

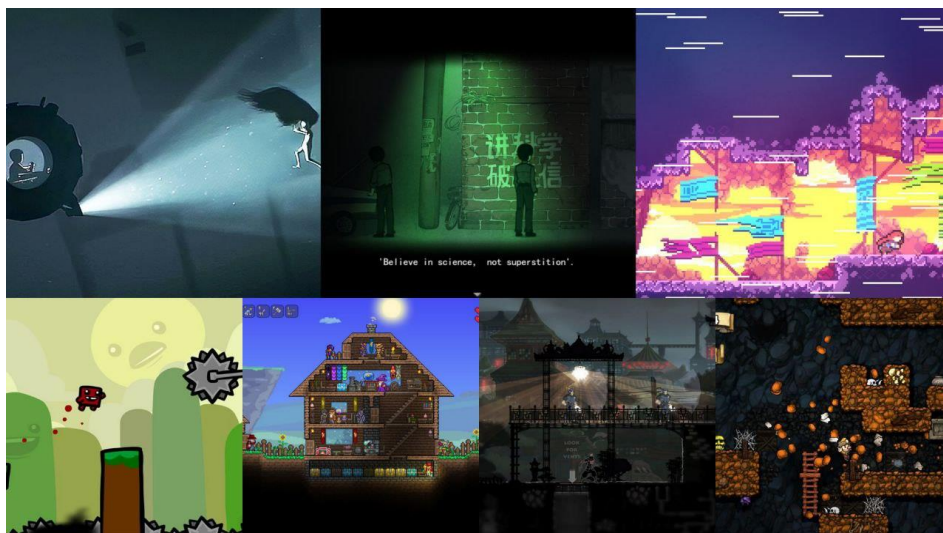
فصل اول:

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 ۱-۱ 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 ارائه 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 یوبلی نرم‌افزارهایی هستند که امکان دسترسی به بازی‌های تحت شبکه را فراهم می‌کنند.



تصویر ۱-۲ نرم‌افزارهای 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/بازی‌های 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 نویسی 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 ۱-۳ 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 عامل‌های 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 ۸۰۰ پیکسل و ارتفاع ۶۰۰ پیکسل ایجاد کنید:

```
sf::RenderWindow window(sf::VideoMode(800, 600), "SFML Window");
```

تصویر ۱-۴ 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 ۱۰۰×۲۰۰ پیکسل را رسم کنید:

```
sf::RectangleShape rectangle(sf::Vector2f(200, 100));
rectangle.setFillColor(sf::Color::Red);
window.draw(rectangle);
```

تصویر ۵-۱ 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:

```
sf::SoundBuffer buffer;
buffer.loadFromFile("sound.wav");
sf::Sound sound(buffer);
sound.play();
```

تصویر ۶-۱ 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:

```
sf::Event event;
while (window.pollEvent(event))
{
    if (event.type == sf::Event::KeyPressed && event.key.code == sf::Keyboard::Escape)
    {
        window.close();
    }
}
```

تصویر ۷-۱ کد کنترل ورودی کاربر و رویدادها

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:

```
sf::Texture texture;
texture.loadFromFile("animation.png");

sf::Sprite sprite(texture);
sprite.setTextureRect(sf::IntRect(0, 0, 32, 32));

while (window.isOpen())
{
    // بارگیری فریم‌های جدید و تغییر موقعیت
    // ...

    window.clear();
    window.draw(sprite);
    window.display();
}
```

تصویر ۸-۱ کد انیمیشن و روح بازی

این معرفی شما را با برخی از 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:

```
sf::Sprite sprite(texture);
sf::FloatRect bounds = sprite.getGlobalBounds();
```

تصویر ۹-۱ کد دریافت موقعیت‌های 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:

```
sf::Sprite sprite(texture);
sf::Vector2i mousePosition = sf::Mouse::getPosition(window);

if (sprite.getGlobalBounds().contains(sf::Vector2f(mousePosition)))
{
    // قرار دارد sprite نقطه ماوس درون
}
```

تصویر ۱۰-۱ کد بررسی 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:

```
sf::FloatRect rect(100, 100, 200, 100);
float x = rect.left;      // مستطیل x موقعیت
float y = rect.top;       // مستطیل y موقعیت
float width = rect.width; // عرض مستطیل
float height = rect.height; // ارتفاع مستطیل
```

تصویر ۱۱-۱ 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:

```
sf::Font font;
font.loadFromFile("arial.ttf"); // بارگیری فایل فونت

sf::Text text;
text.setFont(font);
text.setString("Hello, SFML!");
text.setCharacterSize(24);
text.setFillColor(sf::Color::Red);
text.setPosition(100, 100);
```

تصویر ۱۲-۱ 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:

```
sf::RectangleShape rectangle(sf::Vector2f(200, 100));  
rectangle.setFillColor(sf::Color(255, 0, 0)); // رنگ قرمز
```

تصویر ۱۳-۱ 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:

```
sf::UdpSocket socket;  
sf::IpAddress recipient = "192.168.1.100";  
unsigned short port = 5000;  
  
// ارسال پیام  
std::string message = "Hello, SFML!";  
socket.send(message.c_str(), message.size() + 1, recipient, port);  
  
// دریافت پیام  
char buffer[1024];  
std::size_t received;  
sf::IpAddress sender;  
unsigned short senderPort;  
socket.receive(buffer, sizeof(buffer), received, sender, senderPort);
```

تصویر ۱۴-۱ 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 مقدارهای 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 گرفته

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:

```
#include <SFML/Graphics.hpp>

int main()
{
    sf::RenderWindow window(sf::VideoMode(800, 600), "SFML Window");

    // ایجاد یک شیء مستطیل
    sf::RectangleShape rectangle(sf::Vector2f(100, 100));
    rectangle.setFillColor(sf::Color::Red);
    rectangle.setPosition(200, 200);

    while (window.isOpen())
    {
        sf::Event event;
        while (window.pollEvent(event))
        {
            if (event.type == sf::Event::Closed)
                window.close();
        }

        window.clear();

        // رسم شیء مستطیل در پنجره
        window.draw(rectangle);

        window.display();
    }

    return 0;
}
```

تصویر ۱-۱۵ نمونه کد یک برنامه ساده با استفاده از کتابخانه SFML

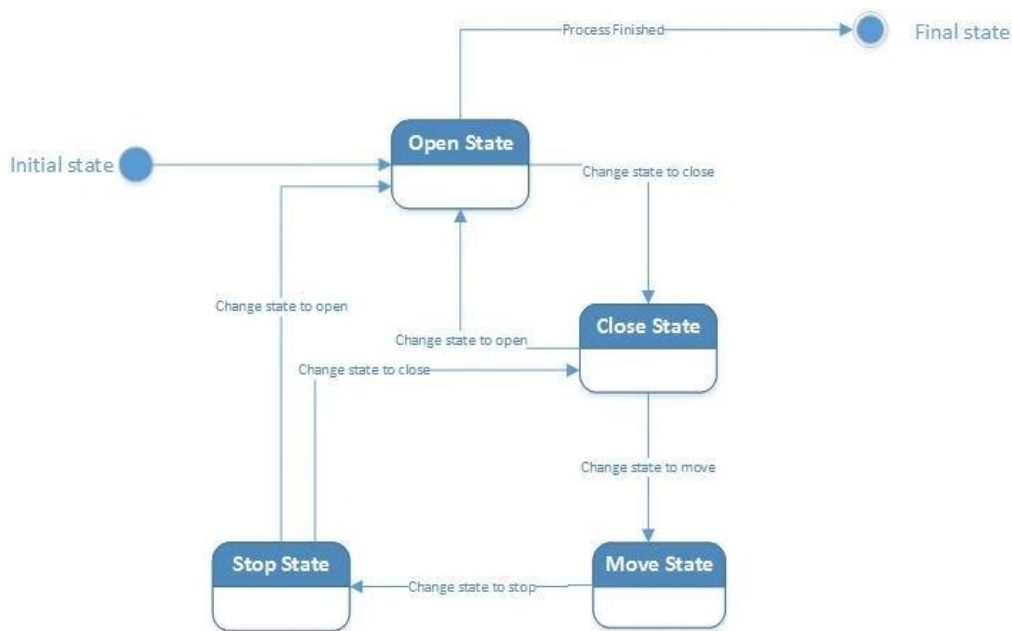
در این کد، یک شیء از کلاس `sf::RenderWindow` با سایز ۸۰۰×۶۰۰ پیکسل و عنوان "SFML Window" ایجاد می‌شود. سپس یک شیء مستطیل با ابعاد ۱۰۰×۱۰۰ پیکسل و رنگ قرمز ایجاد می‌شود و در موقعیت (۲۰۰، ۲۰۰) در پنجره قرار می‌گیرد.

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 مهندسين نرم‌افزار با تجربه توليد شده و ميتوان براي حل مشکلات از آن‌ها در پروژه‌های مختلف استفاده کرد.

بنابر این مهم نیست با چه زبانی کد نویسی می‌کنید می‌توانید از این الگوها 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 مجدد در سایر پروژه ها و نیاز ها را دارد.



تصویر ۱۶- ۱ 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 ۱۷- ۱ 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.

```
#include <iostream>
#include <boost/asio.hpp>

using boost::asio::ip::udp;

int main() {
    try {
        boost::asio::io_context io_context;

        // ساخت یک سوکت UDP
        udp::socket socket(io_context, udp::endpoint(udp::v4(), 0));

        std::string server_ip = "127.0.0.1";
        unsigned short server_port = 5000;

        // تنظیم آدرس سرور
        udp::endpoint server_endpoint(boost::asio::ip::make_address(server_ip), server_port);

        std::string message = "Hello, server!";
        std::cout << "Sending message to server: " << message << std::endl;

        // ارسال پیام به سرور
        socket.send_to(boost::asio::buffer(message), server_endpoint);

        char receive_buffer[1024];
        udp::endpoint sender_endpoint;

        // دریافت پاسخ از سرور
        size_t length = socket.receive_from(boost::asio::buffer(receive_buffer), sender_endpoint);

        std::cout << "Received response from server: ";
        std::cout.write(receive_buffer, length);
        std::cout << std::endl;
    } catch (std::exception& e) {
        std::cerr << "Exception: " << e.what() << std::endl;
    }

    return 0;
}
```

تصویر ۱۸-۱ کد نمونه ارسال داده به شبکه

انگیزه:

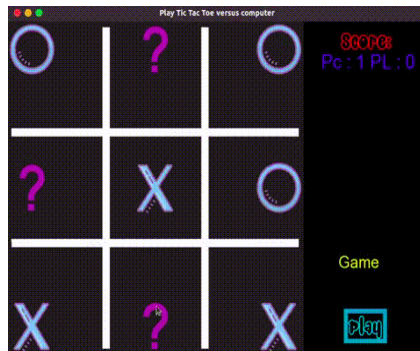
بعد از توضیحات بالا شاید سؤال برای شما سؤال پیش بیاید که چرا از یک فریم ورک یا موتور بازی سازی مناسب برای ایجاد پروژه استفاده نکرده‌ایم. هدف از انتخاب این مسیر این بوده که یک نرم‌افزار تک صفحه‌ای با استفاده از Game Design Pattern ها تولید کنیم و همچنین با انجام این پروژه به یادگیری و کسب تجربه عمیق‌تری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

فصل دوم

کارهای مرتبط و ابزارهای مورد نیاز

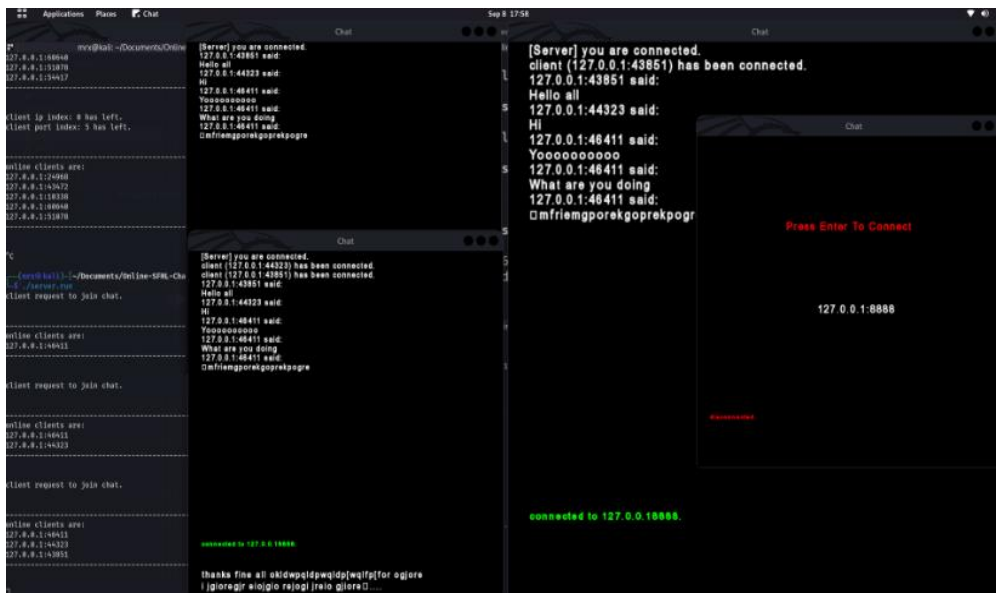
کارهای مشابه که قبلاً صورت گرفته:

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 ۱-۲ بازی دوز

و به یک برنامه چت ساده تحت شبکه:



تصویر ۲-۲ برنامه چت آنلاین

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



تصویر ۳-۲ بازی بازو کای کوچک



تصویر ۲-۴ سیستم عامل لینوکس

سیستم عامل:

برای توسعه نرم افزار حرفه ای سیستم عامل های مختلفی وجود دارند. اما یکی از آن ها لینوکس است و تقریباً تمامی سیستم عامل ها قابلیت های یکسان و قابل مقایسه ای دارند و در صورت 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 کرده ایم به دلیل اینکه نیاز به کرک نرم افزار ها نیست و همچنین راحت شدن از آپدیت های و سرویس های ویندوز و استفاده از ابزار ها بدون نیاز به شبیه سازها.



