

فهرست مطالب

صفحه

عنوان

مقدمه..... ۱

فصل اول: مقدمات

۱-۱- برنامه‌نویسی رویداد محور ۳

۱-۲- زبان برنامه‌نویسی C++ ۴

۱-۳- نرم‌افزار Qt creator ۵

۱-۴- پرتکل لایه انتقال UDP ۵

۱-۵- سیستم مدیریت پایگاه داده ۸

فصل دوم: طراحی

۲-۱- طراحی پرتکل ارتباطی..... ۱۱

۲-۲- طراحی کلاس‌ها..... ۱۶

فصل سوم: پیاده‌سازی

۳-۱- طراحی کلاس مختصات..... ۲۱

۳-۲- طراحی کلاس کرم..... ۲۲

۳-۳- طراحی کلاس کاربر..... ۳۳

۳-۴- طراحی کلاس‌های دیوار و زمین بازی..... ۳۵

۳-۵- طراحی کلاس بازی..... ۳۵

۳-۶- طراحی کلاس سرور..... ۳۹

فصل چهارم: نتیجه‌گیری و پیشنهادات

۴-۱- نتیجه‌گیری..... ۴۵

۴-۲- پیشنهادات..... ۴۵

فهرست شکل‌ها

صفحه

عنوان

فصل اول: مقدمات

۱-۱- سرآیند پرتکل UDP ۶

فصل دوم: طراحی

۱-۲- ساختار پیام‌های نوع C و a ارسال شده توسط کاربر ۱۲

۲-۲- پیام نوع C از نوع اول ارسال شده توسط سرور ۱۲

۳-۲- پیام نوع C از نوع دوم ارسال شده توسط سرور ۱۲

۴-۲- پیام نوع a از نوع اول ارسال شده توسط سرور ۱۳

۵-۲- پیام نوع a از نوع دوم ارسال شده توسط سرور ۱۳

۶-۲- پیام نوع p ۱۳

۷-۲- پیام نوع g ۱۴

۸-۲- پیام نوع n ۱۵

۹-۲- پیام نوع S از نوع اول ۱۵

۱۰-۲- پیام نوع S از نوع دوم ۱۵

۱۱-۲- پیام نوع S از نوع سوم ۱۶

۱۲-۲- نمودار کلاس مختصات ۱۶

۱۳-۲- نمودار کلاس کرم ۱۷

۱۴-۲- نمودار کلاس کاربر ۱۸

۱۵-۲- نمودار کلاس دیوار ۱۸

۱۶-۲- نمودار کلاس زمین بازی ۱۸

۱۷-۲- نمودار کلاس بازی ۱۹

۱۸-۲- نمودار کلاس سرور ۲۰

فصل سوم: پیاده‌سازی

- ۳-۱- نحوه پیاده‌سازی کلاس مختصات..... ۲۱
- ۳-۲- نحوه‌ی تعریف کلاس کرم در فایل سرآیند worm.h..... ۲۲
- ۳-۳- حالتی که راستای سر کرم بازیکن افقی و جهت حرکت وارد شده بالا یا پایین باشد..... ۲۳
- ۳-۴- راستای سر کرم بازیکن افقی، جهت حرکت راست یا چپ و تعداد نقاط شکستگی صفر است..... ۲۴
- ۳-۵- راستای کرم بازیکن افقی، جهت حرکت راست یا چپ و تعداد نقاط شکستگی حداقل یک است..... ۲۴
- ۳-۶- حالتی که راستای سر کرم عمودی و جهت حرکت راست یا چپ است..... ۲۵
- ۳-۷- راستای سر کرم عمودی، جهت حرکت بالا یا پایین و تعداد نقاط شکستگی کرم صفر است..... ۲۵
- ۳-۸- راستای سر کرم عمودی ، جهت حرکت بالا یا پایین و تعداد نقاط شکستگی حداقل یک است..... ۲۶
- ۳-۹- حالتی که طول کرم بیش از حد آستانه و کرم فاقد نقطه‌ی شکستگی است..... ۲۷
- ۳-۱۰- طول کرم بیش از حد آستانه، کرم حداقل دارای یک نقطه شکستگی و جهت سر به سمت راست است... ۲۸
- ۳-۱۱- طول کرم بیش از حد آستانه، کرم حداقل دارای یک نقطه شکستگی و جهت سر به سمت چپ است... ۲۹
- ۳-۱۲- طول کرم بیش از حد آستانه، کرم حداقل دارای یک نقطه شکستگی و جهت سر به سمت پایین است.. ۲۹
- ۳-۱۳- طول کرم بیش از حد آستانه، کرم حداقل دارای یک نقطه شکستگی و جهت سر به سمت بالا است..... ۳۰
- ۳-۱۴- حالتی از تابع `increase_lenght()` که راستای سر کرم افقی است..... ۳۱
- ۳-۱۵- حالتی از تابع `increase_lenght()` که راستای سر کرم عمودی است..... ۳۱
- ۳-۱۶- کد تابع `move_tail()`..... ۳۲
- ۳-۱۷- کد تابع `self_smash()`..... ۳۳
- ۳-۱۸- نحوه تعریف کلاس کاربر در فایل سرآیند `user.h`..... ۳۴
- ۳-۱۹- کد تابع `creat_worm()`..... ۳۴
- ۳-۲۰- نحوه‌ی تعریف کلاس‌های دیوار و زمین بازی در فایل سرآیند `playground.h`..... ۳۵
- ۳-۲۱- کد تابع `add_user()`..... ۳۵
- ۳-۲۲- نحوه‌ی تعریف کلاس بازی در فایل سرآیند `game.h`..... ۳۶
- ۳-۲۳- کد تابع `move()`..... ۳۷
- ۳-۲۴- قسمتی از کد تابع `calculate_state()`..... ۳۸

- ۳۹-۲۵-۳ نحوه‌ی تعریف کلاس سرور در فایل سرآیند `server.h` ۳۹
- ۴۰-۲۶-۳ قسمتی از `ready_read()` که در اثر دریافت پیام نوع `c` اجرا می‌شود..... ۴۰
- ۴۱-۲۷-۳ قسمتی از `ready_read()` که در اثر دریافت پیام نوع `a` اجرا می‌شود..... ۴۱
- ۴۲-۲۸-۳ قسمتی از `ready_read()` که در اثر دریافت پیام نوع `p` اجرا می‌شود..... ۴۲
- ۴۲-۲۹-۳ قسمتی از `ready_read()` که در اثر دریافت پیام نوع `n` اجرا می‌شود..... ۴۲
- ۴۳-۳۰-۳ کد تابع `send_state()` ۴۳
- ۴۴-۳۱-۳ قسمتی از کد تابع `add_new_game` که برای تک‌تک بازیکنان پیام نوع `g` ارسال می‌کند..... ۴۴

مقدمه

بازی کرم‌ها یک بازی قدیمی است و الان هم نسخه‌های زیادی از آن چه به صورت آفلاین و چه به صورت آنلاین وجود دارد. قاعده بازی به این صورت است که بازیکنان در یک زمین بازی تعریف شده جمع می‌شوند و به هر بازیکن یک کرم با طول اولیه مشخص اختصاص داده می‌شود. بازیکنان می‌توانند در هر لحظه کرم خود را در یکی از چهار جهت بالا، راست، پایین یا چپ حرکت دهند. در زمین بازی نقاطی با نام «نقاط افزایشی» به صورت پراکنده وجود دارند. بازیکنان می‌توانند با قرار دادن سر کرم خود بر روی نقاط افزایشی – اصطلاحاً گفته می‌شود که کرم بازیکن نقطه افزایشی را خورده است – طول کرم خود افزایش دهند. همچنین اگر سر کرم یک بازیکن به بدن کرم بازیکن دیگر یا دیوارهای حاشیه‌ای زمین بازی برخورد کند، طول کرم بازیکن برخورد کننده به اندازه مشخصی کاهش پیدا می‌کند. در صورتی طول کرم بازیکن از حد مشخصی کمتر شود، بازیکن بازنده به حساب می‌آید و باید اقدام به شروع بازی جدید کند. در همه‌ی این نسخه‌ها یک انسان کرم خود را کنترل می‌کند. ولی در این پروژه هدف این است که یک رباط بجای انسان کنترل کرم را برعهده بگیرد و بیشتر جامعه برنامه‌نویسان را هدف گرفته است. به عبارتی در این پروژه پرتکلی معرفی می‌شود که برنامه‌نویسان می‌توانند طبق آن و با بهره‌گیری از تکنیک‌ها و مفاهیم هوش مصنوعی رباط بازیکن خود را بسازند و بازی خود را شخصی‌سازی کنند. این پرتکل توضیح می‌دهد که در هر مرحله رباط بازیکن باید چه پیامی با چه ساختاری را برای سرور ارسال کند و در جواب آن منتظر چه پیامی با چه ساختاری از جانب سرور باشد.

در راستای تکمیل پروژه ابتدا پرتکل ارتباطی را طراحی کردیم. سپس با بهره‌گیری از متدهای طراحی شی‌گرا کلاس‌های طراحی را شناسایی کردیم. در مرحله بعد برای هر کلاس طراحی ویژگی‌ها و عملیات‌هایش را طراحی نمودیم. بعد از طی مرحله طراحی نوبت به مرحله پیاده‌سازی می‌رسد. در مرحله پیاده‌سازی یکی یکی کلاس‌های طراحی شده در مرحله قبل را پیاده‌سازی کردیم. بعد از اتمام پیاده‌سازی برنامه سرور، یک بازیکن اتوماتیک ساده با الگوی حرکتی تصادفی طراحی و پیاده‌سازی نمودیم.

