی که نیاز است اطلاعات را در لیست قرار بدهیم آن را قفل می‌کنیم و زمانی که می‌خواهیم اطلاعات را بخوانیم برسی می‌کنیم که قفل نباشد. با این روش می‌توان دسترسی بخش دیگر را از لیست مشترک گرفته و محدود کنیم.



تصویر 14-3 نمونه کد استفاده از Mutex

این نرم‌افزار مشابه به Main Client است یعنی از ماشین وضعیت استفاده می‌کند و وضعیت‌های مختلفی را دارد و در کنار آن مدل های بازی هم نیاز است در این قسمت پیاده‌سازی شوند تا کاربر بتواند درصورت دریافت کد پاسخ از سمت سرور که مربوط به مدل بازی می‌باشد را بصورت درست به نمایش بگذارد و کاربر نتیجه عمل خود را ببیند. بنابر این در یکی از وضعیت‌ها که Main Page نام دارد اشاره گر به مدل بازی قرار خواهیم داد و با استفاده از آن توابع مربوطه را زمانی که بازی اجرا شده باشد صدا خواهیم زد و این امر باعث می‌شود تا کد های مدل بازی در کنار این نرم‌افزار به اجرا دربیاییند.

در این بخش نکته این است که لیست درخواست و پاسخ‌های این بخش با Game Server یکی هستند و باید تمامی موارد مشابه هم باشند در غیر این صورت سرور و کاربر نمی‌توانند درخواست و پاسخ‌های یک دیگر را متوجه شوند.

زمانی که درخواست ها برسی می‌شود کد آن در یک Switch قرار می‌گیرد.

تصویر 15-3 کد برسی یک پاسخ از سوی سرور

درصورتی که کد جز موارد قابل پذیرش اتاق بازی نبود. وضعیت مدل بازی را برسی می‌کند و اگر بازی در حال اجرا و مدل بازی انتخاب شده بود آن کد را با صدا زدن تابع update مدل بازی ارسال می‌کند.

تصویر 16-3 کد ارسال پیام دریافت شده به مدل بازی

روند برسی درخواست به مدل بازی داده می‌شود و آن عمل‌های مورد نیاز را برای آن درخواست یا پاسخ اعمال می‌کند.

در این بخش تمامی مولفه ها مشابه Main Client هستند ولی ممکن است در برخی توابع تغییراتی ایجاد شده باشد ولی یک مولفه جدید و مخصوص اتاق بازی توسعه یافته نیاز است توضیحاتی در مورد آن داده شود:

جدول 13-3 مولفهی جدید برای Game Client

|  |  |  |  |
| --- | --- | --- | --- |
| **نام** | **کاربرد** | **تابع و**  **کاربرد آن‌ها** | **تأثیر گرفته از** |
| Player  List | زمانی که کاربری به اتاق وارد می‌شود برای نمایش اطلاعات همچون شناسه و نام کاربری و وضعیت‌های مدیر. ادمین و تماشاگر از این مولفه استفاده می‌شود. که شامل دو متن و سه تصویر و یک دکمه جهت تغییر وضعیت ارتباط صوتی می‌شود.  و تمامی موارد با استفاده از Padding نسبت به موقیت پدر موقیت دهی خواهند شد. | |  |  | | --- | --- | | وضعیت مدیر بودن را برای کاربر تغییر می‌دهد. | setOwner | | اطلاعات کاربر را پاک می‌کند | clear | | وضعیت ارتباط صوتی کاربر را فعال یا غیر فعال می‌کند | updateVoiceChatStatus | | مقدار نام کاربر را بازمیگرداند | getName | | تنظیم ابعاد دکمه | getId | | راه اندازی PlayerList با مقادیر اولیه و یا مقادیر داده شده. | init | | تنظیم نام برای کاربر | setName | | تابع سازنده که صورت‌های مختلفی دارد جهت استفاده های مختلف و مقادیر دریافتی را به تابع init ارسال خواهد کرد | PlayerList | | پنجره را بصورت رفرنس دریافت می‌کند و شئ های (شناسه و نام کاربری و وضعیت ارتباط صوتی) را ترسیم می‌کند و درصورت فعال بودن وضعیت مدیریت. ادمین. تماشاگر تصاویر آن‌ها را ترسیم خواهد کرد. | render | | بصورت مستقیم از  UiComponent  غیر مستیم از  Icon  و  Text  و  Picture  Button |