تصویر ۲-۵ مدیریت نسخه گیتهاب

مدیریت نسخه:

برای اینکه پروژه بخش های مختلفی دارد و همینطور تعداد فایل ها کد زیاد میباشد و میخواهیم یک توسعه حرفه ای را تجربه کنیم. نیازمند یک برنامه مدیریت نسخه (Version Controller) هستیم ابزار های مدیریت نسخه نرم افزار قابلیت های زیر را به ما ارایه می کند:

- 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 در صورت بروز مشکل به آن تغییرات دسترسی داشته باشیم.
- در صورتی که پروژه بصورت تیمی و چند نفره در حال توسعه باشد میتوان از این ابزار برای هماهنگی توسعه اعضای تیم استفاده کرد.
- ذخیره تغییرات با عنوان مناسب میتواند کمک بسیار زیادی در دسترسی به تغییرات آن کند.
- تگ و نسخه زدن به بخشی از کد.
- بازگشت به تغییرات قبل در صورت بروز مشکل.
- بررسی 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 "۱.۰.۰.۰" | 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 ۶-۲ 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 همیشه گفت این برنامه حرفه‌ای برای کد نویسی نیست مخصوصاً اگر زبان برنامه نویسی و محیط مربوط به وب نباشد. زیرا 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 ۷-۲ 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):



تصویر ۸-۲ دیباگر

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.

برخی از محیط‌های توسعه نرم‌افزار مجهز به دیباگر هستند ولی دیباگرهای 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 دریاور تا وقتی که دوباره به آن نقطه مورد نظر برسد:

```
> continue
```

چاپ مقدار یک متغیر در دسترس (یعنی در خط‌های قبل که اجرا شده بودند این متغیر 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 ۹-۲ کتابخانه گرافیکی

نصب کتابخانه Boost:

برای توسعه نیاز به نصب این کتابخانه است
و برای نصب آن می‌توان از دستور زیر استفاده کرد:



تصویر ۱۰-۲ کتابخانه Boost

```
sudo apt-get install libboost-all-dev
```

نصب کتابخانه پایگاه داده SQLite:

برای توسعه نیاز به نصب این کتابخانه است
و برای نصب آن می‌توان از دستور زیر استفاده کرد:



تصویر ۱۱-۲ کتابخانه SQLite

```
sudo apt-get install sqlite3
```

فصل سوم

جزئیات روش و کارهای انجام شده

چگونگی خروجی گرفتن پروژه:

با استفاده از bash script که نوشته‌ایم تمامی فایل‌ها و پوشه‌های 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 عدد‌های زیر به حال مورد نظر وارد می‌شویم:

۱. کامپایل ساده: فقط برنامه را کامپایل می‌کند.
۲. کامپایل برای دیباگر: با فعال کردن پرچم یا flag دیباگ کامپایل می‌کند.
۳. اجرای برنامه.
۴. اجرای برنامه در دیباگر.
۵. اطلاعاتی را می‌گیرد و سپس برنامه را اجرا می‌کند ولی با ارسال آن اطلاعات به عنوان ورودی یا Argument.

```
#!/bin/bash
compileMain="main.cpp ./include/*.h ./include/gameModes/*.h ./src/*.cpp ./src/gameModes/*.cpp -o gs.run -std=c++11
-lboost_regex -lboost_system -lboost_filesystem -lboost_thread -lsqlite3"
PS3="Select compile type: "
select compileMode in normal debug run runViaGDB runViaGDBGargs
do
    case $REPLY in
        1)
            #compile whole program
            echo -e "\n\e[31m[START]\e[39m compile $compileMode started at: `date`"
            # rm fluffy.db
            g++ $compileMain
            rm *.o
            rm *.gch
            spd-say 'ready to run'
            echo -e "\n\e[32m[END]\e[39m compile $compileMode finished at: `date`"
            ;;
        2)
            #compile whole program for gdb
            echo -e "\n\e[31m[START]\e[39m compile $compileMode started at: `date`"
            # rm fluffy.db
            g++ -g $compileMain
            rm *.o
            rm *.gch
            spd-say 'ready to run'
            echo -e "\n\e[32m[END]\e[39m compile $compileMode finished at: `date`"
            ;;
        3)
            ./gs.run
            break
            ;;
        4)
            gdb ./gs.run
            break
            ;;
        5)
            read -p "Enter value id maxPlayers gameMode currentPlayers
voiceStatus textStatus specterStatus inGameStatus password
textPort voicePort ownerId: " lid maxplayers gamemode currentplayers voicestatus
textstatus specterstatus ingamestatus password textport voiceport ownerid

            gdb --args ./gs.run lid maxplayers gamemode currentplayers voicestatus textStatus
specterstatus ingamestatus password textport voiceport ownerid
            ;;
        *)
            echo "incorrect index."
            ;;
    esac
done
```

تصویر ۳-۰ کد اسکریپت کامپایل برنامه

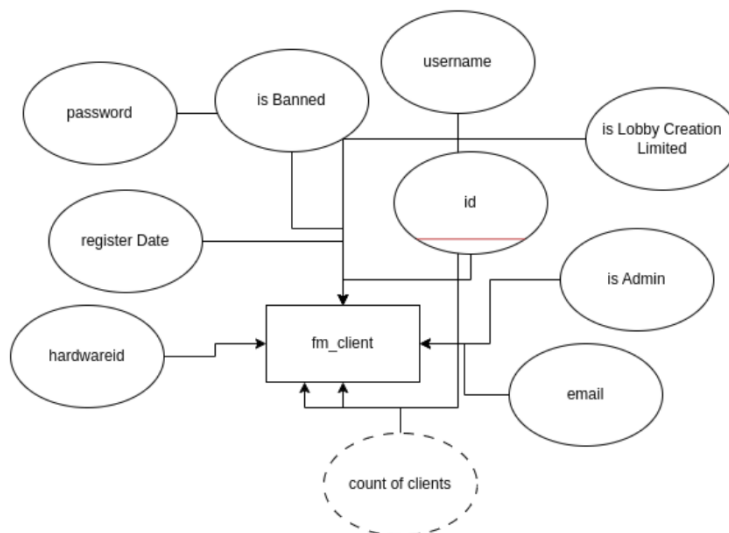
پایگاه داده و نحوه ذخیره اطلاعات:

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

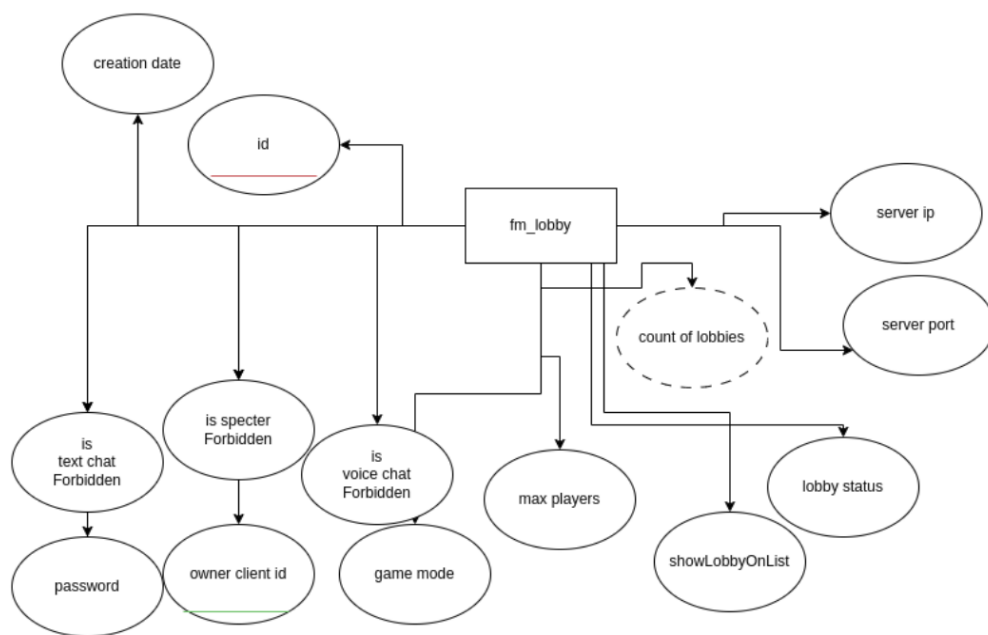
جدول ۳-۱ جدول های پایگاه داده

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

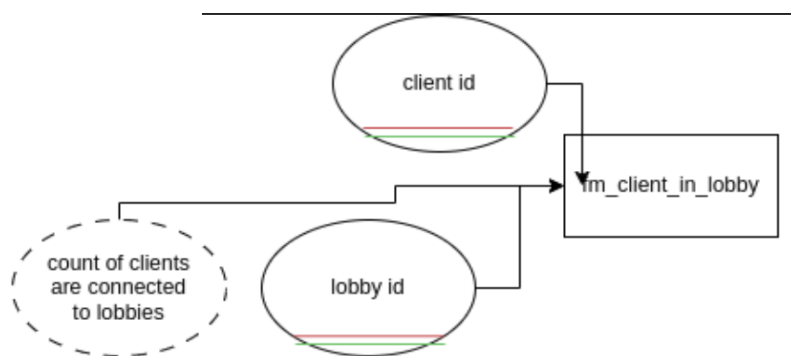
همچنین جزئیات جداول در تصاویر زیر آمده است همچنین می‌توانید در پروژه در پوشه docs به پوشه database رفته و اطلاعات کامل تری از پایگاه داده کسب کنید.



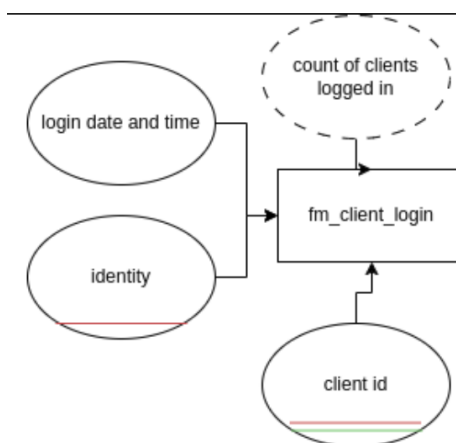
تصویر ۳-۱ پایگاه داده جدول حساب کاربر ها