خروجی‌های پروژه عبارتند از: برنامه سرور بازی برخط کرم‌ها با قابلیت اتصال بازیکن اتوماتیک، یک بازیکن اتوماتیک ساده با الگوی حرکتی تصادفی، پرتکل ارتباطی که نحوه ارتباط بین سرور و بازیکن اتوماتیک و

پیام‌های انتقالی بینشان را نشان می‌دهد، مستندات طراحی مانند نمودار کلاس‌ها و گزارشی نهایی پروژه که در قالب پایان نامه تهیه می‌شود.

در فصل اول در مورد تکنولوژی‌های به کاربرده شده شامل برنامه‌نویسی مبتنی بر رویداد، زبان برنامه نویسی C++، محیط توسعه برنامه Qt creator، پرتکل لایه انتقال UDP و سیستم پایگاه داده SQL Server به صورت مختصر توضیح می‌دهیم. در فصل دوم به طراحی پرتکل ارتباطی و کلاس‌ها می‌پردازیم. در فصل سوم کلاس‌های طراحی شده در فصل دوم را پیاده‌سازی می‌کنیم و در نهایت در فصل چهارم نتیجه‌گیری می‌کنیم و پیشنهادهایی را ارائه می‌دهیم.

فصل اول: مقدمات

در این فصل در مورد هر یک از تکنولوژی‌های استفاده شده برای توسعه پروژه توضیحاتی ارائه می‌دهیم. برنامه سرور به روش برنامه‌نویسی مبتنی بر رویداد، با استفاده از زبان برنامه‌نویسی C++ و در محیط توسعه نرم‌افزار Qt Creator توسعه پیدا می‌کند. به منظور برقراری ارتباطات تحت شبکه از پرتکل لایه انتقال UDP و برای حفظ اطلاعات کاربران و احراز اصالت آن‌ها از سیستم پایگاه داده SQL Server استفاده می‌کنیم. در ادامه در مورد هر یک از موارد ذکر شده توضیحات مختصری ارائه می‌دهیم.

۱-۱- روش برنامه‌نویسی مبتنی بر رویداد

در برنامه‌نویسی کامپیوتر، برنامه‌نویسی مبتنی بر رویداد یک الگو و روش برنامه‌نویسی است که در آن جریان برنامه بوسیله‌ی رویدادها هدایت می‌شود. یک رویداد می‌تواند اقدامات کاربر مانند فشردن کلیک‌های موشواره یا فشار دادن دکمه‌های صفحه کلید، خروجی‌های حسگرها یا پیام‌هایی از برنامه‌ها و نخ‌های اجرایی دیگر باشد. در برنامه‌های رویداد محور، عموماً یک حلقه اصلی وجود دارد که منتظر رخداد رویدادها است. در صورت رخداد یک رویداد، حلقه اصلی تابع ثبت شده متناظر با رویداد را - عموماً به این تابع ثبت شده متناظر هر رویداد callback function گفته می‌شود- فراخوانی می‌کند و کنترل نخ اجرایی به این تابع سپرده می‌شود^[۱]. پس از پایان یافتن اجرای تابع کنترل به حلقه اصلی بازگردانده می‌شود و دوباره حلقه اصلی همین روند را برای بقیه رویدادهای منتظر در صف اجرا می‌کند.

در محیط توسعه نرم‌افزار QT Creator برنامه‌نویسی مبتنی بر رویداد بوسیله جفت‌های سیگنال و اسلات پیاده‌سازی می‌گردد. هر کلاس بسته به عملکردش در واکنش به رویدادهای مختلف سیگنال‌های متفاوتی را از خود صادر می‌کند. سیگنال صادر شده نشان می‌دهد که چه نوع رویدادی اتفاق افتاده است. برای مثال کلاس QPushButton -این کلاس برای برنامه‌نویسی گرافیکی استفاده می‌شود و یک دکمه را نشان می‌دهد- سیگنالی به نام clicked() دارد که در صورتی صادر می‌شود که کاربر بر روی دکمه در واسط گرافیکی کلیک

کند. بوسیله دستور `connect()` می‌توان سیگنال صادر شده از یک شی را به یک اسلات متصل کرد تا در صورت صادر شدن سیگنال مورد نظر از آن شی، اسلات مورد نظر اجرا شود. در واقع به این طریق به برنامه می‌فهمانیم که اسلات متناظر هر سیگنال چیست. پس می‌توان گفت اسلات در واقع همان `callback function` توضیح داده شده در قسمت قبل است.

برنامه‌نویسی مبتنی بر رویداد کاربرد زیادی در برنامه‌نویسی گرافیکی به خصوص پیاده‌سازی واسط گرافیکی دارد. همچنین از این روش در نوشتن برنامه درایورها استفاده می‌شود.

۱-۲- زبان برنامه‌نویسی C++

C++ یک زبان برنامه‌نویسی همه منظوره است و از برنامه‌نویسی رویه‌ای، تجرید داده‌ها و برنامه‌نویسی شی‌گرا پشتیبانی می‌کند. از قابلیت‌های این زبان می‌توان به انواع داده‌های ایستا و چند مدلی اشاره کرد. از آنجا که در این زبان می‌توان اشیاء را ابتدا به ساکن از کلاس‌هایی ایجاد کرد که به هیچ سلسله مراتب رده‌ها و وراثت مقید نیستند، لذا این زبان از برنامه‌نویسی مبتنی بر شیء نیز پشتیبانی می‌کند. این زبان هم از قابلیت‌های زبان‌های سطح بالا و هم از قابلیت‌های زبان‌های سطح پایین پشتیبانی می‌کند لذا می‌توان گفت که C++ یک زبان سطح میانی است^[۱].

C++ بیشتر برای برنامه‌نویسی سیستمی و سیستم‌های جاسازی شده، سیستم‌هایی با محدودیت منابع و سیستم‌های بزرگ طراحی شده است ولی می‌توان از آن در زمینه‌های دیگری مانند برنامه‌های کاربردی دسکتاپ و سرورها استفاده کرد. این زبان یک زبان کامپایل شده است و پیاده‌سازی‌های مختلفی از آن بر روی پلتفرم‌های مختلف وجود است^[۱].

C++ توسط بی‌یارنه‌استراستروپ ریاضیدان دانمارکی در سال ۱۹۷۹ در آزمایشگاه‌های بل، برای بهبود زبان C و بر مبنای آن ساخته شد. نام اولیه این زبان «C with classes» بود ولی بعداً در سال ۱۹۸۳ به C++ تغییر یافت. این زبان در سال‌های بعد با اضافه کردن کلاس‌ها و ویژگی‌های دیگری از قبیل توابع مجازی، سربرگزاری عملگرها، وراثت چندگانه، قالب توابع و پردازش استثناء توسعه پیدا کرد و سرانجام در سال ۱۹۹۸ تحت نام ISO/IEC ۱۴۸۸۲:۱۹۹۸ استاندارد گردید. استاندارد فعلی این زبان ISO/IEC ۱۴۸۸۲:۲۰۱۴ است^[۱].

در کتاب «طراحی و تکامل C++» استراستروپ قوانین مورد استفاده در طراحی C++ را بیان می‌کند. دانستن این قوانین با ما کمک می‌کند نحوه عملکرد C++ را درک کنیم و به چرایی آن پی ببریم. از جمله این قوانین می‌توان به موارد زیر اشاره کرد^[۱]:

- C++ طراحی شده است تا یک زبان عمومی با کنترل نوع ایستا و همانند C پربازده و قابل حمل باشد.
- C++ طراحی شده است تا مستقیماً و به صورت جامع از چندین شیوه برنامه‌نویسی از قبیل برنامه‌نویسی ساخت‌یافته، برنامه‌نویسی شی‌گرا، انتزاع داده و برنامه‌نویسی جنریک پشتیبانی کند.
- C++ طراحی شده است تا به برنامه‌نویس امکان انتخاب دهد حتی اگر این انتخاب اشتباه باشد.
- C++ طراحی شده تا حداکثر تطابق با C وجود داشته باشد و یک انتقال راحت از C را ممکن سازد.
- C++ از به کار بردن ویژگی‌های خاص که مانع از عمومی شدن است خودداری می‌نماید.
- C++ از ویژگی‌هایی که به کار برده نمی‌شوند استفاده نمی‌کند.
- C++ طراحی شده است تا بدون یک محیط پیچیده عمل کند.

۳-۱- محیط توسعه نرم‌افزار Qt Creator

Qt مجموعه‌ای از کتابخانه‌ها و سرآیندهای نوشته شده به زبان C++ است که به برنامه‌نویس امکان توسعه آسان نرم‌افزارهای کاربرد را می‌دهد. Qt شامل چندین کلاس برای کار با واسط گرافیکی، چندرسانه‌ای، ابزارهای پایگاه داده، شبکه و... است. می‌توان نرم‌افزارهای نوشته شده با این ابزار را با استفاده از کامپایلر زبان C++ برای طیف وسیعی از سیستم‌های عامل از جمله گنو/لینوکس، ویندوز، ویندوز CE، MAC OS و... همگردانی کرد. بدین ترتیب حمل نرم‌افزار نوشته شده بدون تغییر در متن کد نوشته شده امکان‌پذیر است. از Qt می‌توان در زبان‌های برنامه‌نویسی متعددی مانند C++، java و پایتون استفاده کرد^[۲].