**نحوه توسعه یک مد بازی (Game Mode) در سمت کاربر (Game Client):**

در این بخش توضیحاتی درمورد چگونگی توسعه مد بازی برای سمت کاربر خواهیم داد. در ابتدا نیاز است توضیحاتی درمورد کلاس GameMode بدهیم.



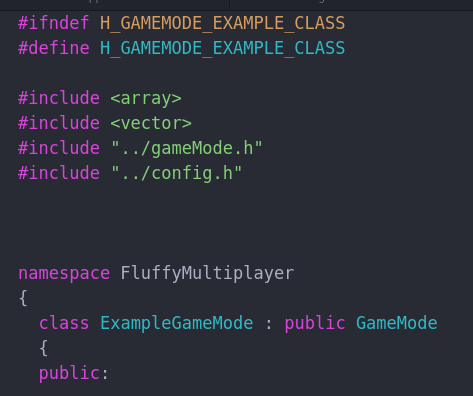
تصویر 17-3 کلاس GameMode در سمت کاربر

جدول 14-3 معرفی متغیر های GameMode در سمت کاربر

|  |  |
| --- | --- |
| **نام** | **کاربرد** |
| gameStarted | زمانی که بازی شروع شده و می‌خواهیم محتوایی که در تابع render قرار داده شده‌اند به نمایش گذاشته شوند مقدار این متغیر را True می‌کنیم. |
| gameModeId | این شناسه مد بازی است که سایر بخش‌های برنامه با استفاده از این شناسه مد بازی ما را میشناسند و برای تنظیم مد بازی از آن استفاده می‌کنند و باید غیر تکراری باشد. |

جدول 15-3 معرفی توابع GameMode در سمت کاربر

|  |  |  |
| --- | --- | --- |
| **نام تابع** | **کی اجرا می‌شود** | **کاربرد** |
| Update  Players  In  Game | زمانی که دستور شروع بازی داده می‌شود این تابع از مد بازی اجرا می‌شود و کاربر هایی که به اتاق بازی متصل شده‌اند را به عنوان ورودی به این تابع ارسال می‌کند و اطلاعاتی مثل (شناسه. نام کاربری. Index آن‌ها که به ترتیبی که متصل شده اند)‌ را بصورت پیوسته و با Delimiter تفکیک شده‌اند. | زمانی که این تابع اجرا شد لیست کاربران درون اتاق برای ما آماده می‌شود و می‌توان مقدار های بازیکن های درون بازی را با آن مقدار دهی کنیم. همچنین شناسه این کاربر جاری که در App ذخیره شده است را به ما میدهد. با استفاده از این شناسه می‌توانیم این سیستم که در حال اجرا بازی است را شناسایی کنیم که نام و شناسه آن چیست |
| update | زمانی که یک درخواست دریافت شود و کد درخواست از محدود اتاق بازی بیشتر باشد. این تابع از مد بازی اجرا می‌شود. | زمانی که این تابع اجرا شد اطلاعاتی همچون کد درخواست و داده ی درخواست را بصورت لیست به ما ارسال می‌کند و نیاز است برای دسترسی به مقادیر به اندیس های آن لیستی که دریافت شده است استفاده کنیم. همچنین می‌توان برای پردازش ابتدا کد دریافتی را در یک Switch قرار داد و کد های پاسخ‌های خود را به عنوان case قرار دهیم و درصورتی که وارد آن case شد دستورات ما را اجرا کند. |
| render | زمانی که متغیر gameStarted مقدارش True باشد و مد بازی انتخاب شده باشد و همچنین مدیر در صحفه تنظیمات نباشد. | برای ترسیم اشیاء مد بازی می‌توان از این تابع استفاده کرد. |
| eventHandle | زمانی که مد بازی انتخاب شده باشد بدون شرط دیگری برنامه این تابع راه صدا خواهد زد و مقادیر:  Window, Event, SendTextDataList, SendMutex  را به ما می‌دهد که به راحتی بتوانیم به رویداد ها و پنجره و لیست ارسال دسترسی داشته باشیم. | دریافت رویداد ها از سمت کاربر مثل کیبورد. ماوس. پنجره. دسته بازی و سایر event هایی که در SFML قابل استفاده اند. |
| Get  Game  Started | این تابع مربوط به App است زمانی که نیاز است با استفاده از این متغیر به مقدار آن متغیر gameStarted دسترسی خواهد داشت. | - |