تصویر ۳-۲ پایگاه داده جدول اتاق‌های بازی



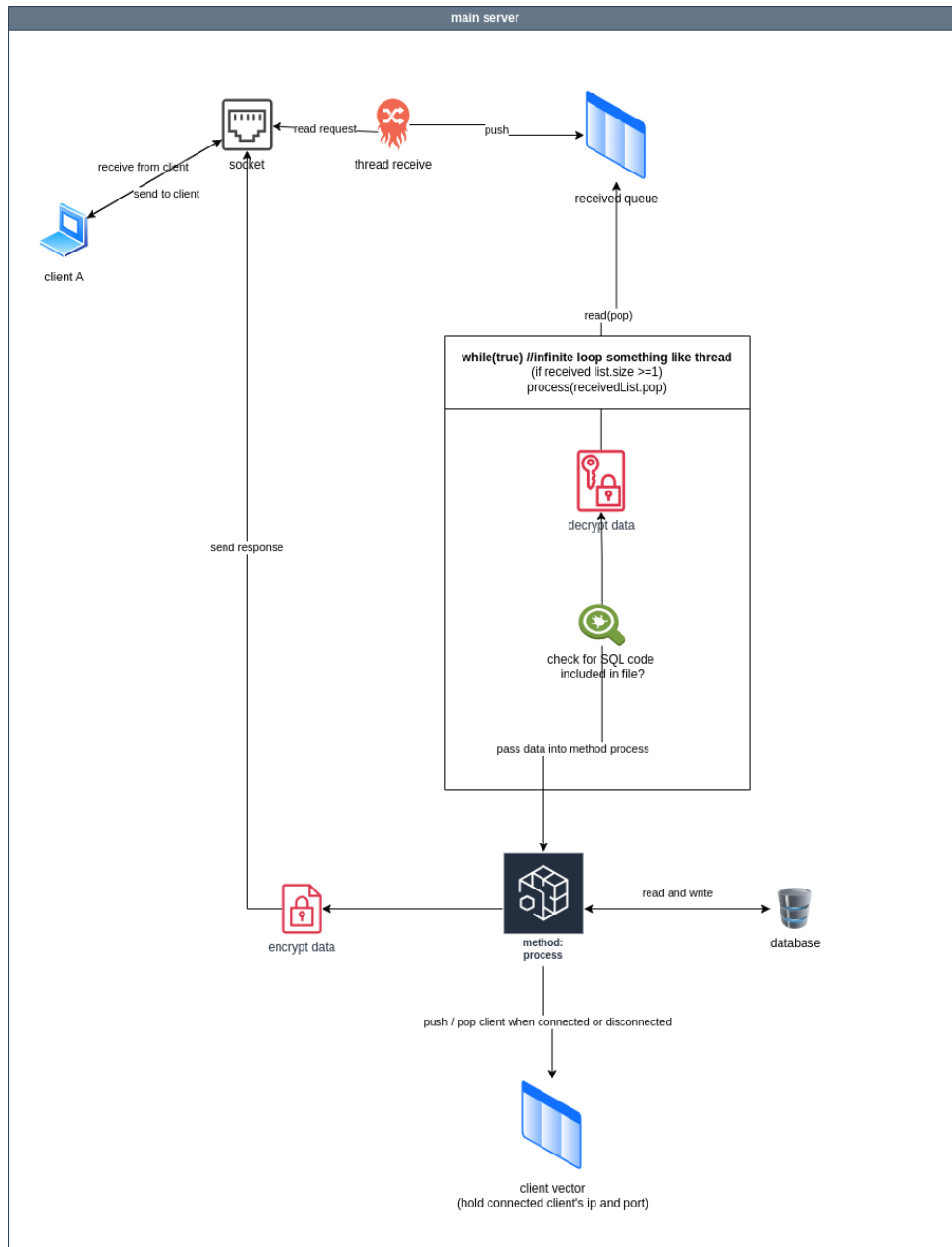
تصویر ۳-۳ پایگاه داده جدول کاربر به کدام اتاق متصل شده



تصویر ۳-۴ 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 ۳-۵ نحوه کار کرد کلی برنامه Main Server

(در صورتی 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 قابلیت 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 می‌خواهیم 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 در صورتی 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:

۱. ثبت حساب کاربری.
۲. ورود به حساب کاربری.
۳. ساخت اتاق بازی.
۴. ارسال مشخصات اتاق بازی به کاربر (توسط شناسه ای که کاربر ارسال کرده است).
۵. ارسال لیست اتاق بازی به کاربر.
۶. ورود به حساب کاربری بدون نیاز به مجدد وارد کردن مشخصات حساب کاربری (در صورتی 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: در صورتی 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 بررسی و حذف کد های مخرب sql دریافت شده.	امن نگه داشتن اطلاعات و بررسی اطلاعات دریافتی توسط سرور و موارد امنیتی.

	<p>تابع تولید شناسه یکتا (identity) که جهت احراز هویت کاربر استفاده شده.</p> <p>تابع بررسی سختی و آسانی کلمه عبور وارد شده توسط کاربر.</p>	
<p>نگهداری و جمع آوری داده‌ها و نوع‌های پر تکرار و مورد استفاده توسط فایل‌های متفاوت.</p>	<p>در پروژه نوع‌ها (Type) های داده‌ای زیاد خواهند بود و در صورتی 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 نوع‌ها 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).</p>	DataType.h
<p>پردازش و پاسخ دهی به درخواست/اطلاعات دریافتی از سوی کاربران.</p>	<p>بررسی 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.</p>	ProcessData.h
<p>نگهداری کد های درخواست و پاسخ با نام خوانا.</p>	<p>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 نام‌های 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.</p>	Server-request-response-list.h
<p>تابع دریافت اطلاعات همراه با Thread آن</p> <p>شی پایگاه داده</p> <p>لیست اطلاعات دریافت شده</p>	<p>اجرای همزمان توابع و متدها در کنار یکدیگر بصورتی که با لیست موجود در تعامل باشند.</p> <p>(به اشتراک گذاشتن یا پاس دادن منابعی همچون لیست دریافتی ها و پایگاه داده به</p>	Main.cpp

تابع receiveData و شی process)	صدا زدن تابع process از کلاس ProcessData	
--------------------------------	---	--

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 بررسی می‌کند در صورتی که فایل پایگاه داده‌ای در کنار آن موجود نباشد. ابتدا تلاش می‌کند و فایل پایگاه داده را می‌سازد و سپس جدول‌های 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 شی از نوع ProcessData تولید می‌کند و در یک حلقه بی‌نهایت متد process آن شی 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 در صورتی که شرط‌های مورد نظر (بررسی اندازه اطلاعات دریافتی) صحت یافت اطلاعات را در قالب نوع داده‌ای SocketDataQueue قرار می‌دهد و آن را در لیست اطلاعات دریافتی اضافه 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 کد‌ها تعریف شده‌اند.
boost::asio::ip::address	ip	آدرس آی‌پی 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 جداکننده‌ها (Delimiters) جدا خواهند شد.

همچنین همزمان آن متد process در حال بررسی اندازه لیست اطلاعات دریافتی است در صورتی که اندازه آن بیشتر از صفر شد آن اطلاعات را در بررسی قرار می دهد. اطلاعات دریافتی را رمز گشایی می کند سپس امنیت اطلاعات را توسط تابع isSQLCodeIncluded بررسی می کند در صورت 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 بررسی قرار می دهد.

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 می کنیم.

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

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 بخش هایی از نام حذف شده اند. برای مثال عبارت MS به معنای MainServer و MC به معنای MainClient می باشد که از نام حذف شده اند همچنین خط های زیر “ _ ” هم حذف شده اند.

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

کد های درخواست و پاسخ ها 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 پاسخ ها به همراه توضیحات

نوع	کد	نام	توضیح	اطلاعات دریافتی/ارسالی
۱	۱۰۰	REQUEST CONNECT	درخواست متصل شدن به سرور	-
۱	۱۰۱	REQUEST DISCONNECT	درخواست قطع اتصال از سرور	-
۱	۱۰۲	REQUEST LOGIN	درخواست ورود به حساب کاربری	نام کاربری کلمه عبور شناسه سخت افزار
۱	۱۰۳	REQUEST REGISTER	درخواست ثبت حساب کاربری جدید	ایمیل نام کاربری کلمه عبور شناسه سخت افزار
۱	۱۰۴	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
۱	۱۰۵	REQUEST GET LOBBY LIST	درخواست دریافت لیست اتاق های بازی	-
۱	۱۰۶	REQUEST GET LOBBY INFO	درخواست دریافت اطلاعات اتاق بازی توسط شناسه آن	شناسه اتاق بازی
۱	۱۰۷	REQUEST LATEST VERSION	درخواست دریافت ورژن جاری سرور	-
۱	۱۰۸	REQUEST RELOGIN	درخواست ورود به حساب کاربری بدون وارد کردن نام کاربری و کلمه عبور	Identity شناسه سخت افزار
۲	۲۰۰	RESPONSE CONNECTION ACCEPTED	موفقیت در اتصال به سرور	-

-	موفقیت در قطع اتصال از سرور	RESPONSE DISCONNECTED	۲۰۱	۲
identity	ورود با موفقیت انجام شده است	RESPONSE SUCCESS LOGIN	۲۰۲	۲
identity	ثبت نام با موفقیت انجام شده است.	RESPONSE SUCCESS REGISTER	۲۰۳	۲
آدرس سرور : آدرس پورت سروری که ساخته شده	ساخت اتاق بازی با موفقیت انجام شده است.	RESPONSE SUCCESS LOBBY CREATED	۲۰۴	۲
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_COUN T_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.	درخواست دریافت لیست اتاق‌های بازی با موفقیت انجام شده است.	RESPONSE SUCCESS GET LOBBY LIST	۲۰۵	۲

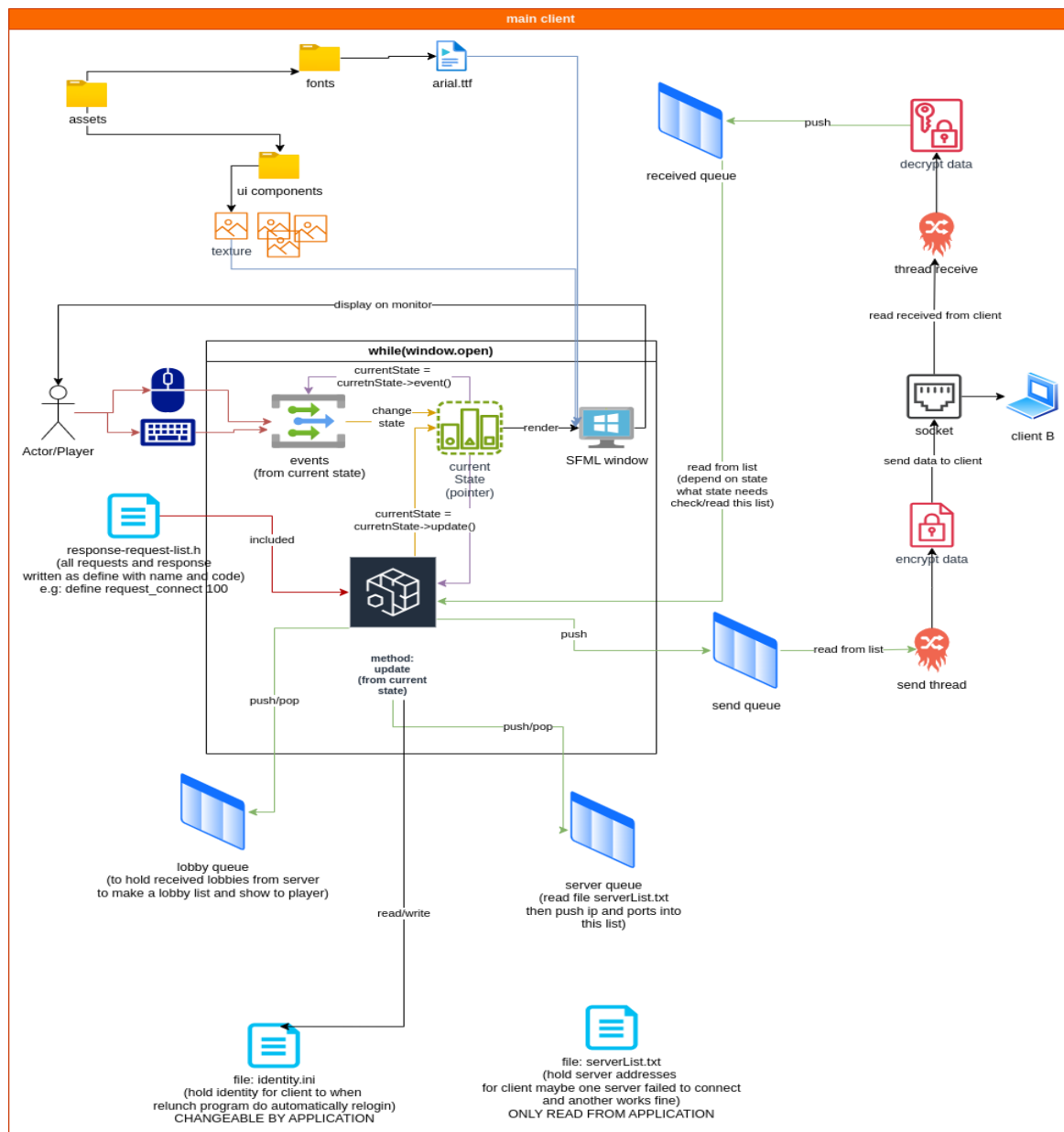
تعداد کاربر های جاری در اتاق				
مدل بازی				
آدرس سرور (آدرس و پورت یک جا با عبارت : جدا شده اند)				
اطلاعات آن اتاق را بصورت پشت سر هم با استفاده از تفکیک کننده جدا شده. ترتیب اطلاعات مثل RESPONSE SUCCESS GET LOBBY LIST می باشد	با دریافت اطلاعات اتاق توسط شناسه اتاق با موفقیت انجام شده است.	RESPONSE SUCCESS GET LOBBY INFO	۲۰۶	۲
ورژن سرور را از config.h با نام : MS_VERSION	پاسخ موفقیت دریافت عدد آخرین نسخه برنامه	RESPONSE SUCCESS LATEST VERSION	۲۰۷	۲
-	خطای اتصالی با این آدرس و پورت وجود دارد.	ERROR CONNECTION EXISTS	۴۰۰	۳
-	خطای اتصال رد شده است به دلیل استفاده از فیلتر شکن	ERROR CONNECTION REJECTED USING VPN	۴۰۱	۳
-	خطای اتصال رد شده است.	ERROR CONNECTION REJECTED	۴۰۲	۳
-	خطای اتصالی با این آدرس و پورت یافت نشد.	ERROR CONNECTION NOT EXISTS	۴۰۳	۳
-	خطا اطلاعات ورود به حساب کاربری درست نیست.	ERROR FAILED TO LOGIN INCORRECT	۴۰۴	۳
-	خطا حساب کاربری وجود ندارد	ERROR FAILED TO LOGIN NOT EXISTS	۴۰۵	۳
-	خطا حساب کاربری ممنوع شده است.	ERROR_FAILED_TO_LOGIN_BANNED	۴۰۶	۳