Qt یک فریم ورک مولتی پلتفرم برای توسعه نرم‌افزار می‌باشد که اکثراً برای ایجاد برنامه‌هایی با رابط کاربری گرافیکی مورد استفاده قرار می‌گیرد. اما پس از نسخه ۴ امکان ایجاد برنامه‌های متنی نیز فراهم شده است. بیشترین استفاده از Qt در رابط گرافیکی KDE بوده است که یکی از مهمترین محیط‌های گرافیکی لینوکس می‌باشد. نرم‌افزارهای بسیاری چون Qtopia، Skype، Google Earth، Opera و... نیز توسط این نرم‌افزار ایجاد گردیده‌اند. Qt تقریباً در اکثر سیستم عامل‌های موجود چون لینوکس، ویندوز، مک و سیستم‌های خاصی چون PDA ها و تلفن‌های هوشمند قابل اجراست^[۲].

۴-۱- پرتکل لایه انتقال UDP

مجموعه پرتکل‌های اینترنت، از یک پرتکل انتقال بی‌اتصال به نام UDP استفاده می‌کنند. UDP کوتاه شده‌ی عبارت User Datagram Protocol به معنای پرتکل دیتاگرام کاربر می‌باشد. UDP این امکان را فراهم کرده است که برنامه‌های کاربردی بتوانند بدون ایجاد هرگونه اتصال قبلی، داده‌ها را دیتاگرام‌های IP جاسازی

و ارسال کنند^[۳]. همچنین این پرتکل دیتاگرام‌های دریافتی را تصدیق نمی‌کند، پس می‌توان گفت UDP برخلاف TCP یک پرتکل نامطمئن است. UDP در RFC ۷۶۸ توصیف شده است.

UDP قطعاتی را ارسال می‌کند که شامل سرآیند ۸ بیتی و فیلد داده‌ها است. سرآیند UDP در شکل ۱-۱ نشان داده شده است. دو فیلد پورت برای شناسایی نقاط پایانی (فرآیندها) در داخل ماشین‌های مبدا و مقصد به کار می‌روند. وقتی یک بسته UDP به مقصد می‌رسد، فیلد داده‌ی آن به فرآیند متصل به پورت مقصد تحویل داده می‌شود. اتصال فرآیند به پورت در اجرای عملیات پایه Bind (به معنای مقید کردن) صورت می‌گیرد. پورت‌ها را مانند صندوق‌های پستی در نظر بگیرید که که برنامه‌های کاربردی می‌توانند برای دریافت بسته‌ها آن‌ها را اجاره کنند. در حقیقت امتیاز اصلی UDP نسبت به IP خام، اضافه کردن فیلد پورت‌های مبدا و مقصد است. بدون فیلدهای پورت، لایه انتقال نمی‌داند با هر بسته ورودی چه کاری انجام دهد. با استفاده از فیلدهای پورت، قطعه دریافتی را به برنامه کاربردی مناسب تحویل می‌دهد^[۳].

Source port (پورت مبدا، ۲ بایت)	Destination port (پورت مقصد، ۲ بایت)
UDP length (۲ بایت)	UDP checksum (جمع تطبیقی، ۲ بایت)

شکل ۱-۱- سرآیند پرتکل UDP

برای این که پاسخی به فرآیند مبدا بازگردانده شود، نیاز به پورت مبدا است. برای این منظور فرآیند موجود در ماشین مقصد، فیلد پورت مبدا از قطعه ورودی را در فیلد پورت مقصد قطعه خروجی کپی می‌کند و قطعه را برای ماشین مبدا ارسال می‌کند^[۳].

فیلد UDP length کل طول قطعه UDP را که شامل داده و سرآیند است ۸ بیتی است مشخص می‌کند. حداقل طول ۸ بایت است که سرآیند را دربرمی‌گیرد و حداکثر طول ۶۵۵۱۵ بایت است که کوچک‌تر از بزرگ‌ترین عددی است که در ۱۶ بیت جا می‌شود زیرا این محدودیت در بسته IP اعمال شده است^[۳].

فیلد اختیاری UDP checksum (جمع تطبیقی UDP) برای قابلیت اعتماد بیشتر فراهم شده است. این فیلد جمع تطبیقی سرآیندها، داده‌ها و یک شبه‌سرآیند مفهومی IP را محاسبه می‌کند. هنگام انجام این محاسبه، فیلد UDP checksum صفر است و چنانچه طول فیلد داده اندازه‌ی فردی از بایت‌ها باشد، یک بایت صفر به انتهای آن اضافه می‌شود. الگوریتم جمع تطبیقی کلمات ۱۶ بیتی را به روش متمم یک جمع می‌کند و متمم یک مجموع را نیز بدست می‌آورد. در نتیجه، وقتی گیرنده محاسبات را در کل قطعه انجام می‌دهد (با احتساب فیلد UDP checksum) نتیجه صفر خواهد بود. اگر جمع تطبیقی محاسبه نشود، مقدارش صفر است، زیرا اگر مقدار واقعی آن صفر باشد، کل فیلد با بیت‌های ۱ پر خواهد شد. اما عدم محاسبه این فیلد، منطقی نیست مگر این که کیفیت سرویس اهمیتی نداشته باشد^[۳].

شاید اشاره صریح به بعضی از کارهایی که UDP انجام نمی‌دهد، ارزشمند باشد. این پرتکل، کنترل جریان ازدحام را انجام نمی‌دهد و چنانچه قطعه خراب دریافت کند، درخواست ارسال دوباره آن را از مبدا نمی‌کند. تمام این کارها بر عهده فرآیندهای کاربر است^[۳]. تنها کاری که UDP انجام می‌دهد، فراهم کردن یک واسطه برای پرتکل IP و انجام عمل مالتی‌پالکسینگ/دی‌مالتی‌پلکسینگ قطعه بین چندین فرآیند با استفاده از پورت‌ها و تشخیص خطای انتها به انتها است. البته فقط خطا را تشخیص ولی در قبال بسته خطا دار عکس‌العملی از خود نشان نمی‌دهد.

در برنامه‌های کاربردی که نیاز به کنترل دقیقی روی جریان بسته‌های کنترل خطا یا زمان‌بندی دارند UDP کار چندانی انجام نمی‌دهد. یکی از مواردی که UDP مفید واقع می‌شود، در برنامه‌های کلاینت-سرور است. معمولاً کلاینت یک تقاضای کوتاه را به سرور می‌فرستد و انتظار دارد یک پاسخ کوتاه توسط سرور برگردانده شود. اگر تقاضا یا پاسخ از بین بروند، مهلت کلاینت به اتمام می‌رسد و دوباره شروع می‌کند. در این حالت نه تنها کد برنامه ساده است بلکه نسبت به پرتکل‌هایی که نیاز به تنظیمات اولیه دارند، مثل TCP، پیام‌های اندکی مبادله می‌شود^[۳].

یکی از برنامه‌های کاربردی که از UDP استفاده می‌کند DNS است. DNS به طور خلاصه به این صورت عمل می‌کند که وقتی برنامه‌ای می‌خواهد آدرس IP میزبانی مثلاً به نام www.cs.berkeley.edu را پیدا کند، یک بسته UDP که حاوی نام میزبان است به سرور DNS می‌فرستد. سرور DNS در پاسخ، یک بسته UDP را برمی‌گرداند که حاوی آدرس IP میزبان درخواستی است. در اینجا نیاز به تنظیمات قبلی برای ایجاد اتصال و سپس خاتمه اتصال نیست. فقط دو پیام در شبکه مبادله می‌شوند^[۳].

در برنامه‌نویسی تحت شبکه برای استفاده از پرتکل‌های لایه انتقال نیاز به مفهومی به نام سوکت داریم. هر سوکت ترکیبی از IP و پورت است. در پرتکل UDP هر سوکت با پورت و IP میزبان مشخص می‌شود ولی در TCP هر سوکت با پورت مبدا، IP مبدا، پورت مقصد و IP مقصد توصیف می‌گردد. برای هر سوکت TCP به محض ایجاد به صورت خودکار حافظه اختصاص داده می‌شود ولی اختصاص حافظه برای سوکت‌های UDP دستی است و توسط برنامه‌نویس انجام می‌شود زیرا TCP یک پرتکل جریان محور ولی UDP یک پرتکل دیتاگرام محور است. بر روی هر سوکت TCP تنها می‌توان برای یک میزبان داده ارسال کرد و هر بار برای ارسال داده نیاز نیست آدرس و پورت میزبان مورد نظر را مشخص نمود ولی بر روی هر سوکت UDP می‌توان برای میزبان‌های زیادی داده ارسال کرد و باید هر بار آدرس و پورت مقصد را تعیین نمود زیرا که TCP یک پرتکل متصل ولی UDP یک پرتکل بی‌اتصال است. همچنین UDP به دلیل بی‌اتصال بودنش برخلاف TCP از حالت ارسال داده به صورت چندپخشی پشتیبانی می‌کند. برای این کار کافی است به جالی IP تک‌پخشی از یک IP همه‌پخشی به عنوان آدرس مقصد استفاده کنیم.