ابتدا فایل مد بازی خود را مثل تصویر با نام مد بازی نام گذاری کنید و header guard و include gameMode.h را بنویسید و سپس مد بازی را کلاس GameMode بصورت عمومی ارث بدهید.

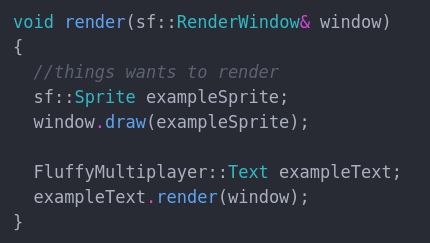
سپس در تابع سازنده خود یک شناسه غیر تکراری عددی صحیح به عنوان gameModeId قرار دهید.

تصویر 18-3 نوشتن مد بازی

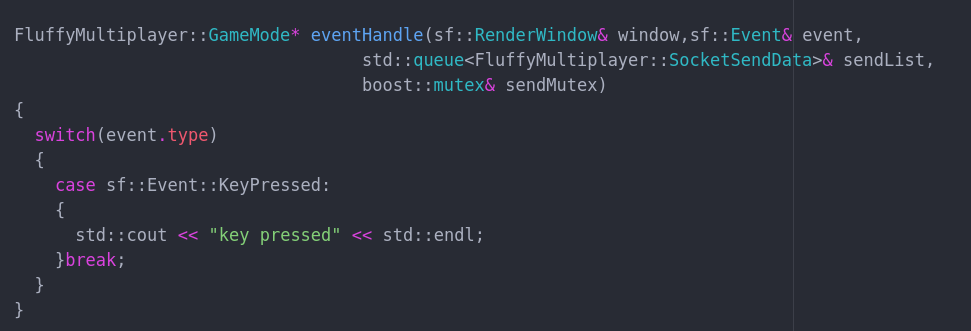
از توابع GameMode مثال‌هایی برای پیاده‌سازی کد نویسی شده است الگو برداری کنید و مد بازی خود را بنویسید:

تصویر 19-3 مثال چگونگی تعریف تابع updatePlayersInGame

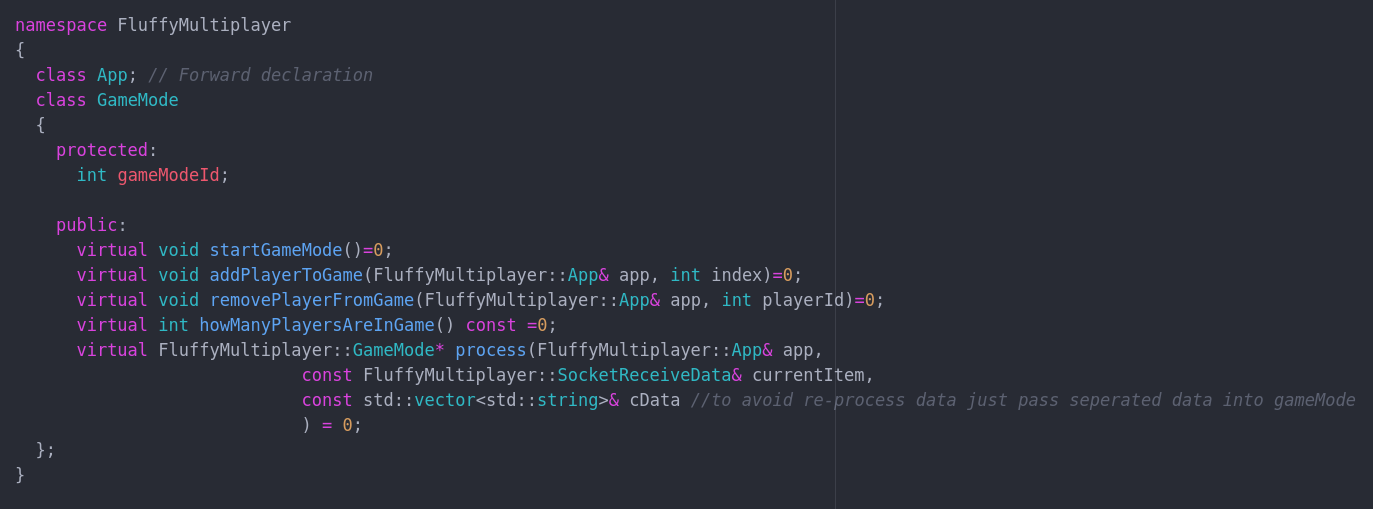
تصویر 20-3 مثال چگونگی تعریف تابع update



تصویر 21-3 مثال چگونگی تعریف تابع render

تصویر 22-3 مثال چگونگی تعریف تابع eventHandle

**چگونگی توسعه مد بازی (Game Mode) در سمت سرور:**

تصویر 24-3 کلاس GameMode در سمت سرور

جدول 16-3 معرفی متغیر های GameMode در سمت سرور

|  |  |
| --- | --- |
| **نام** | **کاربرد** |
| gameModeId | این شناسه مد بازی است که سایر بخش‌های برنامه با استفاده از این شناسه مد بازی ما را میشناسند و برای تنظیم مد بازی از آن استفاده می‌کنند و باید غیر تکراری باشد. |

جدول 17-3 معرفی توابع GameMode در سمت سرور

|  |  |  |
| --- | --- | --- |
| **نام تابع** | **کی اجرا می‌شود** | **کاربرد** |
| Start  Game  Mode | زمانی که دستور شروع بازی داده می‌شود این تابع از مد بازی اجرا می‌شود. | زمانی که تابع اجرا شد یعنی بازی قراره اجرا بشه و می‌توان init مولفه های بازی و سایر مقادیر در بازی را مقدار دهی کنیم.  برای مثال: تا زمانی که بازی شروع نشده تعداد کاربر ها معلوم نیستند یا لیست بازی شما بصورت array تعریف شده و نیاز است تمامی کاربران را داشته باشید تا بتوانید اقدام‌ها خود را انجام دهید |
| Add  Player  To  Game | زمانی که کاربری به اتاق بازی وصل می‌شود این تابع از مد بازی اجرا می‌شود. | می‌توان همان موقع اتصال آن را به بازیکن های مد بازی خود اضاف کنید. |
| Remove  Player  FromGame | زمانی که کاربری از اتاق بازی خارج می‌شود این تابع از مد بازی اجرا می‌شود. | می‌توان همان موقع قطع اتصال آن را از بازیکن های مد بازی خود حذف کنید. |
| process | زمانی که کد درخواست مربوط به اتاق بازی نباشد این تابع از مد بازی اجرا خواهد شد و مقدار های آن درخواست را به مد بازی انتقال خواهد داد. | اطلاع از درخواست های کاربرها و پردازش آن‌ها و ارسال پاسخ به آن‌ها. توجه داشته باشید که شئ app بصورت مرجع انتقال داده شده پس می‌توان به لیست ارسال و سایر دسترسی کامل داشت. |
| How  Many  Players  Are  In  Game | زمانی که بازی می‌خواهد شروع شود این تابع اجر خواهد شد و یک عدد را از مد بازی می‌گیرد جهت برسی تعداد کاربران درون بازی. | متغیری تعریف می‌کنیم با عنوان تعداد کاربر های متصل شده و تعداد کاربران را در آن قرار می‌دهیم بنا بر این زمانی که تعداد کاربر ها به تعداد ۴ نفر نرسیده مد بازی شروع نخواهد شد و به کاربر پیغام کاربران بیشتری نیاز است نمایش داده می‌شود. |