-	خطا ساختار درخواست ورود به حساب کاربری مشکل دارد.	ERROR_FAILED_TO_LOGIN_BAD_DATA_SYNTAX	۴۰۷	۳
-	خطا در ثبت نام حساب کاربری: نام کاربری وجود دارد.	ERROR_FAILED_TO_REGISTER_USERNAME_EXISTS	۴۰۸	۳
-	خطا در ثبت نام حساب کاربری: پست الکترونیکی وارد شده قبلاً ثبت شده.	ERROR_FAILED_TO_REGISTER_EMAIL_EXISTS	۴۰۹	۳
-	خطا در ثبت نام حساب کاربری: کلمه عبور وارد شده آسان است	ERROR_FAILED_TO_REGISTER_EASY_PASSWORD	۴۱۰	۳
-	خطا ساختار درخواست ساخت حساب کاربری مشکل دارد.	MS_ERROR_FAILED_TO_REGISTER_BAD_DATA_SYNTAX	۴۱۱	۳
-	خطا در ساخت اتاق بازی: (مقدار identity) هویت نامعتبر است.	MS_ERROR_FAILED_TO_LOBBY_CREATION_INVALID_IDENTITY	۴۱۲	۳
-	خطا در ساخت اتاق بازی: امکان ساخت دو اتاق توسط یک نفر نیست.	MS_ERROR_FAILED_TO_LOBBY_CREATION_CANT_OWN_TWO_LOBBY	۴۱۳	۳
-	خطا در ساخت اتاق بازی: ساخت اتاق بازی برای این حساب محدود شده است.	MS_ERROR_FAILED_TO_LOBBY_CREATION_FORBIDDEN_FOR_YOU	۴۱۴	۳
-	خطا در ساخت اتاق بازی: ساختار درخواست نامعتبر است.	MS_ERROR_FAILED_TO_LOBBY_CREATION_BAD_DATA_SYNTAX	۴۱۵	۳
-	خطا: اتاق بازی وجود ندارد که به عنوان لیست اتاق‌ها ارسال شود.	MS_ERROR_FAILED_TO_GET_LOBBY_LIST_NO_LOBBY_AVAILABLE	۴۱۶	۳
-	خطا در دریافت اطلاعات اتاق بازی: اتاقی با چنین شناسه‌ای یافت نشد.	MS_ERROR_FAILED_TO_GET_LOBBY_INFO_LOBBY_NOT_FOUND	۴۱۷	۳

-	خطا در دریافت اطلاعات اتاق بازی: ساختار درخواست نامعتبر است.	MS_ERROR_FAILED_TO_GET_LOBBY_INFO_BAD_DATA_SYNTAX	۴۱۸	۳
-	خطا در ورود مجدد به حساب کاربری: (مقدار identity) هویت نامعتبر است.	MS_ERROR_FAILED_TO_RELOGIN_IDENTITY_INVALID_OR_NOT_EXISTS	۴۱۹	۳
-	خطا در ورود مجدد به حساب کاربری: زمان ورود مجدد منقضی شده است	MS_ERROR_FAILED_TO_RELOGIN_IDENTITY_EXPIRED	۴۲۰	۳
-	خطا داخلی سرور: هنگام ثبت هویت کاربر در پایگاه داده	MS_ERROR_FAILED_TO_INSERT_CLIENT_IDENTITY	۵۰۰	۴
-	خطا داخلی سرور: هنگام ثبت حساب کاربری جدید	MS_ERROR_FAILED_TO_REGISTER_CLIENT	۵۰۱	۴
-	خطا داخلی سرور: هنگام ساخت اتاق بازی	MS_ERROR_FAILED_TO_CREATE_LOBBY	۵۰۲	۴
-	خطا داخلی سرور: هنگام دریافت لیست اتاق‌های بازی	MS_ERROR_FAILED_TO_GET_LOBBY_LIST	۵۰۳	۴
-	خطا داخلی سرور: هنگام دریافت اطلاعات اتاق بازی توسط شناسه اتاق	MS_ERROR_FAILED_TO_GET_LOBBY_INFO	۵۰۴	۴
-	خطا داخلی سرور: هنگام ورود مجدد به حساب کاربری	MS_ERROR_FAILED_TO_RELOGIN	۵۰۶	۴

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 ۳-۶ نحوه کار کرد کلی Main Client

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

- ثبت نام حساب کاربری جدید.
- ورود به حساب کاربری.
- ورود به حساب کاربری بدون نیاز به مجدد وارد کردن مشخصات حساب کاربری (در صورتی که زمان انقضای ورود تمام نشده باشد).
- خروج از برنامه.
- دیدن لیست اتاق های بازی.
- تازه کردن لیست اتاق های 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 فایل ها و پوشه های درون این نرم افزار خواهیم پرداخت:

جدول ۳-۶ 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	نگهداری Font 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 وضعیت های توسعه داده شده.	تعداد وضعیت ها زیاد است و ممکن است خوانایی برنامه و فایل ها را کمتر کند و پیدا کردن آن ها سخت باشد. بنابر این تمامی وضعیت های توسعه داده شده در این پوشه قرار داده شده است.

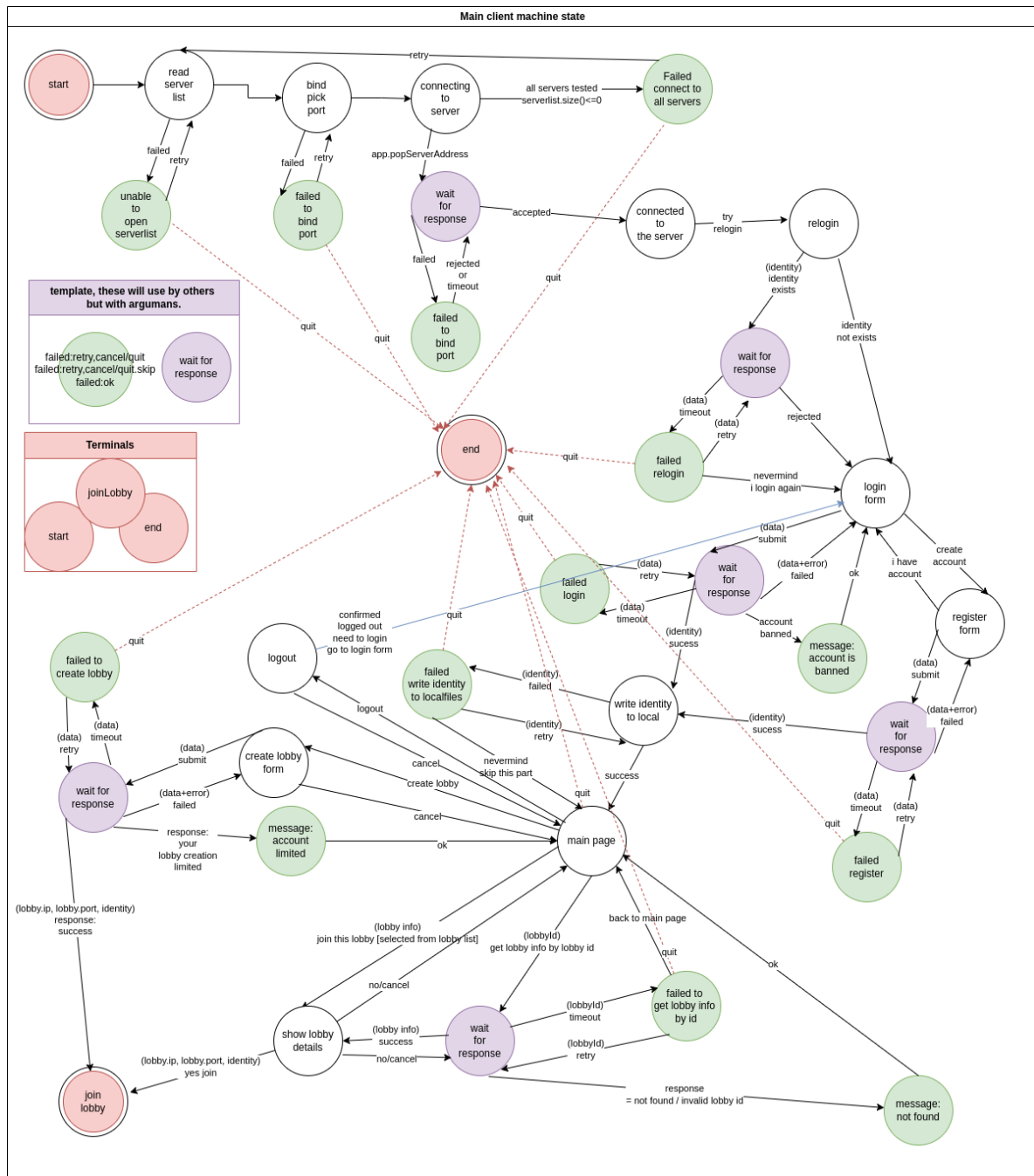
نگهداری مولفه های توسعه داده شده	تعداد مولفه های توسعه داده شده زیاد است بنابر این نیاز است آن ها را دسته بندی کنیم برای خوانایی و تمیزی پروژه.	Include/uiComponents
نگهداری و راه اندازی سوکت لیست اتاق بازی آدرس سرور مقدار identity نخ های ارسال و دریافت لیست سرور های یافت شده از فایل serverList.txt لیست اطلاعات دریافتی لیست اطلاعات ارسالی پنجره اصلی برنامه لیستی از پیام ها (Notification)	تمیز نگه داشتن فایل main همچنین سازماندهی کردن و گذاشتن محدودیت دسترسی به متغیر ها. ارسال راحت تر شیء و تابع ها با ارسال یک شیء از نوع App	App.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 وضعیت ها را حین توسعه کنترل و مدیریت کند و توسط این فایل و کلاس که به عنوان پدر به تمامی وضعیت ها ارث داده شده است.	AppState.h
نگهداری مقدار های حیاتی برنامه و همچنین مقدار های فاصله نام فایل های Texture و ... این برنامه و رابط کاربری آن.	تمیز و خوانا نگهداشتن متغیر ها و مقدار های آن ها در زمان پیشرفرض یا زمانی که متغیر در فایل های 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	Config.h

فایل با پیشوند نامگذاری شده‌اند برای تغییر و دسترسی سریعتر.		
برقراری امنیت و جدا سازی کد های مربوطه به آن از سایر بخش ها.	رمز گذاری و رمز گشایی اطلاعات ارسال شده توابع آن	DataSecurity.h
در پروژه نوع ها (Type) های داده‌ای زیاد خواهند بود و در صورتی 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).	نگهداری و جمع آوری داده‌ها و نوع های پر تکرار و مورد استفاده توسط فایل‌های متفاوت نگهداری و جمع آوری داده‌ها و نوع های پر تکرار و مورد استفاده توسط فایل‌های متفاوت	DataType.h
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.	نگهداری کد های درخواست و پاسخ با نام خوانا	Server-request-response-list.h
جلوگیری از مشکلات include های زیاد مشکل (circular dependencies)	تمامی وضعیت‌ها را در این بخش نگهداری می‌کنیم بصورت include و این فایل را در بخش source وضعیت‌ها اضافه می‌کنیم برای کد هر وضعیت نیاز به داشتن تعریفی از سایر وضعیت‌ها داریم به جهت ساخت شی	States.h

	و ارسال آن به عنوان خروجی	
UdpSocket.h	تعریف سوکت و مقدار buffer و دریافت کنند.	با جدا سازی سوکت از سایر بخش های برنامه امکان افزایش خوانایی و ماژولار کردن برنامه خواهد بود.
ServerList.txt	یک فایل که آدرس سرور های Main Server را در خود ذخیره می کند. این فایل در کنار فایل های سمت کاربر قرار خواهد گرفت.	برای آینده پیش بینی شده است که به راحتی بتوان در صورت 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 می کند. این فایل در کنار فایل های 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 شئی تولید خواهد شد با نوع App و اشاره گر مقادیری دارد که با استفاده از تابع init آن ها را مقداری دهی خواهد کرد و سپس برنامه با صدا زدن تابع run آن شئی شروع به اجرای اصلی و نمایش رابط کاربری خواهد کرد.

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



تصویر ۳-۷ وضعیت ماشین Main Client

در تصویر بالا که وضعیت ماشین را مشاهده کردید. این نرم افزار با استفاده از یک اشاره گر یا همان (Pointer) وضعیت را نگهداری می کند و در صورتی 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 می شود و سپس در حلقه اصلی توابع مربوط به آن وضعیت صدا زده خواهد شد.