۱-۵- سیستم مدیریت پایگاه داده

یک سیستم مدیریت پایگاه داده از مجموعه‌ای از داده‌های مرتبط به هم و چندین نرم‌افزار برای دسترسی به آن داده‌ها تشکیل شده است. مجموعه اطلاعات مرتبط به هم در پایگاه داده معمولاً اطلاعات مربوط به یک شرکت یا سازمان می‌باشد. هدف اصلی یک سیستم مدیریت پایگاه داده فراهم کردن راهی مناسب و کارآمد برای ذخیره و بازیابی اطلاعات است.

سیستم‌های پایگاه داده برای مدیریت حجمی بزرگی از اطلاعات طراحی می‌شوند. مدیریت داده‌ها شامل تعریف ساختار برای ذخیره‌سازی اطلاعات و فراهم کردن مکانیزم‌هایی برای دسترسی به اطلاعات است. بعلاوه، سیستم پایگاه داده می‌بایست امنیت اطلاعات ذخیره شده را در برابر خرابی‌های سیستم و یا تلاش برای دسترسی غیرمجاز تضمین کند. اگر داده بین چند کاربر به اشتراک گذاشته شود، سیستم پایگاه داده باید بتواند از وقوع نتایج غیرعادی جلوگیری کند.

امروزه سیستم‌های مدیریت پایگاه داده به طور گسترده استفاده می‌شوند. برای مثال شرکت‌ها می‌توانند برای نگهداری و مدیریت اطلاعات فروش از قبیل اطلاعات مشتری، محصول و خرید، اطلاعات حسابداری از قبیل اطلاعات پرداخت، قبض، مانده حساب، دارایی و دیگر اطلاعات حسابداری، اطلاعات منابع انسانی از قبیل اطلاعات کارمندان، حقوق، مالیات حقوق، سود و صدور چک، اطلاعات تولید از قبیل مدیریت زنجیره تامین، ردیابی و پیگیری ساخت محصولات در کارخانه، فهرست کالاهای موجود در انبارها و مغازه‌ها و سفارش آن‌ها و اطلاعات خرده‌فروشی‌های آنلاین از قبیل داده‌های مرتبط با فروش، دنبال کردن سفارش‌های آنلاین، ایجاد لیست‌های پیشنهادی و بدست آوردن آنلاین بازاریابی محصولات از سیستم‌های مدیریت پایگاه داده استفاده کنند.

قبل از معرفی تکنولوژی سیستم‌های مدیریت پایگاه داده از سیستم‌های پردازش فایل برای ذخیره کردن و مدیریت اطلاعات استفاده می‌کردند. این سیستم‌ها داده‌ها را به صورت رکورد در فایل‌های تحت کنترل سیستم عامل ذخیره می‌کردند و برای واکنشی رکوردها و یا افزودن رکوردها به فایل‌های مناسب برنامه‌های کاربردی مختلف ساخته می‌شد. به دلیل مشکلاتی از قبیل افزونگی داده‌ها و ناسازگاری، مشکل در دسترسی به اطلاعات، ایزوله کردن داده‌ها، مشکلات جامعیت، مشکلات حالت اتمی، ناهنجاری‌های ایجاد شده در اثر دسترسی همزمان و مشکلات امنیتی این سیستم‌ها منسوخ شدند و به جای آن‌ها سیستم‌های مدیریت پایگاه داده توسعه پیدا کردند.

پایگاه داده به مرور زمان و با توجه به اطلاعاتی که اضافه و حذف می‌شوند تغییر می‌کند. به مجموعه اطلاعات ذخیره شده در پایگاه داده در یک لحظه خاص نمونه می‌گویند. به کل پایگاه داده، شمای پایگاه داده گویند.

شماهای پایگاه داده به ندرت تغییر می‌کند یا اصلاً عوض نمی‌شوند. مفهوم شما و نمونه‌های پایگاه‌های داده را می‌توان از طریق مقایسه با یک برنامه نوشته شده در یک زبان برنامه‌نویسی درک کرد. شما پایگاه داده مانند تعاریف متغیرها (همراه با تعریف نوع مرتبط) در یک برنامه است. هر متغیر در هر لحظه مقدار مشخص و خاصی دارد. مقادیر متغیرهای یک برنامه در یک زمان خاص، مشابه یک نمونه از شما پایگاه داده است.

پایه اصلی ساختار یک پایگاه داده، مدل داده است. مدل داده‌ای مجموعه‌ای از ابزارهای ادراکی برای توصیف داده، روابط بین داده، معانی داده و سازگاری محدودیت‌هاست. یک مدل داده راهی برای توصیف طراحی پایگاه داده در سطوح فیزیکی، منطقی و نمایشی را فراهم می‌کند. انواع مدل‌های داده‌ای عبارتند از: مدل رابطه‌ای، مدل موجودیت رابطه، مدل داده‌ای مبتنی بر شی و مدل داده‌ای نیمه‌ساخت یافته. چون سیستم مدیریت پایگاه داده SQL Server از مدل رابطه‌ای استفاده می‌کند، توضیحات مختصری در مورد این مدل می‌دهیم.

مدل رابطه‌ای: مدل رابطه‌ای از مجموعه‌ای از جدول‌ها برای نمایش داده‌ها و ارتباطات بین آن‌ها استفاده می‌کند. هر جدول شامل چندین ستون است و هر ستون یک نام منحصر به فرد دارد. به یک جدول، رابطه نیز می‌گویند. مدل رابطه‌ای یک نمونه از مدل مبتنی بر رکورد است. دلیل نام‌گذاری این مدل به مدل مبتنی بر رکورد این است که پایگاه داده با انواع مختلف رکوردهای فرمت ثابت سازمان‌دهی شده است. هر جدول شامل تعدادی رکورد از یک نوع خاص است. هر نوع رکورد تعداد ثابتی از فیلدها یا ویژگی‌ها را تعریف می‌کند. ستون‌های جدول، ویژگی‌های نوع رکورد را نشان می‌دهند. مدل داده‌ای رابطه‌ای رایج‌ترین نوع از مدل داده‌ای است و اکثر سیستم‌های پایگاه داده کنونی بر روی این مدل بنا شده‌اند.

یک سیستم پایگاه داده یک زبان تعریف داده برای تعریف شما پایگاه داده و یک زبان مدیریت داده برای پرس‌وجو از پایگاه داده و بروزرسانی‌ها فراهم می‌کند. در عمل زبان‌های تعریف و مدیریت داده، دو زبان جدا از هم نیستند، بلکه آن‌ها بخش‌هایی از یک زبان پایگاه داده واحد مانند SQL را تشکیل می‌دهند.

زبان مدیریت داده، زبانی است که کاربران را قادر می‌سازد به داده‌ها دسترسی یابند و یا آن‌ها را به طور سازمان‌دهی شده توسط مدل داده‌ای مناسب دستکاری کنند. انواع دستکاری‌ها عبارتند از: بازیابی اطلاعات ذخیره شده در پایگاه داده، درج کردن اطلاعات جدید در پایگاه داده، حذف اطلاعات از پایگاه داده و تغییر اطلاعات ذخیره شده در پایگاه داده. این زبان‌ها اساساً به دو نوع رویه‌ای و اعلانی تقسیم می‌شوند. در زبان‌های مدیریت داده رویه‌ای، کاربر باید مشخص کند چه داده‌هایی را نیاز دارد و چگونه می‌خواهد داده‌ها را بدست آورد ولی در زبان‌های مدیریت داده اعلانی، کاربر تنها باید مشخص کند چه داده‌هایی را می‌خواهد، بدون این که لازم باشد مشخص کند چگونه می‌خواهد داده‌ها را بدست آورد.

پرس‌وجو یک دستور برای تقاضای بازیابی اطلاعات است. به بخشی از زبان مدیریت داده که شامل بازیابی اطلاعات است، زبان پرس‌وجو می‌گویند. معمولاً از عناوین زبان پرس‌وجو و زبان مدیریت داده به صورت مترادف استفاده می‌شود، اگرچه از لحاظ فنی نادرست است. چندین زبان پرس‌وجو برای پایگاه داده وجود دارند که به صورت تجاری یا آزمایشی در حال استفاده هستند. سیستم مدیریت پایگاه داده SQL Server از زبان مدیریت داده SQL استفاده می‌کند.

فصل دوم: طراحی

در این فصل ابتدا یک پرتکل ارتباطی برای پیام‌های انتقالی بین سرور و کلاینت طراحی خواهیم کرد. این پرتکل بیان می‌دارد که سرور و کلاینت در هر مرحله از بازی باید چه پیام‌هایی را با چه ساختاری برای یکدیگر ارسال کنند. در مرحله بعد کلاس‌های طراحی را برای برنامه سرور شناسایی و معرفی می‌کنیم و به بررسی روابط بین کلاس‌ها می‌پردازیم. در فصل پیاده‌سازی در مورد پیاده‌سازی کلاس‌های معرفی شده در این فصل مطالبی را بیان می‌داریم.

۲-۱- طراحی پرتکل ارتباطی

پرتکل به معنای قاعده و قانون است. پس پرتکل ارتباطی قواعد و قوانینی را بیان می‌دارد که لازم است طرفین برای برقراری ارتباط و در حین ارتباط طبق آن قواعد و قوانین رفتار کنند.