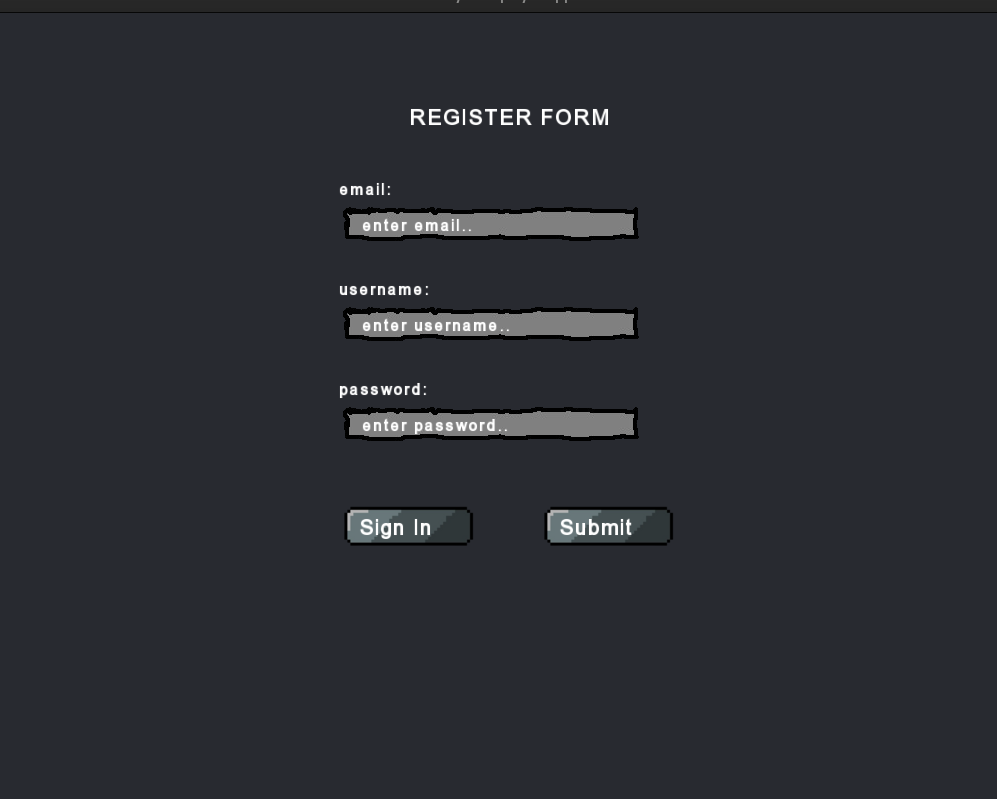
تصویر 25-3 مثال چگونگی تعریف مد بازی در سمت سرور

همجنین سایر متد ها مثل مثال‌های بالا که در سمت کاربر توسعه داده شد اینجا هم با نگاه کردن به صورت و توابع کلاس GameMode مد بازی خود را کامل کنید.