جدای از توضیح وضعیت ها و ماشین وضعیت ها 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 آپی و پورت سرور های اصلی است. هدف قرار دادن این فایل و وضعیت آینده بینی این نرم افزار 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 نیازهای 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. در صورتی که کاربر نتوانست 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 ۳-۵ 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 ۳-۷ 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 شیء متن.	initSimpleText
تنظیم و جابجایی مختصات شیء.	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 ۳-۸ توضیح درباره وضعیت‌ها

نام وضعیت	وظیفه	نام وضعیت	وظیفه
State Read ServerList	خواندن فایل serverList.txt و جدا سازی آدرس و پورت‌ها و افزودن آن‌ها به لیست سرور که در App تعریف شده	State Register Form	نمایش فرم ثبت نام و بررسی رویداد های و واکنش به آن‌ها
State Bind Pick Port	انتخاب یک عدد تصادفی و قرار دادن آن به عنوان پورت برنامه و بررسی 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 بررسی می‌کند در صورتی که لیست دریافت اطلاعات خالی بود به سایر وضعیت‌هایی که گرفته است منتقل خواهد شد. در صورت 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 شیء app
State Connected To The Server	پس از اتصال به سرور به این وضعیت منتقل می‌شویم و در صورتی 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 بررسی رویداد های و واکنش به آن‌ها
State Relogin	تلاش برای ورود مجدد به حساب کاربری در صورت 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 شیء app

State Login Form	نمایش فرم ورود به حساب کاربری و بررسی 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 بازگشت 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 قابلیت های آن کتابخانه جهت توسعه رابط کاربری و فرم ها نیازمند اقداماتی می باشد 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 روش ها را استفاده و پیاده سازی خواهیم کرد.

```
namespace FluffyMultiplayer
{
    class IconText : public Icon , public Text
    {
    public:

        void initIconText(std::string text, std::string texture,
            float x, float y)
        {
            initIcon(texture,x,y);
            initText(text,x ICON_TEXT_TEXT_PADDING_X,y ICON_TEXT_TEXT_PADDING_Y);
        }

        IconText(std::string text, std::string texture,float x, float y)
            : Icon(texture,x,y) ,
              Text(text,x ICON_TEXT_TEXT_PADDING_X,y ICON_TEXT_TEXT_PADDING_Y)
        {

        }

        IconText() : Icon() , Text()
        {

        }

        ~IconText()
        {

        }

        void render(sf::RenderWindow& window)
        {
            window.draw(icon); //from paren (Icon)
            window.draw(textText);
        }

    };
}
```

تصویر ۸-۳ کد نمونه تولید مولفه IconText

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

جدول ۹-۳ جزئیات مولفه های (Component) توسعه یافته

نام	کاربرد	تابع و کاربرد آنها	تأثیر گرفته از
۱	Button	<div>تنظیم و جابجایی مختصات شیئ</div> <div>تنظیم رنگ متن دکمه</div> <div>تنظیم رنگ زمینه دکمه</div> <div>تنظیم ابعاد دکمه</div> <div>راه اندازی دکمه با مقادیر اولیه و یا مقادیر داده شده.</div> <div>تابع سازنده که صورت های مختلفی دارد جهت استفاده های مختلف و مقادیر دریافتی را به تابع init ارسال خواهد کرد</div> <div>پنجره را بصورت رفرنس دریافت می کند و دو شیئ (متن و تصویر زمینه) را با دستور draw ترسیم خواهد کرد.</div>	UiComponent
۲	CheckBox	<div>به عنوان ورودی یک مقدار ۰ یا ۱ را می گیرد و وضعیت دکمه را تغییر می دهد (وضعیت دارد Checked و Unchecked)</div> <div>تنظیم و جابجایی مختصات شیئ</div> <div>اعلام مختصات و ابعاد شیئ</div>	UiComponent

			نمایش در کنار دکمه که هر دو آن‌ها با بر طبق یک نقطه مختصات دهی می‌شوند و مقدار Padding آن‌ها در فایل Config.h قرار دارد.											
		<table><tr><td>init</td><td>راه اندازی CheckBox با مقادیر اولیه و یا مقادیر داده شده</td></tr><tr><td>getStatus</td><td>مقدار متغیر isChecked را باز می گرداند</td></tr><tr><td>render</td><td>پنجره را بصورت رفرنس دریافت می کند و دو شیء (متن و تصویر زمینه) را با دستور draw ترسیم خواهد کرد</td></tr><tr><td>CheckBox</td><td>تابع سازنده که صورت‌های مختلفی دارد جهت استفاده های مختلف و مقادیر دریافتی را به تابع init ارسال خواهد کرد</td></tr></table>	init	راه اندازی CheckBox با مقادیر اولیه و یا مقادیر داده شده	getStatus	مقدار متغیر isChecked را باز می گرداند	render	پنجره را بصورت رفرنس دریافت می کند و دو شیء (متن و تصویر زمینه) را با دستور draw ترسیم خواهد کرد	CheckBox	تابع سازنده که صورت‌های مختلفی دارد جهت استفاده های مختلف و مقادیر دریافتی را به تابع init ارسال خواهد کرد				
init	راه اندازی CheckBox با مقادیر اولیه و یا مقادیر داده شده													
getStatus	مقدار متغیر isChecked را باز می گرداند													
render	پنجره را بصورت رفرنس دریافت می کند و دو شیء (متن و تصویر زمینه) را با دستور draw ترسیم خواهد کرد													
CheckBox	تابع سازنده که صورت‌های مختلفی دارد جهت استفاده های مختلف و مقادیر دریافتی را به تابع init ارسال خواهد کرد													
UiComponent	<table><tr><td>changeIconTexture</td><td>تغییر تصویر شیء</td></tr><tr><td>setIconPosition</td><td>تنظیم و جابجایی مختصات شیء</td></tr><tr><td>initIcon</td><td>راه اندازی Icon با مقادیر داده شده.</td></tr><tr><td>render</td><td>پنجره را بصورت رفرنس دریافت می کند و دو شیء (تصویر) را با دستور draw ترسیم خواهد کرد</td></tr><tr><td>Icon</td><td>تابع سازنده که صورت‌های مختلفی دارد جهت استفاده های مختلف و مقادیر دریافتی را به تابع initIcon ارسال خواهد کرد.</td></tr></table>	changeIconTexture	تغییر تصویر شیء	setIconPosition	تنظیم و جابجایی مختصات شیء	initIcon	راه اندازی Icon با مقادیر داده شده.	render	پنجره را بصورت رفرنس دریافت می کند و دو شیء (تصویر) را با دستور draw ترسیم خواهد کرد	Icon	تابع سازنده که صورت‌های مختلفی دارد جهت استفاده های مختلف و مقادیر دریافتی را به تابع initIcon ارسال خواهد کرد.	یک تصویر را به نمایش می‌گذارد و با کلیک بر روی آن هیچ اتفاقی رخ نخواهد داد.	Icon	۳
changeIconTexture	تغییر تصویر شیء													
setIconPosition	تنظیم و جابجایی مختصات شیء													
initIcon	راه اندازی Icon با مقادیر داده شده.													
render	پنجره را بصورت رفرنس دریافت می کند و دو شیء (تصویر) را با دستور draw ترسیم خواهد کرد													
Icon	تابع سازنده که صورت‌های مختلفی دارد جهت استفاده های مختلف و مقادیر دریافتی را به تابع initIcon ارسال خواهد کرد.													

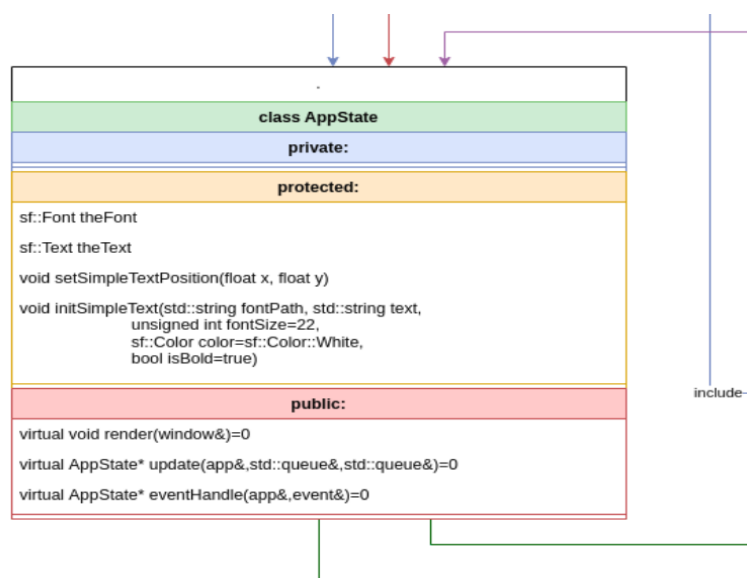
۴	Icon Text	همانند مولفه Icon یک تصویر را دریافت می‌کند. همچنین یک متن را در زیر آن تصویر نمایش خواهد داد و قابلیت کلیک ندارد	مقادیر گرفته شده را به توابع initIcon و initText ارسال خواهد کرد	initIconText	بصورت غیر مستقیم از UiComponent
			تابع سازنده که صورت‌های مختلفی دارد و مقادیر دریافت شده را با صدا زدن تابع های سازندهای Icon و Text ارسال می‌کند.	IconText	مستقیم از Icon و Text
			ترسیم دو شی که در مولفه های پدر موجود دارن اقدام می‌کند	render	
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 دیگری به عنوان Placeholder در صورت 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	اعلام مختصات و ابعاد شی	getInputBoxBound	UiComponent
			تنظیم و جابجایی مختصات شی	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	

			یکی به عنوان Title نمایش داده خواهند شد.		
		enablePlaceholder	فعال کردن Placeholder و نمایش آن		
		disablePlaceholder	غیرفعال کردن Placeholder و نمایش آن		
		render	ترسیم تصویر زمینه. در صورت فعال بودن وضعیت hideTitleAndError دو عنوان بالای مولفه ورودی و همچنین یک متن در وسط را ترسیم می کند.		
		specificRender	یک تابع که بصورت مجازی تعریف شده برای زمانی که می خواهیم از این مولفه ارث ببریم و مولفه جدیدی تولید کنیم برای مثال می توان در آن یک دکمه جدید در کنار تمامی موارد ترسیم کرد با استفاده از این تابع این امر امکان پذیر خواهد بود.		
		getEnteredText	رشته وارد شده توسط کاربر را باز می گرداند.		
UiComponent		setPosition	تنظیم و جابجایی مختصات شی	Text	۶
		setText	تنظیم متن		
		setFontSize	تنظیم اندازه متن		
		initText	فایل فونت را برای متن جاگذاری می کند با استفاده از تابع componentLoadAndSetFont و سپس رشته ورودی را در متن و مختصات قرار گیری آن را تنظیم می کند.		

			<table><tr><td>Text</td><td>تابع سازنده که صورت‌های مختلفی دارد جهت استفاده های مختلف و مقادیر دریافتی را به تابع <code>initText</code> ارسال خواهد کرد.</td></tr><tr><td>render</td><td>پنجره را بصورت رفرنس دریافت می‌کند و متن را با دستور <code>draw</code> ترسیم خواهد کرد.</td></tr></table>	Text	تابع سازنده که صورت‌های مختلفی دارد جهت استفاده های مختلف و مقادیر دریافتی را به تابع <code>initText</code> ارسال خواهد کرد.	render	پنجره را بصورت رفرنس دریافت می‌کند و متن را با دستور <code>draw</code> ترسیم خواهد کرد.		
Text	تابع سازنده که صورت‌های مختلفی دارد جهت استفاده های مختلف و مقادیر دریافتی را به تابع <code>initText</code> ارسال خواهد کرد.								
render	پنجره را بصورت رفرنس دریافت می‌کند و متن را با دستور <code>draw</code> ترسیم خواهد کرد.								
۷	Spin Box	<p>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).</p> <p>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.</p>							
۸	Picture Button	یک دکمه است با تصویر که یک متن در زیر آن قرار دارد.	<table><tr><td>setPosition</td><td>تنظیم و جابجایی مختصات شی</td></tr><tr><td>getButtonBound</td><td>اعلام مختصات و ابعاد شی</td></tr><tr><td>init</td><td>راه اندازی <code>PictureButton</code> با مقادیر اولیه و یا مقادیر داده</td></tr></table>	setPosition	تنظیم و جابجایی مختصات شی	getButtonBound	اعلام مختصات و ابعاد شی	init	راه اندازی <code>PictureButton</code> با مقادیر اولیه و یا مقادیر داده
setPosition	تنظیم و جابجایی مختصات شی								
getButtonBound	اعلام مختصات و ابعاد شی								
init	راه اندازی <code>PictureButton</code> با مقادیر اولیه و یا مقادیر داده								

		شده			
		render	ترسیم متن و دکمه شیء		
		PictureButton	تابع سازنده که صورت‌های مختلفی دارد برای انتقال داده‌ها به تابع init هنگام ساخت شیء		
UiComponent ولی از اشیاء بالا استفاده شده مثل Icon و Text و IconText	getButtonBound	اعلام مختصات و ابعاد شیء	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 شیء از نوع LobbyData نگهداری می‌کند.	Lobby Cell	۹
	getLobbyData	مشخصات اتاق بازی که به نمایش در آورده را باز می‌گرداند.			
	init	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.			
	render	تابع render شیء های درون خود را صدا می‌زند و پنجره را به آن‌ها منتقل می‌کند که آن‌ها موارد مورد نیاز را ترسیم کنند.			
	LobbyCell	تابع سازنده که صورت‌های مختلفی دارد و مقادیر دریافت شده را به تابع init انتقال می‌دهد.			
UiComponent ولی از اشیاء بالا استفاده شده مثل	init	راه اندازی NotificationBox با مقادیر داده شده	برای نمایش یک popup نیازمند لیست کردن آن‌ها هستیم که فقط حاوی متن نیستند بنابر این یک لیست	Notification Box	۱۰
	getBackofBackgroundBound	اعلام مختصات و ابعاد شیء			
	closeBox	وضعیت boxClosed را به خروج تغییر می‌دهد			

Icon و Text	copyError	جهت کپی کردن متن خطا قرار داده شده. (این بخش به صورت کامل کار نمی کند و نیاز است کتابخانه مورد نیاز برای CopyClipBoard وارد و اعمال شود).	از این شیء خواهیم ساخت و مقادیر را در آن اضافه می کنیم. این مولفه حاوی تصویر زمینه و عنوان و متن وسط و همچنین دو دکمه که با فشردن آن متن خطا کپی می شود و دیگری این popup را می بندد. اطلاعات در یک متغیر از نوع NotificationDat a نگهداری می کنیم.	
	NotificationBox	تابع سازنده که هیچ کاری نمی کند و برای مقدار دهی این مولفه نیاز است حتماً init صدا زده و مقدار دهی شود.		
	render	در صورت True بودن وضعیت متغیر boxClosed مولفه ها و شیء های درون را با انتقال پنجره به تابع render ترسیم خواهد کرد.		