هر کاربر برای شروع یک بازی در مرحله اول باید احراز اصالت گردد. روش‌های احراز اصالت زیادی وجود دارد. ساده‌ترین روش احراز اصالت، احراز اصالت مبتنی بر کلمه عبور است. در این روش احراز اصالت هر کاربر دارای یک نام کاربری یکتا و یک کلمه عبور است. هر کاربر باید در مرحله اول نام کاربری و کلمه عبور خود را برای سرور ارسال کند. سرور با دریافت نام کاربری و کلمه عبور با استفاده از زبان پرس‌وجو SQL در پایگاه داده‌ی خود به دنبال کاربری با نام کاربری و کلمه عبور دریافتی می‌گردد. چنانچه سرور کاربر مورد نظر را در پایگاه داده خود پیدا کند پیامی مبتنی بر موفقیت‌آمیز بودن فرآیند احراز اصالت برای کاربر ارسال می‌کند و کاربر را در لیست کاربران احراز اصالت شده قرار می‌دهد. در غیر این صورت به در قالب پیامی مشخص به کاربر اعلام شود به دلیل اشتباه بودن نام کاربری یا کلمه عبور فرآیند احراز اصالت با شکست مواجه شده است.

برای اجرای فرآیند احراز اصالت دو پیام C و A را در نظر گرفته‌ایم. چنانچه کاربر بخواهد یک نام کاربری و کلمه عبور جدید برای خود ایجاد کند و با مشخصات جدید احراز اصالت شود باید از پیام نوع C استفاده کند ولی اگر کاربر بخواهد با نام کاربری و کلمه عبور از قبل ایجاد شده احراز اصالت گردد باید از پیام نوع A استفاده کند. ساختار پیام‌های نوع C و A در شکل ۲-۱ نشان داده شده است. بابت اول پیام نوع پیام را نشان می‌دهد که برای پیام نوع C برابر با کاراکتر «C» و برای پیام نوع A برابر با کاراکتر «A» می‌باشد.

بایت دوم نشان‌دهنده‌ی تعداد کاراکترهای نام کاربری است. مثلاً اگر یک نام کاربری شامل ۱۰ کاراکتر باشد مقدار این بایت برابر با ۱۰ است. از بایت سوم به بعد به اندازه‌ی تعداد کاراکترهای نام کاربری یکی یکی کاراکترهای نام کاربری قرار می‌گیرند. بعد از اتمام نام کاربری همین ساختار برای کلمه عبور در بایت‌های بعدی استفاده می‌شود. یعنی ابتدا تعداد کاراکترهای کلمه عبور و سپس خود کاراکترهای کلمه عبور.

نوع پیام (برای احراز اصالت مقدار این فیلد یا کاراکتر a و یا کاراکتر c است) - یک بایت
تعداد کاراکترهای نام کاربری - یک بایت
کاراکترهای نام کاربری - تعداد کاراکترهای نام کاربری بایت
تعداد کاراکترهای کلمه عبور - یک بایت
کاراکترهای کلمه عبور - تعداد کاراکترهای کلمه عبور بایت

شکل ۲-۱- ساختار پیام‌های نوع C و a که کاربر برای سرور به منظور انجام احراز اصالت ارسال می‌کند.

چنانچه سرور یک پیام نوع C دریافت کند، نام کاربری و کلمه عبور جدید را از پیام استخراج می‌کند و شروع به اجرای پرس‌جو بر روی پایگاه داده می‌کند. اگر در نتیجه جست‌وجو، کاربری با نام کاربری برابر با نام کاربری استخراج شده پیدا شود، سرور وضعیت خطای پیش‌آمده را در قالب یک پیام نوع C به کاربر اطلاع می‌دهد و از قبول نام کاربری جدید امتناع می‌کند. ساختار این نوع پیام در شکل ۲-۲ آمده است. ولی اگر در نتیجه جست‌وجو، کاربری با نام کاربری برابر با نام کاربری استخراج شده پیدا نشود، سرور نام کاربری و کلمه عبور جدید را به پایگاه داده خود اضافه می‌کند و کاربر را در لیست کاربران احراز اصالت شده قرار می‌دهد و یک پیام نوع C دیگر مبتنی بر موفقیت آمیز بودن عملیات احراز اصالت با استفاده از نام کاربری و کلمه عبور جدید برای کاربر ارسال می‌کند. ساختار این نوع پیام در شکل ۲-۳ آمده است.

کاراکتر c - یک بایت
کاراکتر f - یک بایت

شکل ۲-۲- پیام نوع C که سرور برای کاربر ارسال می‌کند و نشان‌دهنده‌ی شکست خوردن عملیات احراز اصالت با استفاده از نام کاربری و کلمه عبور جدید است.

کاراکتر c - یک بایت
کاراکتر p - یک بایت

شکل ۲-۳- پیام نوع C که سرور برای کاربر ارسال می‌کند و نشان‌دهنده‌ی موفقیت آمیز بودن عملیات احراز اصالت با نام کاربری و کلمه عبور جدید است.

در صورتی که سرور پیام نوع a دریافت کند بازهم همان کارهای قبلی را انجام می‌دهد با این تفاوت که در این حالت اگر سرور در پایگاه داده خود کاربری با نام کاربری و کلمه عبور برابر با نام کاربری و کلمه عبور دریافتی پیدا کند، عملیات احراز اصالت موفقیت‌آمیز است و سرور در قالب یک پیام نوع a موفقیت‌آمیز بودن عملیات را به کاربر اطلاع می‌دهد. در غیر این صورت در قالب یک پیام نوع a دیگر سرور به کاربر اعلام می‌کند که نام کاربری یا کلمه عبور ارسالی اشتباه بوده است. ساختار این دو نوع پیام نوع a ارسالی از جانب سرور در شکل‌های ۴-۲ و ۵-۲ آورده شده است.

کاراکتر a - یک بایت
کاراکتر p - یک بایت

شکل ۴-۲- پیام نوع a که سرور برای کاربر ارسال می‌کند و نشان‌دهنده‌ی موفقیت‌آمیز بودن عملیات احراز اصالت است

کاراکتر a - یک بایت
کاراکتر f - یک بایت

شکل ۵-۲- پیام نوع a که سرور برای کاربر ارسال می‌کند و نشان‌دهنده‌ی شکست خوردن عملیات احراز اصالت است.

بعد از این که کاربر با موفقیت احراز اصالت شد و در لیست کاربران احراز اصالت شده قرار گرفت، کاربر باید زمین بازی خود را انتخاب کند و انتخاب خود را در قالب پیام نوع p به اطلاع سرور برساند. ساختار این پیام در ۶-۲ آورده شده است. در بایت اول که مشخص‌کننده نوع پیام ارسالی است کاراکتر «p» قرار می‌گیرد. بایت دوم تعداد کاراکترهای نام کاربری را نشان می‌دهد. از بایت سوم به بعد کاراکترهای نام کاربری به صورت بایت به بایت پشت سر هم قرار می‌گیرند و در نهایت بایت آخر حاوی شناسه زمین بازی انتخابی کاربر است.

نوع پیام (برابر با کاراکتر p) - یک بایت
تعداد کاراکترهای نام کاربری - یک بایت
کاراکترهای نام کاربری - تعداد کاراکترهای نام کاربری بایت
شناسه زمین بازی انتخابی - یک بایت

شکل ۶-۲- پیام نوع p که کاربر برای سرور به منظور انتخاب زمین بازی ارسال می‌کند.

وقتی سرور یک پیام نوع p دریافت می‌کند، نام کاربری را از پیام استخراج می‌کند. سپس بررسی می‌کند که آیا کاربری با نام کاربری استخراج شده در لیست کاربران احراز اصالت شده وجود دارد یا نه. در صورت وجود، کاربر را از لیست کاربران احراز اصالت شده خارج و به لیست کاربران منتظر شروع بازی جدید در زمین انتخاب شده

اضافه می‌کند. چنانچه حداقل یک کاربر منتظر دیگر در لیست کاربران منتظر وجود داشته باشد، سرور یک بازی جدید با یک شناسه جدید منحصر به فرد ایجاد می‌کند، کرم‌های کاربران را مقداردهی اولیه می‌کند، به صورت تصادفی نقاط افزایشی را در سراسر زمین پراکنده می‌کند و در نهایت شناسه بازی، مختصات دم و سر کرم‌های بازیکنان و مختصات نقاط افزایشی را برای کاربران حاضر در زمین بازی در قالب پیام نوع g ارسال می‌کند. ساختار این پیام در شکل ۲-۷ نشان داده شده است. لازم به ذکر است برای هر بازیکن ابتدا اطلاعات کرم آن بازیکن و سپس اطلاعات کرم‌های بقیه بازیکنان قرار می‌گیرد. مثل بقیه پیام‌ها بایت اول نوع پیام را نشان می‌دهد و برابر با کاراکتر g است. بایت دوم حاوی شناسه بازی جدید است. بایت سوم نشان‌دهنده تعداد بازیکنان حاضر در زمین بازی است. تعداد بازیکنان یک بازی حداقل ۲ و حداکثر ۴ نفر است. از بایت چهارم به بعد برای هر بازیکن ساختار به این صورت است: ابتدا مختصات دم کرم بازیکن در قالب دو بایت (یک بایت برای مختص طولی و یک بایت برای مختص عرضی) و سپس مختصات سر کرم بازیکن در قالب دو بایت. این ساختار برای هر بازیکن تکرار می‌شود. بعد از اطلاعات کرم‌های بازیکنان، مختصات نقاط افزایشی پراکنده شده در زمین (هر نقطه افزایشی در قالب دو بایت) قرار داده می‌شوند.