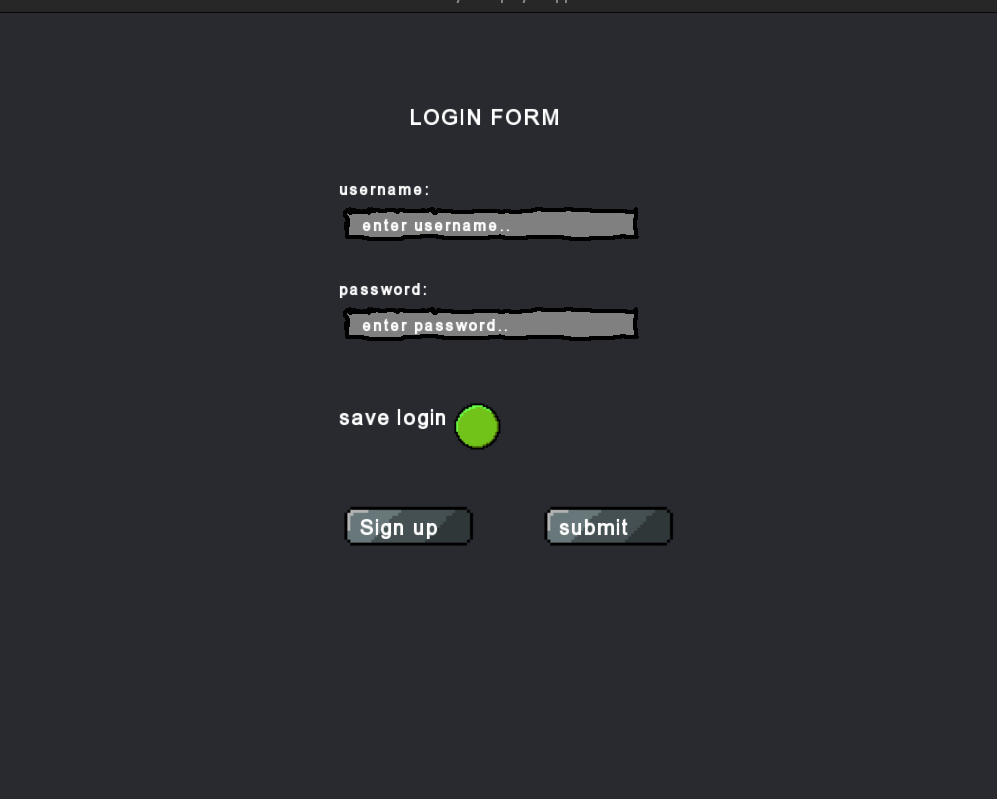
**فصل چهارم**

**نتیجه‌گیری**

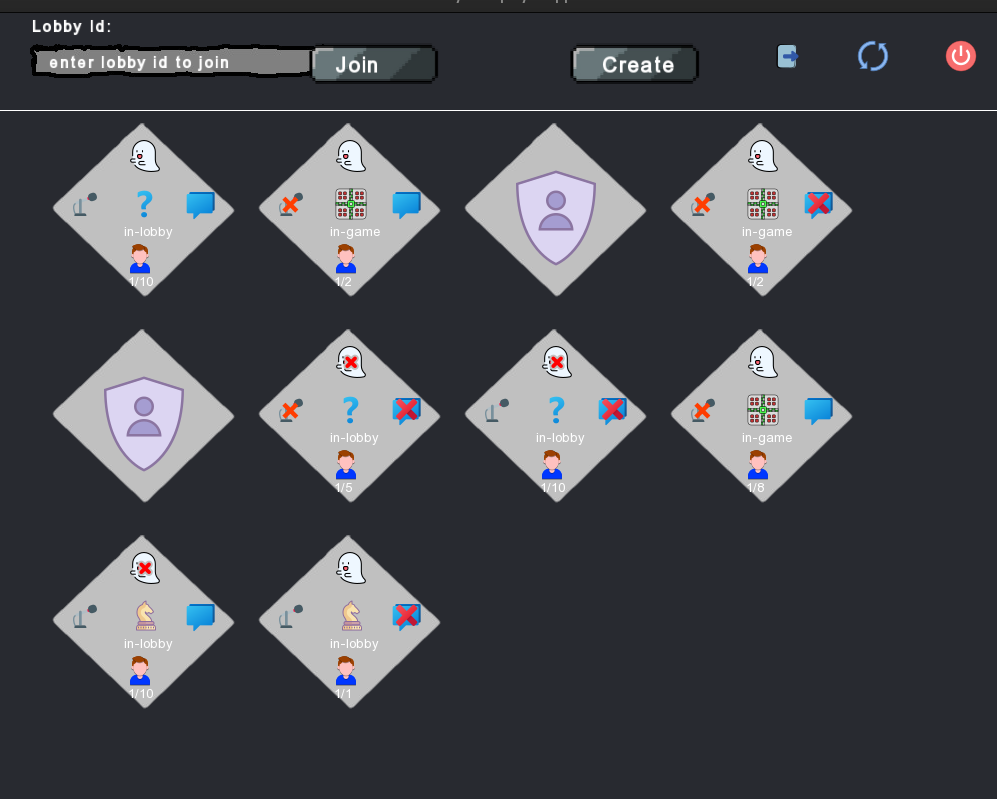
برای توسعه برنامه و بازی با این روش توسعه آن بسیار کند و سخت خواهد بود و همچنین از کیفیت مناسبی نخواهد داشت