تصویر ۹-۳ نمونه UML کلاس AppState

فایل های مستندات مربوط به این نرم افزار در پوشه docs قرار داده شده اند می توانید 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.

```
case REQUEST_JOIN_TO_LOBBY: // player will send cData = [0]->(identity), [1]->(password), [2]->(his client version)
{
    if(cData[1] == lobbyData.password)
    {
        if(ds.isIdentityValid(cData[0]))
        {
            if(cData[2] == GAME_SERVER_VERSION)
            {

                //get client info by identity
                db.queryStr="SELECT clientId FROM fm_client_login WHERE identity='";
                db.queryStr+= cData[0] + "'";
                int cId = stringToInt(db.search_in_db(db.queryStr,true));
                if(cId>=1) //is client exists?
                {
                    //check for old connected with that id and kick them out,
                    //e.g player timeout and connected again we have to remove that old one from player lists
                    if(isIdExistsInLobby(cId))
                    {
                        kickOutPlayerFromLobby(cId);
                    }

                    if(isIdBannedFromLobby(cId))
                    {
                        response(RESPONSE_ERROR_JOIN_LOBBY_YOU_ARE_BANNED,currentItem.sender, false);
                    }
                    else
                    {
                        //try to add client into lobby.
                        db.queryStr="INSERT OR REPLACE INTO fm_client_in_lobby (clientId,lobbyId) VALUES('";
                        db.queryStr+= std::to_string(cId) + "', ";
                        db.queryStr+= std::to_string(lobbyData.id) + "')";
                        if(db.query_to_db())

```

تصویر ۱۰-۳ 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.

```

default:
{
    if(gameIsRunning)
    {
        if(currentGameMode!=nullptr)
        {
            log.print("unknown request code lets passing it to gamemode", FluffyMultiplayer::LogType::Information);
            currentGameMode = currentGameMode->process((*this),currentItem,cData);
            //response to client will push by currentGameMode->process
        }
        else
        {
            log.print("unknown request code and gamemode is nullptr (game is not started)", FluffyMultiplayer::LogType::Warning);
            response(RESPONSE_UNKNOWN_REQUEST_GAME_IS_NOT_STARTED,currentItem.sender, false);
        }
    }
    else
    {
        log.print("unknown request code and game running = false (not started or paused)", FluffyMultiplayer::LogType::Warning);
        response(RESPONSE_UNKNOWN_REQUEST_GAME_PAUSED_OR_NOT_STARTED,currentItem.sender, false);
    }
}
}

```

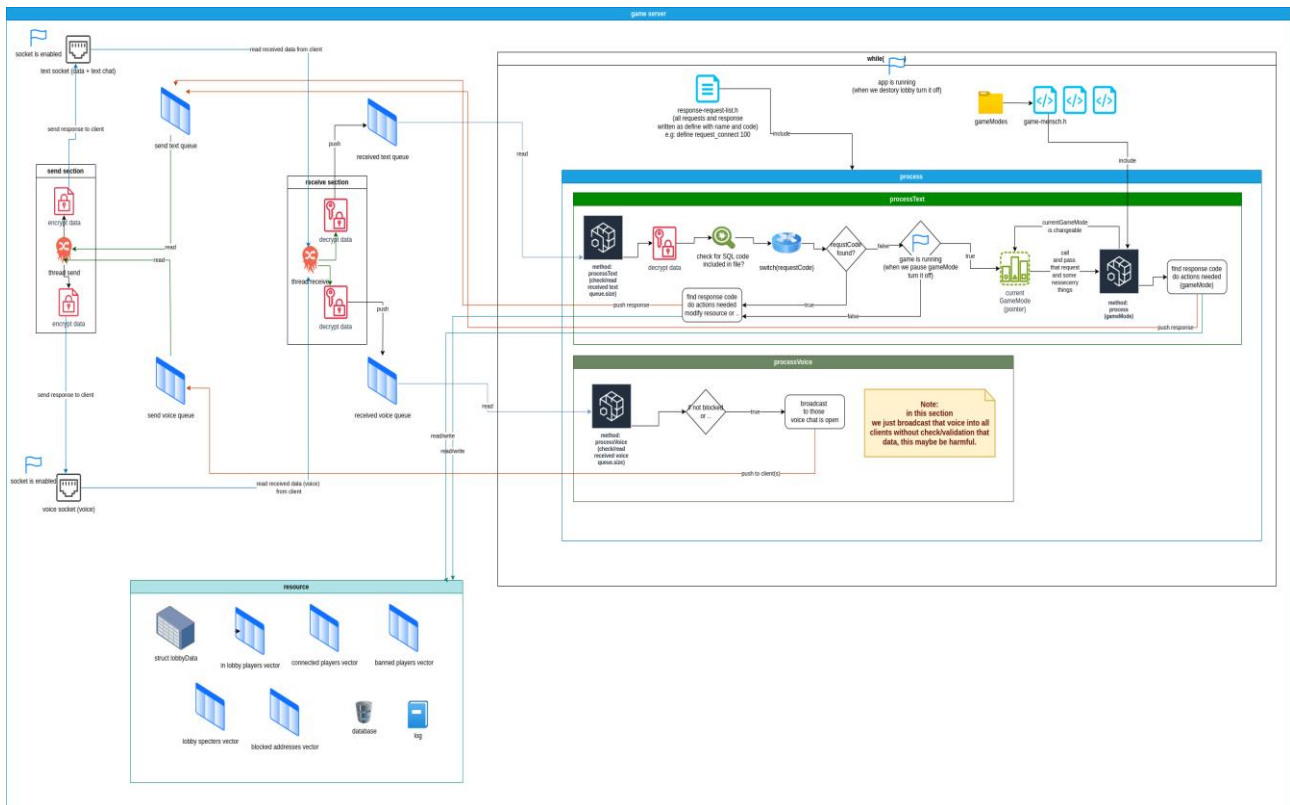
تصویر ۳-۱۱ کد ارسال درخواست جاری در صورت تعریف نشدن در درخواست های اتاق بازی

نگاهی به وظایف این برنامه داشته باشیم:

- تأیید هویت کاربر
- تأیید و بررسی کلمه عبور اتاق
- تأیید و بررسی 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
- در صورت خروج مدیر از اتاق مدیریت را انتقال کند به سایر کاربران
- در صورت خروج تمامی کاربران اتاق بازی را تخریب کند
- شروع و توقف بازی
- ارسال پیام متنی
- ارسال پیام صوتی
- ارسال تنظیمات جدید (در صورتی 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 ۳-۱۲ کار کرد کلی Game Server

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

جدول ۱۰-۳ 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 آن ها 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 پاسخ ها 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 راحت تر 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 از خطا ها یا اتفاقاتی که در سرور رخ می دهد دفترچه ثبت وقایعی داشته باشیم جهت عیب یابی.
include/gameModes	نگهداری فایل های مدل بازی	مدل های بازی ممکن است زیاد شوند بنابر این با استفاده از این پوشه آن ها را جدا می کنیم 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:

۱. در صورتی 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.

۲. در صورتی 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 سرورهای 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 نیازها و درخواست‌های کاربرها می‌باشد. با دریافت دستور شروع بازی توسط مدیر اتاق شروع به ارسال درخواست‌های بازی به سمت مدل بازی در خود می‌کند.

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

جدول ۳-۱۱ محدوده کدها درخواست‌ها و پاسخ‌های Game Client و Game Server

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 ۱۲-۳ درخواست و پاسخ Game Client و Game Server

نوع	کد	نام	توضیحات	حاوی اطلاعات
۱	۱۰۰	REQUEST CONNECT TO LOBBY	درخواست اتصال به اتاق بازی	-
۱	۱۰۱	REQUEST RECONNECT TO LOBBY	درخواست بازیابی اتصال اتاق بازی (جای گذاری شده برای آینده نرم افزار در صورتی که کاربر از اتاق به دلیل مشکلات اینترنتی خارج شده و پس از مدتی دوباره به اتاق متصل می شود).	-
۱	۱۰۲	REQUEST DISCONNEC T FROM LOBBY	درخواست خروج از اتاق بازی	-
۱	۱۰۳	REQUEST JOIN TO LOBBY	درخواست ورود به اتاق بازی که اطلاعاتی از به سرور ارسال می کند و سرور با بررسی آن اطلاعات تصمیم می گیرد که این کاربر را متصل کند یا خیر.	Identity کاربر کلمه عبور وارد شده نسخه برنامه خود
۱	۱۰۴	REQUEST GET LOBBY SETTINGS	این درخواست مخصوص مدیر اتاق می باشد و زمانی که بر روی دکمه تنظیمات اتاق بازی کلیک می کند اطلاعات اتاق بازی را به او ارسال می کند.	-
۱	۱۰۵	REQUEST UPDATE LOBBY SETTINGS	این درخواست مخصوص مدیر اتاق می باشد. 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
۱	۱۰۶	REQUEST_ST OP_START_G AME	این درخواست مخصوص مدیر اتاق می باشد که با آن باز را شروع یا پایان می دهد.	-