نوع پیام (کاراکتر g) – یک بایت
شناسه بازی جدید – یک بایت
تعداد بازیکنان حاضر در زمین – یک بایت
مختصات دم کرم بازیکن اول – دو بایت
مختصات سر کرم بازیکن اول – دو بایت
.....
مختصات دم کرم بازیکن nام – دو بایت
مختصات سر کرم بازیکن nام – ۲ بایت
مختصات نقاط افزایشی – (تعداد نقاط افزایشی*۲) بایت

شکل ۲-۷- پیام نوع g که سرور بلافاصله بعد از شروع بازی جدید برای بازیکنان حاضر در زمین می‌فرستد.

پس از این که هر بازیکن پیام نوع g را از سرور دریافت کرد باید هر بار تا قبل از زمان مشخصی یک پیام نوع n برای سرور ارسال کند. این پیام حامل حرکت بعدی بازیکن است و ساختار آن در شکل ۲-۸ آورده شده است. بایت اول نوع پیام را نشان می‌دهد و برابر با کاراکتر n است. بایت دوم حاوی تعداد کاراکترهای نام کاربری است. از بایت سوم به بعد یکی یکی کاراکترهای نام کاربری قرار داده می‌شوند. بعد اتمام نام کاربری، بایت بعدی شناسه بازی را نشان می‌دهد و در نهایت بایت آخر حاوی حرکت بازیکن است. حرکت هر بازیکن می‌تواند یکی

از کاراکترهای r (حرکت به سمت راست)، d (حرکت به سمت پایین)، l (حرکت به سمت چپ) و u (حرکت به سمت بالا) باشد.

نوع پیام (کاراکتر n) - یک بایت
تعداد کاراکترهای نام کاربری - یک بایت
کاراکترهای نام کاربری - تعداد کاراکترهای نام کاربری بایت
شناسه بازی - یک بایت
حرکت بعدی - یک بایت

شکل ۲-۸-پیام نوع n که هر بازیکن باید برای تعیین حرکت بعدیش برای سرور ارسال کند.

سرور با دریافت پیام نوع n از هر بازیکن، کرم آن بازیکن را در جهت ارسال شده حرکت می‌دهد. در پایان اسلات‌های زمانی مشخصی، سرور وضعیت جدید زمین بازی و کرم‌های حاضر در آن را محاسبه می‌کند. یعنی برخورد کرم‌های بازیکنان به یکدیگر را در نظر می‌گیرد. مثلاً اگر سر کرم یک بازیکن به بدن کرم بازیکن دیگر برخورد کرده باشد، سرور طول وضعیت کرم برخورد کننده را با کاهش طولش به‌روز می‌کند. بعد از محاسبه وضعیت جدید، سرور اطلاعات به‌روز شده زمین و بازیکنان را در قالب پیام نوع S برای تک‌تک بازیکنان حاضر در زمین ارسال می‌کند. پیام نوع S به سه نوع مختلف تقسیم می‌شود. نوع اول و دوم دو بایت طول دارند و به ترتیب برای بازیکن برنده و بازیکنان بازنده ارسال می‌شوند. نوع سوم برای مابقی بازیکنان (بازیکنانی که هنوز بازی آن‌ها تمام نشده است) ارسال خواهد شد. ساختار پیام‌های S نوع اول، دوم و سوم به ترتیب در شکل‌های ۲-۹، ۲-۱۰ و ۲-۱۱ نشان داده شده است.

نوع پیام (کاراکتر S) - یک بایت
کاراکتر w - یک بایت

۲-۹-پیام S نوع اول که سرور برای بازیکن برنده ارسال می‌کند.

نوع پیام (کاراکتر S) - یک بایت
کاراکتر o - یک بایت

شکل ۲-۱۰-پیام S نوع دوم که سرور برای بازیکن‌های بازنده ارسال می‌کند.

نوع پیام (کاراکتر S) - یک بایت
تعداد بازیکنان حاضر در زمین - یک بایت
تعداد نقاط شکستگی کرم بازیکن اول - یک بایت
مختصات دم کرم بازیکن اول - دو بایت
مختصات نقاط شکستگی کرم بازیکن اول - (تعداد نقاط شکستگی کرم بازیکن اول*۲) بایت
مختصات سر کرم بازیکن اول - دو بایت
.
.
تعداد نقاط شکستگی کرم بازیکن n ^{ام} - یک بایت
مختصات دم کرم بازیکن n ^{ام} - دو بایت
مختصات نقاط شکستگی کرم بازیکن n ^{ام} - (تعداد نقاط شکستگی کرم بازیکن n ^{ام} *۲) بایت
مختصات سر کرم بازیکن n ^{ام} - دو بایت
مختصات نقاط افزایشی - (تعداد نقاط افزایشی*۲) بایت

شکل ۲-۱۱-پیام S نوع سوم که سرور برای بازیکنانی ارسال می کند که هنوز بازی آن ها تمام نشده است.

۲-۲-طراحی کلاس ها

در این بخش می خواهیم کلاس های تحلیل را شناسایی کنیم. اولین کلاس، کلاس مختصات است. برای مدل کردن مختصات عناصر مختلف موجود در زمین بازی مثل نقاط شکستگی بازیکن ها به این کلاس نیاز داریم. نمودار کلاس مختصات در شکل ۲-۱۲ نشان داده شده است. این کلاس فقط حاوی دو صفت Xposition برای نگهداری مختص طولی و Yposition برای نگهداری مختص عرضی می باشد.

coordinates
+Xposition: char
+Yposition: char

شکل ۲-۱۲-نمودار کلاس مختصات

برای هر بازیکن به محض شروع بازی، یک کرم در نظر گرفته می شود. پس یکی دیگر از کلاس های موجود، کلاس کرم است. شکل ۲-۱۳ نمودار کلاس کرم را نشان می دهد. صفت head نشان دهنده ی مختصات سر کرم است. صفت breaking_points حاوی نقاط شکستگی کرم و صفت tail نشان دهنده ی دم کرم است. صفت speed سرعت حرکت کرم را تعیین می کند و صفت های increase و decrease به ترتیب مشخص

می‌کنند که در صورت برخورد کرم به دیوارهای زمین یا کرم‌های دیگر چقدر از طول کرم کم می‌شود و در صورت خوردن نقطه‌ای افزایشی چقدر به طول کرم اضافه می‌شود. صفتهای `rasta_head`، `rasta_tail`، `length` و `initial_lenght` به ترتیب نشان‌دهنده‌ی طول کرم، راستای سر کرم (افقی یا عمودی)، راستای دم کرم و طول اولیه کرم هستند. عملیات `move(jahat)` کرم را در جهت `jahat` حرکت می‌دهد. در صورت برخورد به دیوارهای زمین یا بدن کرم دیگر، عملیات `decrease_lenght()` طول کرم را کاهش می‌دهد. از عملیات `increase_lenght()` برای افزایش طول کرم زمانی استفاده می‌شود که کرم یک نقطه‌ی افزایشی را بخورد. عملیات `move_tail()` دم کرم را حرکت می‌دهد و عملیات `self_smash()` بررسی می‌کند که آیا سر کرم با بدن خودش برخورد دارد یا خیر.

Worm
+ head: coordinates + breaking_points: vector<coordinates> + tail: coordinates + speed: short + increase: short + decrease: short + initial_lenght: short + rasta_head: char + rasta_tail: char + lenght: short
+ move(jahat) + increase_lenght() + decrease_lenght() + self_smash()

شکل ۲-۱۳-نمودار کلاس کرم

هر کرم به یک کاربر تعلق دارد. پس می‌توان گفت کلاس دیگری به نام کلاس کاربر داریم. نمودار این کلاس در شکل ۲-۱۴ نمایش داده شده است. صفتهای `ucn` و `pcn` به ترتیب تعداد کاراکترهای نام کاربری و کلمه عبور را نشان می‌دهند. صفتهای `username` و `password` به ترتیب به نام کاربری و کلمه عبور کاربر اشاره می‌کنند. صفتهای `w`، `address` و `port` به ترتیب نشان‌دهنده‌ی کرم اختصاصی کاربر، آدرس IP کاربر و پورت استفاده شده در ماشین کاربر هستند. تنها عملیات کلاس کاربر، عملیات `creat_worm()` است که یک کرم اختصاصی برای کاربر ایجاد می‌کند.

user
+ ucn: short
+ pcn: short
+ username: char *
+ password: char *
+ w: worm *
+ address: IPAddress
+ port: int
+ creat_worm(head,tail,rasta_head,rasta_tail)

شکل ۲-۱۴- نمودار کلاس کاربر

هر بازی در یک زمین بازی برگزار می‌شود و در هر زمین بازی ممکن است علاوه بر دیوارهای حاشیه‌ایی، دیوارهای دیگری در داخل خود زمین بازی وجود داشته باشد. پس می‌توان نتیجه گرفت دو کلاس دیگر به نام‌های کلاس دیوار و کلاس زمین بازی وجود دارد. کلاس دیوار در شکل ۲-۱۵ و کلاس زمین بازی در شکل ۲-۱۶ نمایش داده شده‌اند. کلاس دیوار تنها حاوی دو صفت `type` و `top_left` است که اولی نوع دیوار و دومی مختصات نقطه‌ی بالایی سمت چپ دیوار را نشان می‌دهد. کلاس زمین بازی فقط حاوی صفت `walls` است. این صفت لیستی از دیوارهای موجود در زمین را در خود جای می‌دهد.