بنابر این بهتر است از فریمورک ها و متورهای بازی سازی و ابزارهای پیشرفته و مدرن برای توسعه استفاده کرد. تصاویری از برنامه پیاده شده:



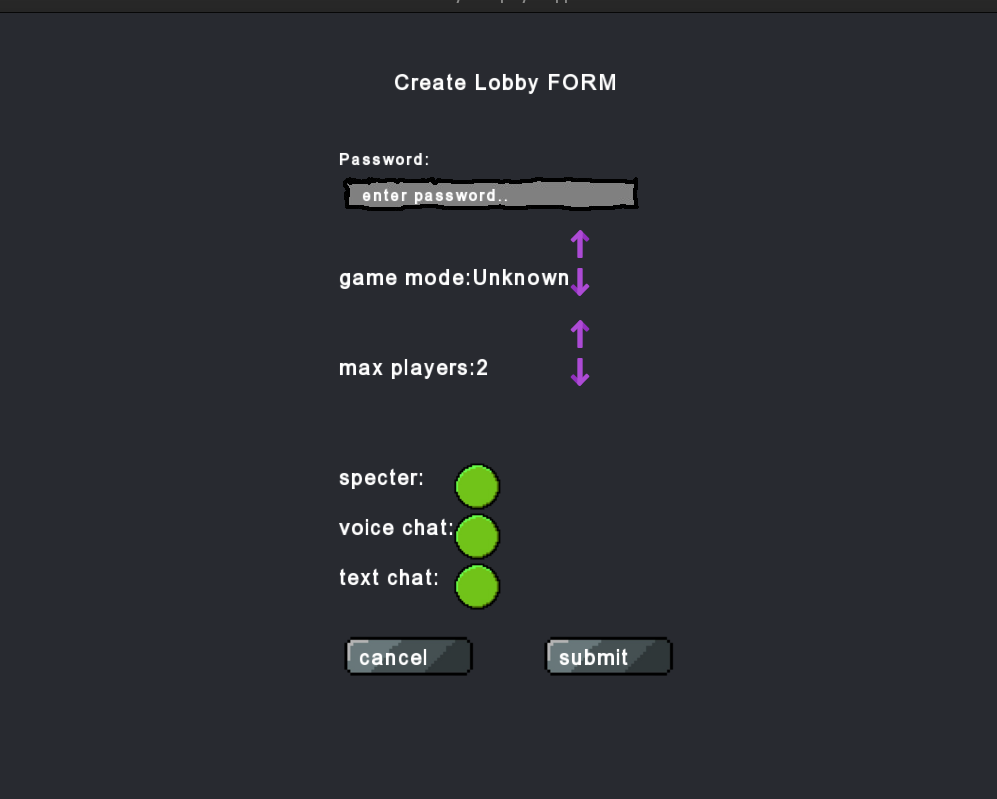
تصویر 1-4 فرم ساخت حساب کاربری Main Client