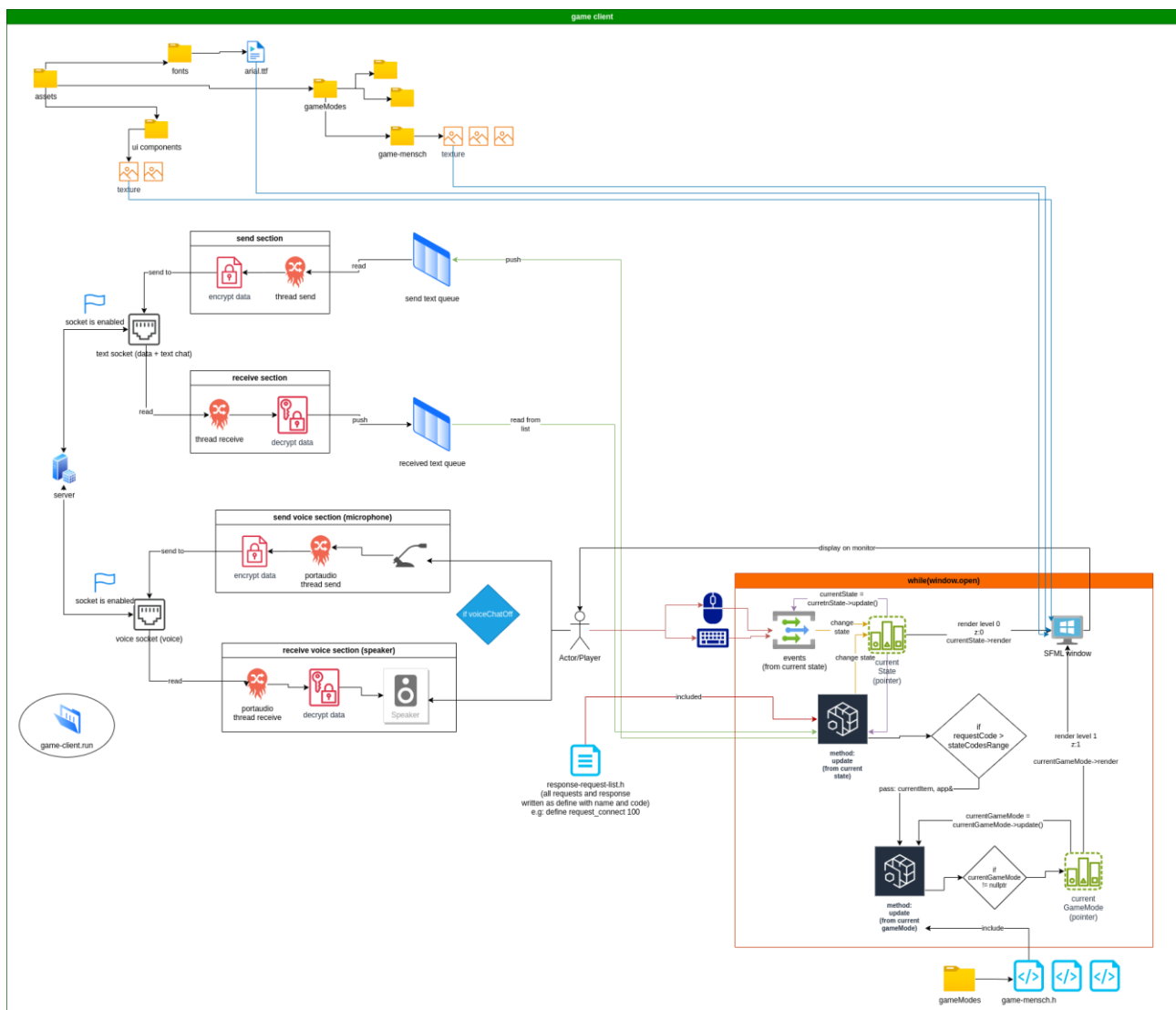
۱	۱۰۷	-	-	-
۱	۱۰۸	REQUEST_DELETE_LOBBY	این درخواست مخصوص مدیر اتاق می باشد که با آن اتاق بازی را تخریب می کند.	-
۱	۱۰۹	REQUEST_KICK_PLAYER	این درخواست مخصوص مدیر اتاق می باشد که برای آینده پیش بینی شده که یک کاربر را از اتاق بیرون کند.	-
۱	۱۱۰	REQUEST_BAN_PLAYER	این درخواست مخصوص مدیر اتاق می باشد که برای آینده پیش بینی 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 پیش بینی شده و قرار گرفته و هنگام اتصال کاربر شناسه آن و همچنین آدرس آیپی آن مورد بررسی قرار می گیرد و از ورود فرد ممنوع شده جلوگیری می شود.	شناسه کاربر هدف دلیل بصورت متنی زمان ممنوعیت
۱	۱۱۱	-	-	-
۱	۱۱۲	REQUEST_SEND_TEXT_CHAT	کاربران درون اتاق با استفاده از این درخواست می توانند پیام متنی را به اتاق ارسال کنند.	متن پیام
۱	۱۱۳	REQUEST_SEND_VOICE_CHAT	کاربران درون اتاق در صورتی که وضعیت ارتباط صوتی آن ها فعال باشد می توانند پیام های صوتی خود را به اتاق ارسال کنند.	محتوای صوتی (به اندازی buffer که در نظر گرفته شده).
۱	۱۱۴	REQUEST_ENABLE_DISABLE_VOICE_CHAT	با این درخواست کاربران وضعیت جاری ارتباط صوتی خود را برعکس می کنند.	-
۱	۱۱۵	REQUEST_GET_LOBBY_INFO	زمانی که مدیر اتاق تنظیمات اتاق را ویرایش می کند و به صفحه اصلی باز می گردد این درخواست بصورت خودکار به سرور ارسال می شود 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.				
۲	۲۰۳	RESPONSE_P LAYER_DISC ONNECTED	هنگامی که یک کاربر از اتاق خارج می‌شود این پیام به سایر کاربران ارسال خواهد شد	شناسه کاربری که خارج شده
۲	۲۰۴	RESPONSE_P LAYER_JOIN T_INTO_LOB BY	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)
۲	۲۰۶	RESPONSE_P LAYER_KICK ED_FROM_L OBBY	هنگامی که یک کاربر از اتاق اخراج می‌شود این پیام به سایر کاربران ارسال خواهد شد.	شناسه کاربر هدف
۲	۲۰۷	RESPONSE_P LAYER_BAN NED_FROM_ LOBBY	هنگامی که یک کاربر از اتاق ممنوع می‌شود این پیام به سایر کاربران ارسال خواهد شد.	شناسه کاربر هدف
۲	۲۱۰	RESPONSE_P LAYER_VOIC E_CHAT_ENA BLED	هنگامی که وضعیت ارتباط صوتی یک کاربر فعال می‌شود این پاسخ به سایر کاربران ارسال خواهد شد.	شناسه کاربر هدف
۲	۲۱۱	RESPONSE_P LAYER_VOIC E_CHAT_DIS ABLED	هنگامی که وضعیت ارتباط صوتی یک کاربر غیرفعال می‌شود این پاسخ به سایر کاربران ارسال خواهد شد.	شناسه کاربر هدف
۲	۲۱۲	RESPONSE_P LAYER_SENT _VOICE_MES SAGE	هنگامی که یک پیام صوتی دریافت شده باشد در آن را به سایر کاربران ارسال خواهد کرد.	محتوای پیام صوتی
۲	۲۱۳	RESPONSE_P LAYER_SENT _TEXT_MESS AGE	هنگامی که یک پیام متنی دریافت شده باشد در آن را به سایر کاربران ارسال خواهد کرد.	محتوای پیام متنی
۲	۲۱۶	RESPONSE_L OBBY_SETTI NGS_UPDAT ED	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				
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	اطلاعات و مشخصات اتاق بازی را بدون فیلتر را به مدیر اتاق ارسال می‌کند.	RESPONSE_LOBBY_SETTINGS_IS	۲۱۷	۲
شناسه مدیر جدید	زمانی که مدیر اتاق از اتاق خارج می‌شود این پیام به سایر کاربران ارسال خواهد شد.	RESPONSE_LOBBY_OWNER_CHANGED	۲۱۸	۲
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)	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).	RESPONSE_LOBBY_PLAYERS_ARE	۲۱۹	۲
شناسه اتاق حداکثر کاربران مجاز شناسه مدل بازی تعداد کاربران جاری وضعیت مجوز ارتباط صوتی وضعیت مجوز ارتباط متنی وضعیت مجوز ورود تماشاگر	مشخصات اتاق بازی را به کاربران باز می گرداند.	RESPONSE_LOBBY_INFO_IS	۲۲۲	۲
-	زمانی که کاربر درخواست ویرایش تنظیمات اتاق بازی را می‌کند و مجوز و دسترسی لازم را ندارد این پاسخ ارسال خواهد شد.	RESPONSE_ERROR_UPDATE_LOBBY_SETTINGS_NO_PERMISSION	۳۱۳	۳

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 درخواست‌ها و پاسخ‌های 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 (درخواست‌های پایه) را شامل می‌شود و همینطور مدل بازی هردو از لیست ارسال اطلاعات استفاده می‌کنند.



تصویر ۱۳-۳ کارکرد کلی Game Client

زمانی که Thread در حال بررسی 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 می شود بصورت ناقص باشد بنابر این نیاز است از Mutex استفاده کنیم.

روش کار بدین صورت می شود که Mutex شبیه به یک پرچم Flag است که زمانی که نیاز است اطلاعات را در لیست قرار بدهیم آن را قفل می کنیم و زمانی که می خواهیم اطلاعات را بخوانیم بررسی می کنیم که قفل نباشد. با این روش می توان دسترسی بخش دیگر را از لیست مشترک گرفته و محدود کنیم.

```
void App::sendData()
{
    FluffyMultiplayer::SocketSendData currentItem;
    while(true)
    {
        try
        {
            //lock mutex for socket to avoid segment fault or data corruption.
            boost::lock_guard<boost::mutex> lock(sendMutex);

            //text data
            if(sendTextDataList.size()>=1) //on that socket self will check socketStatus before send
            {
                currentItem = sendTextDataList.front();
                ds.encryptData(currentItem.data);

                socketText->send(currentItem);

                //remove processed element
                sendTextDataList.pop();
            }
        }
        catch (std::exception& e)
        {
            std::string errorMsg = e.what();
            log.print("from App::sendData catched exception: "+errorMsg, FluffyMultiplayer::LogType::Warning);
            continue;
        }
    }
}
```

تصویر ۱۴-۳ نمونه کد استفاده از Mutex

این نرم افزار مشابه به Main Client است یعنی از ماشین وضعیت استفاده می کند و وضعیت های مختلفی را دارد و در کنار آن مدل های بازی هم نیاز است در این قسمت پیاده سازی شوند تا کاربر بتواند در صورت دریافت کد پاسخ از سمت سرور که مربوط به مدل بازی می باشد را بصورت درست به نمایش بگذارد و کاربر نتیجه عمل خود را ببیند. بنابر این در یکی از وضعیت ها 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 صدا خواهیم زد و این امر باعث می شود تا کد های مدل بازی در کنار این نرم افزار به اجرا دربیایند.

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

زمانی که درخواست ها بررسی می شود کد آن در یک Switch قرار می گیرد.

```
FluffyMultiplayer::AppState* StateMainPage::update(FluffyMultiplayer::App& app)
{
    //read from received data..
    FluffyMultiplayer::SocketReceiveData currentItem;
    std::vector<std::string> cData;
    if(app.receivedTextDataList.size()>=1)
    {
        for(int i=0; i<app.receivedTextDataList.size(); i++)
        {
            currentItem = app.receivedTextDataList.front();
            cData = dataSeparator(currentItem.data, MS_DATA_DELIMITER);

            app.log.print("update called, currentItem.Code="+std::to_string(currentItem.code), FluffyMultiplayer::LogType::Information);

            switch(currentItem.code)
            {
                case RESPONSE_LOBBY_INFO_IS:
```

تصویر ۳-۱۵ کد بررسی یک پاسخ از سوی سرور

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

```
default:
{
    //apply commands from server into client game
    if(app.currentGameMode!=nullptr && app.gameIsRunning)
        app.currentGameMode->update(currentItem.code,cData);
    else
        std::cout << "gameMode is null and received response code =" << currentItem.code << std::endl;
}
}

//remove proccess item
app.receivedTextDataList.pop();
```

تصویر ۳-۱۶ کد ارسال پیام دریافت شده به مدل بازی

روند بررسی درخواست به مدل بازی داده می شود و آن عمل های مورد نیاز را برای آن درخواست یا پاسخ اعمال می کند.

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:

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.	<div> <div>setOwner</div> <div>وضعیت مدیر بودن را برای کاربر تغییر می‌دهد.</div> </div> <div> <div>clear</div> <div>اطلاعات کاربر را پاک می‌کند</div> </div> <div> <div>updateVoiceChatStatus</div> <div>وضعیت ارتباط صوتی کاربر را فعال یا غیر فعال می‌کند</div> </div> <div> <div>getName</div> <div>مقدار نام کاربر را بازمیگرداند</div> </div> <div> <div>getId</div> <div>تنظیم ابعاد دکمه</div> </div> <div> <div>init</div> <div>راه اندازی PlayerList با مقادیر اولیه و یا مقادیر داده شده.</div> </div> <div> <div>setName</div> <div>تنظیم نام برای کاربر</div> </div> <div> <div>PlayerList</div> <div>تابع سازنده که صورت‌های مختلفی دارد جهت استفاده های مختلف و مقادیر دریافتی را به تابع init ارسال خواهد کرد</div> </div> <div> <div>render</div> <div>پنجره را بصورت رفرنس دریافت می‌کند و شیء های (شناسه و نام کاربری و وضعیت ارتباط صوتی) را ترسیم می‌کند و در صورت فعال بودن وضعیت مدیریت. ادمین. تماشاگر تصاویر آن‌ها را ترسیم خواهد کرد.</div> </div>	<div> <div>بصورت مستقیم از UiComponent</div> <div>غیر مستقیم از Icon و Text و Picture Button</div> </div>

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.

```
namespace FluffyMultiplayer
{
    class GameMode
    {
    protected:
        int gameModeId;

        //a flag to turn on when game started, App will render game in some case gameMode before start has bad position of elemnts
        //so this flag make game mode to avoid that render before,
        bool gameStarted;

    public:
        bool getGameStarted() const
        {
            return gameStarted;
        }
        virtual void updatePlayersInGame(std::vector<std::string>& receivedPlayersInGameInfo,int myId)=0;
        virtual FluffyMultiplayer::GameMode* update(int& currentItemCode, std::vector<std::string>& cData)=0;
        virtual void render(sf::RenderWindow& window)=0;
        virtual FluffyMultiplayer::GameMode* eventHandle(sf::RenderWindow& window,sf::Event& event,
            std::queue<FluffyMultiplayer::SocketSendData>& sendList,
            boost::mutex& sendMutex)=0;
    };
}
```

تصویر ۳-۱۷ کلاس GameMode در سمت کاربر

جدول ۳-۱۴ معرفی متغیرهای 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 ۱۵-۳ 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 در صورتی 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 صفحه 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 د ستر سی 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.	-

```

#ifndef H_GAMEMODE_EXAMPLE_CLASS
#define H_GAMEMODE_EXAMPLE_CLASS

#include <array>
#include <vector>
#include "../gameMode.h"
#include "../config.h"

namespace FluffyMultiplayer
{
    class ExampleGameMode : public GameMode
    {
    public:

```

تصویر ۳-۱۸ نوشتن مد بازی

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 gameId قرار دهید.