Wall
+ type: short
+ top_left: coordinates

شکل ۲-۱۵- نمودار کلاس دیوار

playground
+ walls: vector<wall>

شکل ۲-۱۶- نمودار کلاس زمین بازی

هر کاربر در یک بازی شرکت می‌کند. پس می‌توان گفت کلاس دیگری به نام کلاس بازی خواهیم داشت. شکل ۲-۱۷ نمودار کلاس بازی را نشان می‌دهد. صفت `id` همان شناسه بازی است. صفت `users` کاربران حاضر در بازی را نشان می‌دهد. صفت `increasing_points` حاوی لیستی از مختصات نقاط افزایشی است که به صورت تصادفی در هنگام ایجاد بازی در سراسر زمین بازی پراکنده شده‌اند. صفت `ground` نشان می‌دهد این بازی

در کدام زمین بازی در حال انجام است. صفت `timer` برای زمان‌بندی برای محاسبه وضعیت جدید زمین بازی و کرم‌های حاضر در آن کاربرد دارد. عملیات `move_worm()` کرم کاربر با نام کاربری `username` را در جهت `jahat` حرکت می‌دهد و پس از انجام حرکت بررسی می‌کند که آیا کرم کاربر نقاط افزایشی را خورده است یا به دیوارهای درون و حاشیه زمین برخورد کرده است یا نه. عملیات `add_user()` کاربر جدیدی را به لیست کاربران حاضر در بازی اضافه می‌کند و عملیات `delete_user()` کاربری را از لیست کاربران حاضر در بازی حذف می‌کند. هر بار با تایم‌اوت شدن تایمر، عملیات `calculate_state()` اجرا می‌شود و وضعیت زمین و کرم‌های موجود در آن را به‌روزرسانی می‌کند.

Game
+ id: int + users: vector<user> + increasing_points: vector<coordinates> + timer: Timer + ground: playground *
+ move_worm(username,jahat) + add_user() + delete_user() + calculate_state()

شکل ۲-۱۷- نمودار کلاس بازی

تمامی بازی‌ها در یک سرور انجام می‌گیرد. پس می‌توان گفت کلاس دیگری به نام کلاس سرور داریم. نمودار کلاس سرور در شکل ۲-۱۸ قابل مشاهده است. صفت `games` حاوی لیستی از بازی‌های در حال اجرا است. صفت `authenticated_users` دربردارنده‌ی لیست کاربرانی است که با موفقیت احراز صلاحیت شده‌اند. صفت `socket` به سوکت UDP برای ارسال و دریافت داده‌ها روی شبکه و صفت `playgrounds` به زمین‌های بازی موجود در سرور اشاره دارد. صفت `waiting_for_playing` حاوی لیست کاربرانی است که احراز صلاحیت شده‌اند و منتظر شروع بازی جدید در زمین شماره یک (ساده ترین زمین که فقط دیوارهای حاشیه‌ای دارد) هستند. صفت `db` برای ارتباط با پایگاه داده و انجام عملیات احراز اصالت به کار می‌رود. زمان‌بندی برای شروع بازی جدید با استفاده از صفت `start_new_game` انجام می‌شود. صفت `highest_available_id` حاوی بزرگترین شناسه بازی قابل استفاده برای ایجاد یک بازی جدید و صفت `removed_game_ids` دربردارنده‌ی لیستی از شناسه بازی‌هایی است که به اتمام رسیده‌اند. هر بار که سرور بسته‌ایی را از شبکه دریافت می‌کند عملیات `ready_read()` برای تحلیل و پردازش بسته دریافتی و نشان دادن عکس‌العمل مناسب فراخوانی و اجرا می‌شود. در اثر تایم‌اوت شدن تایمر `start_new_game`، عملیات

`add_new_game()` فراخوانی و اجرا می‌شود. این عملیات در صورت وجود حداقل ۲ کاربر منتظر برای شروع بازی جدید، یک بازی جدید برای آن کاربران منتظر ایجاد می‌کند و مقداردهی‌های اولیه را انجام می‌دهد. هر بار که تایمر یکی از بازی‌های در حال انجام تایم‌اوت شود، عملیات `send_state()` فراخوانی می‌شود. این عملیات وضعیت جدید را برای تک‌تک بازیکنان حاضر در بازی با شناسه `id_game` ارسال می‌کند.

Server
+ games: vector<game *> + authenticated_users: vector<user> + _socket: UDPsocket + playgrounds: playground * + waiting_for_playing: vector<user> + db: SQLiteDatabase + start_new_game: Timer + the_highest_availble_id: int + removed_game_ids: vector<int>
+ ready_read() + send_state(id_game) + add_new_game()

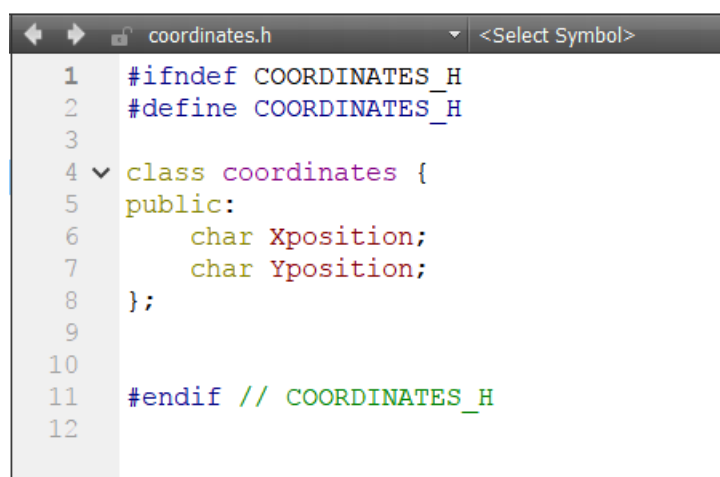
شکل ۲-۱۸- نمودار کلاس سرور

فصل سوم: پیاده‌سازی

در این فصل در مورد پیاده‌سازی تک‌تک کلاس‌های طراحی شده در فصل قبل توضیحاتی را ارائه خواهیم داد. برنامه سرور با استفاده از زبان برنامه‌نویسی C++ و در محیط توسعه نرم‌افزار Qt Creator توسعه پیدا کرده است به همین دلیل برای درک نحوه پیاده‌سازی کلاس‌ها آشنایی اولیه‌ای با زبان برنامه‌نویسی C++، محیط توسعه Qt و نحوه‌ی استفاده از روش برنامه‌نویسی مبتنی بر رویداد در Qt لازم است.

۳-۱- پیاده‌سازی کلاس مختصات

در بخش طراحی گفته شد از این کلاس برای مدل کردن صفحه مختصات دوبعدی استفاده خواهیم کرد. این کلاس فاقد هیچ‌گونه عملیات است و فقط دو صفت `Xposition` و `Yposition` را شامل می‌شود که اولی نشان‌دهنده‌ی محور Xها و دومی نشان‌دهنده‌ی محور Yها است. شکل ۳-۱ پیاده‌سازی این کلاس با استفاده از زبان C++ در محیط Qt را نشان می‌دهد.



```
1 #ifndef COORDINATES_H
2 #define COORDINATES_H
3
4 class coordinates {
5 public:
6     char Xposition;
7     char Yposition;
8 };
9
10
11 #endif // COORDINATES_H
12
```

شکل ۳-۱- نحوه پیاده‌سازی کلاس مختصات

۳-۲- پیاده‌سازی کلاس کرم

کلاس کرم، نماینده کرم هر بازیکن است و باید با استفاده از صفتهای خود، کارهایی نظیر حرکت دادن سر و دم کرم، کاهش طول کرم در صورت برخورد به دیوارهای حاشیه‌ایی یا درون زمین، بدن کرم‌های دیگر یا بدن خود کرم و افزایش طول کرم در صورت خوردن نقاط افزایشی را انجام دهد. شکل ۳-۲ نحوه‌ی تعریف این کلاس را در فایل سرآیند worm.h نشان می‌دهد.

```

1  #include <coordinates.h>
2  #include <vector>
3  using namespace std;
4  #ifndef WORM_H
5  #define WORM_H
6
7  class worm{
8  public:
9      coordinates head;
10     vector<coordinates> breaking_points ;
11     coordinates tail ;
12     short speed ;
13     short increase ;
14     short decrease ;
15     short initial_lenght ;
16     char rasta_head ;
17     char rasta_tail ;
18     short lenght;
19
20     worm(coordinates head,coordinates tail,char rasta_h,char rasta_t);
21     ~worm();
22 public:
23
24     void move(char jahat);
25     void increase_lenght();
26     void decrease_lenght();
27     void move_tail();
28     void self_smash();
29 };
30
31 #endif // WORM_H
32