تصویر 2-4 فرم ورود به حساب کاربری Main Client

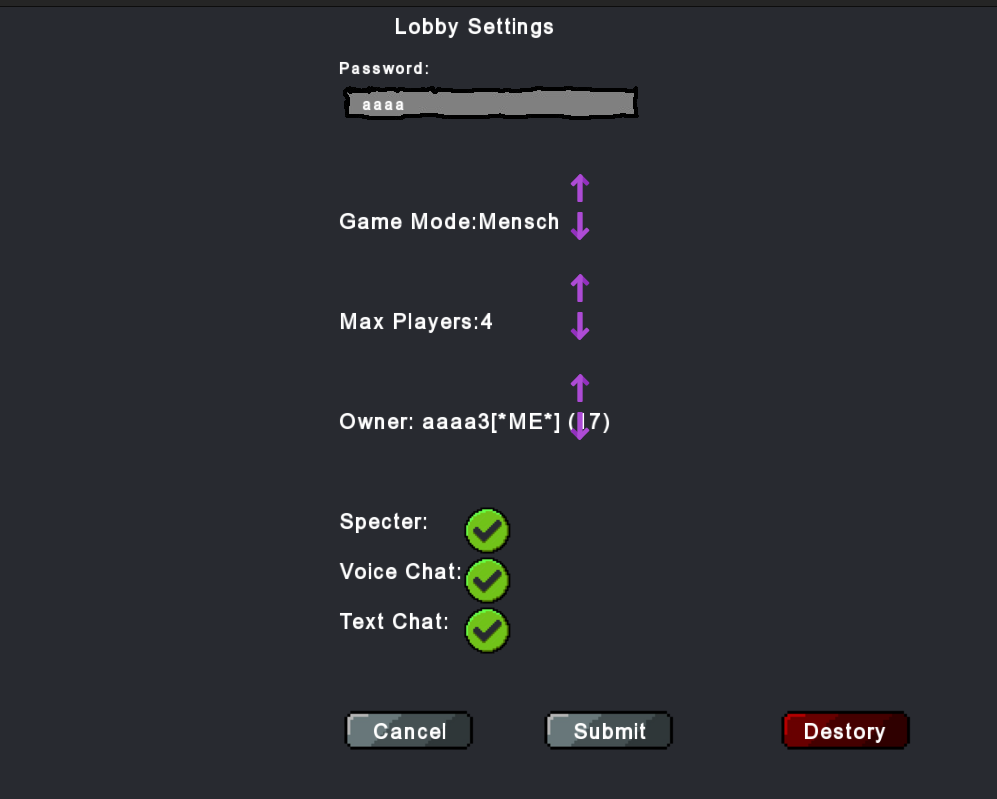


تصویر 3-4 صفحه اصلی Main Client



تصویر 4-4 فرم ساخت اتاق بازی Main Client

تصویر 5-4 صفحه اصلی Game Client (بازی منچ در حال اجرا)



تصویر 6-4 فرم ویرایش اطلاعات اتاق بازی

In the name of Allah



Islamic Republic of Iran

Ministry of Science. Research and technology

Technical University

Shahid Bahonar Technical and Engineering Faculty, Shiraz

**Software for running and managing multiplayer games over the network**

Bachelor thesis

**Field**

Professional computer software engineering

**Supervisor**

Mr. Engineer Hasan Baseri

**Prepared**

Mohammad Salehinejad