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

```

void updatePlayersInGame(std::vector<std::string>& receivedPlayersInGameInfo, int myId)
{
    //code to seperate receivedPlayersInGameInfo. each player has this data: (id,name,index)
    //so each player has 3 data

    //how to know how many players received
    int playersCount = receivedPlayersInGameInfo.size()/3;

    //for example to access to first player's info
    int playerId = receivedPlayersInGameInfo[0];
    std::string playerName = receivedPlayersInGameInfo[1];
    int playerIndex = receivedPlayersInGameInfo[2]; //can use this as index of color or team or ...

    if(playerId == myId)
    {
        //this player is this pc, so for example can assing a tag to make easy to find him self.
        playerName += " [ME]";
    }

    //append collected players into game players or etc on your game mode.
}

```

تصویر ۳-۱۹ مثال چگونگی تعریف تابع updatePlayersInGame


```

FluffyMultiplayer::GameMode* update(int& currentItemCode, std::vector<std::string>& cData)
{
    switch (currentItemCode)
    {
        case 759: //this code is asking for quit game.
        {
            gameStarted = false;

            //to access to first data e.g: first element is id
            int requesterId = convertStringToInt(cData[0]);

            //make game quit in diffrent way
            return nullptr;

        }break;

        default:
        {
            std::cout << "unknown request/response code\n";
        }break;
    }

    return this;
}

```

تصویر ۳-۲۰ مثال چگونگی تعریف تابع update

```

void render(sf::RenderWindow& window)
{
    //things wants to render
    sf::Sprite exampleSprite;
    window.draw(exampleSprite);

    FluffyMultiplayer::Text exampleText;
    exampleText.render(window);
}

```

تصویر ۳-۲۱ مثال چگونگی تعریف تابع render

```

FluffyMultiplayer::GameMode* eventHandle(sf::RenderWindow& window, sf::Event& event,
                                           std::queue<FluffyMultiplayer::SocketSendData>& sendList,
                                           boost::mutex& sendMutex)
{
    switch(event.type)
    {
        case sf::Event::KeyPressed:
        {
            std::cout << "key pressed" << std::endl;
        }break;
    }
}

```

تصویر ۳-۲۲ مثال چگونگی تعریف تابع eventHandle

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

```
namespace FluffyMultiplayer
{
    class App; // Forward declaration
    class GameMode
    {
    protected:
        int gameModeId;

    public:
        virtual void startGameMode()=0;
        virtual void addPlayerToGame(FluffyMultiplayer::App& app, int index)=0;
        virtual void removePlayerFromGame(FluffyMultiplayer::App& app, int playerId)=0;
        virtual int howManyPlayersAreInGame() const =0;
        virtual FluffyMultiplayer::GameMode* process(FluffyMultiplayer::App& app,
            const FluffyMultiplayer::SocketReceiveData& currentItem,
            const std::vector<std::string>& cData //to avoid re-process data just pass seperated data into gameMode
            ) = 0;
    };
}
```

تصویر ۲۴-۳ کلاس GameMode در سمت سرور

جدول ۱۶-۳ 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 ۱۷-۳ 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 اضافه 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 شیء 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 بررسی 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.

```

#ifndef GAMEMODE_EXAMPLE
#define GAMEMODE_EXAMPLE

#include "../gameMode.h"

namespace FluffyMultiplayer
{
    class ExampleGameMode : public GameMode
    {
    private:
        //other methods

    public:
        //pure functions

    };
}

```

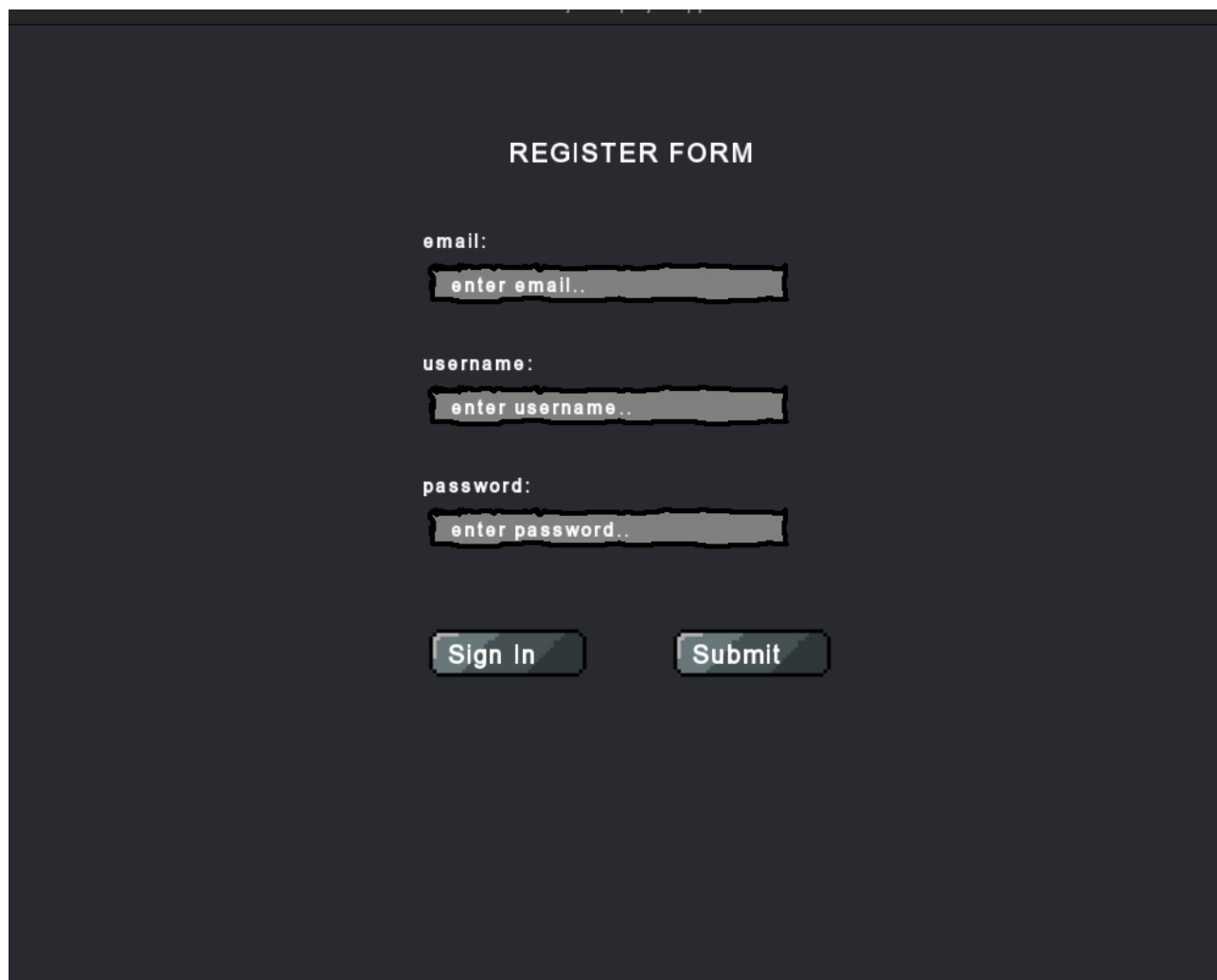
تصویر ۲۵-۳ مثال چگونگی تعریف مد بازی در سمت سرور

همچنین 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

نتیجه گیری

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 فریم‌ورک‌ها و متورهای بازی سازی و ابزارهای پیشرفته و مدرن برای توسعه استفاده کرد. تصویری از برنامه پیاده شده:



The image shows a dark-themed web interface for a registration form. At the top, the title 'REGISTER FORM' is centered in white capital letters. Below the title, there are three input fields, each preceded by a label: 'email:', 'username:', and 'password:'. The input fields are light gray with rounded corners and contain the placeholder text 'enter email..', 'enter username..', and 'enter password..' respectively. At the bottom of the form, there are two buttons: 'Sign In' and 'Submit'. Both buttons are dark gray with rounded corners and white text. The entire form is set against a dark blue-gray background.

تصویر ۴-۱ فرم ساخت حساب کاربری Main Client

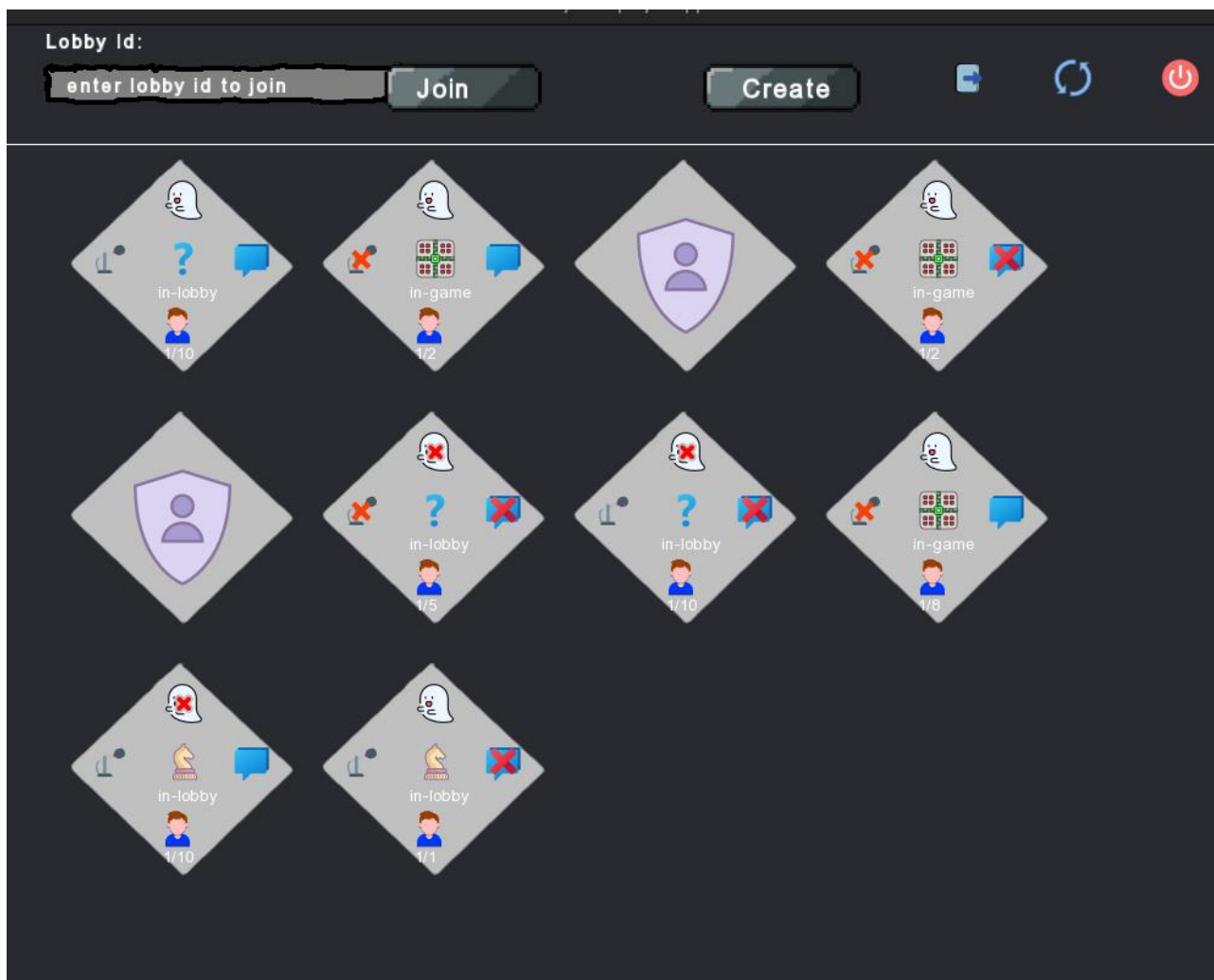
LOGIN FORM

username:

password:

save login ☒

تصویر ۴-۲ فرم ورود به حساب کاربری Main Client



تصویر ۳-۴ صفحه اصلی Main Client

Create Lobby FORM

Password:

enter password..

game mode:Unknown

max players:2

specter: ☒

voice chat: ☒

text chat: ☒

cancel submit

تصویر ۴-۴ فرم ساخت اتاق بازی Main Client



تصویر ۴-۵ صفحه اصلی Game Client (بازی منچ در حال اجرا)

Lobby Settings

Password:

aaaa

Game Mode:Mensch

↑
↓

Max Players:4

↑
↓

Owner: aaaa3[*ME*] (17)

↑
↓

Specter:

✓

Voice Chat:

✓

Text Chat:

✓

Cancel

Submit

Destory

تصویر ۴-۶ 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