```

شکل ۳-۲- نحوه‌ی تعریف کلاس کرم در فایل سرآیند worm.h

تابع `move(char jahat)` کرم بازیکن را در جهت `jahat` به اندازه `speed` حرکت می‌دهد. نحوه‌ی عملکرد این تابع به این صورت است: ابتدا راستای سر کرم بازیکن بررسی می‌شود و با توجه به راستای سر و جهت حرکت وارد شده به تابع مختصات سر و دم کرم دستکاری می‌شوند و ممکن است نقاط شکستگی کرم افزایش یا کاهش یابند. چنانچه راستای سر کرم افقی و جهت حرکت وارد شده بالا یا پایین باشد، راستای سر کرم عمودی می‌شود، یک نقطه شکستگی جدید به لیست نقاط شکستگی کرم بازیکن اضافه می‌شود، مختص عرضی سر کرم بازیکن به اندازه `speed` کاهش یا افزایش می‌یابد و درنهایت تابع `move_tail()` برای تعیین وضعیت

جدید دم کرم صدا زده می‌شود. اگر راستای سر کرم افقی و جهت حرکت وارد شده راست یا چپ باشد دو حالت پیش می‌آید؛ اگر کرم نقطه‌ی شکستگی نداشته باشد کافی است مختص طولی سر و دم کرم افزایش یا کاهش یابد ولی اگر کرم حداقل یک نقطه شکستگی داشته باشد، باید تنها مختص طولی سر کرم دستکاری شود و تابع `move_tail()` برای تعیین وضعیت جدید دم کرم فراخوانی گردد. در صورتی که راستای سر کرم عمودی و جهت حرکت وارد شده راست یا چپ باشد، راستای سر کرم افقی می‌شود، یک نقطه‌ی شکستگی جدید به لیست نقاط شکستگی کرم اضافه می‌شود، مختص طولی سر کرم به اندازه‌ی `speed` افزایش یا کاهش می‌یابد و تابع `move_tail()` برای دستکاری مختصات دم کرم صدا زده می‌شود. ولی اگر راستای سر کرم عمودی و جهت حرکت بالا یا پایین باشد، دو حالت پیش می‌آید؛ چنانچه تعداد نقاط شکستگی کرم بازیکن صفر باشد تنها کافی است مختص عرضی سر و دم کرم به اندازه‌ی `speed` کاهش یا افزایش یابد ولی اگر کرم بازیکن حداقل یک نقطه‌ی شکستگی داشته باشد مختصات سر کرم دستکاری می‌شود و وضعیت جدید دم با فراخوانی تابع `move_tail()` مشخص می‌گردد. شکل ۳-۳ حالتی را نشان می‌دهد که راستای سر بازیکن افقی و حرکت وارد شده بالا یا پایین است.

```
void worm::move(char jahat)
{
    if(rasta_head == 'h')
    {
        if(jahat == 'u' || jahat == 'd')
        {
            rasta_head='v';
            int breaking_points_number =breaking_points.size();
            breaking_points.resize(breaking_points_number +1);
            breaking_points_number+=1;
            for(int i=breaking_points_number-1;i>0;i--)
            {
                breaking_points[i]=breaking_points[i-1];
            }
            breaking_points[0]=head ;
            move_tail();
            if(jahat == 'u')
            {
                cout << "1"<<endl ;
                head.Yposition =head.Yposition- speed ;
            }
            else
            {
                head.Yposition += speed ;
            }
        }
    }
}
```

شکل ۳-۳- حالتی که راستای سر کرم بازیکن افقی و جهت حرکت وارد شده بالا یا پایین باشد.

حالتی که در آن سر کرم بازیکن افقی، جهت حرکت وارد شده راست یا چپ و تعداد نقاط شکستگی بازیکن صفر است، در شکل ۳-۴ نشان داده شده است.

```

else
{
    if(breaking_points.size() == 0)
    {
        if(head.Xposition > tail.Xposition)
        {
            if(jahat == 'r')
            {
                head.Xposition += speed;
                tail.Xposition += speed;
            }
            else
            {
                decrease_lenght();
            }
        }
        else
        {
            if(jahat == 'l')
            {
                head.Xposition -= speed;
                tail.Xposition -= speed;
            }
            else
            {
                decrease_lenght();
            }
        }
    }
}

```

شکل ۳-۴- راستای سر کرم بازیکن افقی، جهت حرکت راست یا چپ و تعداد نقاط شکستگی صفر است.

حالتی که در آن راستای سر کرم بازیکن افقی و جهت حرکت وارد شده راست یا چپ است و کرم بازیکن حداقل یک نقطه شکستگی دارد در شکل ۳-۵ آورده شده است.

```

else
{
    coordinates first_breaking_point = breaking_points[0];
    if(head.Xposition > first_breaking_point.Xposition)
    {
        if(jahat == 'r')
        {
            head.Xposition += speed;
            move_tail();
        }
        else
        {
            decrease_lenght();
        }
    }
    else
    {
        if(jahat == 'l')
        {
            head.Xposition -= speed;
            move_tail();
        }
        else
        {
            decrease_lenght();
        }
    }
}
}

```

شکل ۳-۵- راستای کرم بازیکن افقی، جهت حرکت راست یا چپ و تعداد نقاط شکستگی حداقل یک است.

شکل ۳-۶ حالتی را نشان می‌دهد که راستای سر کرم بازیکن عمودی و جهت حرکت راست یا چپ است.

```
if(jahat == 'r' || jahat== 'l')
{
    rasta_head='h';
    int breaking_points_number =breaking_points.size();
    breaking_points.resize(breaking_points_number +1);
    breaking_points_number+=1;
    for(int i=breaking_points_number-1;i>0;i--)
    {
        breaking_points[i]=breaking_points[i-1];
    }
    breaking_points[0]=head ;
    move_tail();
    if(jahat == 'l')
    {
        head.Xposition -= speed ;
    }
    else
    {
        head.Xposition += speed ;
    }
}
```

شکل ۳-۶- حالتی که راستای سر کرم عمودی و جهت حرکت راست یا چپ است.

اما حالتی که در آن راستای سر کرم بازیکن عمودی، جهت حرکت بالا یا پایین و تعداد نقاط شکستگی کرم بازیکن صفر است در شکل ۳-۷ نشان داده شده است.

```
if(breaking_points.size()==0)
{
    if(head.Yposition < tail.Yposition)
    {
        if(jahat == 'u')
        {
            head.Yposition -= speed;
            tail.Yposition -= speed;
        }
        else
        {
            decrease_lenght();
        }
    }
    else
    {
        if(jahat == 'd')
        {
            head.Yposition += speed;
            tail.Yposition += speed;
        }
        else
        {
            decrease_lenght();
        }
    }
}
```

۳-۷- راستای سر کرم عمودی، جهت حرکت بالا یا پایین و تعداد نقاط شکستگی کرم صفر است.

آخرین حالت ممکن یعنی حالتی که در آن راستای سر کرم عمودی و جهت حرکت بالا یا پایین است و کرم حداقل یک نقطه شکستگی دارد در شکل ۳-۸ آمده است.

```
coordinates first_breaking_point = breaking_points[0];
if(head.Yposition< first_breaking_point.Yposition)
{
    if(jahat== 'u')
    {
        head.Yposition -= speed ;
        move_tail();
    }
    else
    {
        decrease_lenght();
    }
}
else
{
    if(jahat == 'd')
    {
        head.Yposition += speed ;
        move_tail();
    }
    else
    {
        decrease_lenght();
    }
}
```

شکل ۳-۸-حالتی که راستای سر کرم عمودی و جهت حرکت بالا یا پایین است و کرم حداقل یک نقطه شکستگی دارد.

تابع `decrease_lenght()` مسئول کوچک کردن کرم بازیکن در صورت برخورد کردن سر کرم بازیکن به دیوارهای حاشیه‌ایی یا درون خود زمین و یا بدن کرم بازیکنان دیگر یا بدن کرم خود بازیکن است. این تابع به این صورت عمل می‌کند: اگر طول کرم بازیکن از حد مشخصی کمتر باشد، طول کرم بازیکن را برابر صفر قرار می‌دهد تا نشان دهد بازیکن بازی را باخته است. ولی اگر طول کرم بازی بیش از حد آستانه باشد دو حالت پیش می‌آید؛ یا کرم نقطه‌ی شکستگی دارد یا ندارد. اگر کرم نقطه‌ی شکستگی نداشته باشد تنها کافی است بسته به این که راستای سر کرم افقی یا عمودی است، یا مختص طولی یا مختص عرضی دم کرم دستکاری شوند. شکل ۳-۹ حالتی را نشان می‌دهد که طول کرم بیش از حد آستانه و کرم فاقد نقطه شکستگی است.

```

while (not_to_exit)
{
    if(breaking_points.size()==0)
    {
        if(rasta_head == 'h')
        {
            if(head.Xposition > tail.Xposition)
            {
                head.Xposition-= decrease ;
                not_to_exit = false ;
            }
            else
            {
                head.Xposition+= decrease ;
                not_to_exit = false ;
            }
        }
        else
        {
            if(head.Yposition > tail.Yposition)
            {
                head.Yposition-= decrease ;
                not_to_exit = false ;
            }
            else
            {
                head.Yposition+= decrease ;
                not_to_exit = false ;
            }
        }
    }
}

```

شکل ۳-۹- حالتی که طول کرم بیش از حد آستانه و کرم فاقد نقطه‌ی شکستگی است.

ولی اگر کرم حداقل یک نقطه‌ی شکستگی داشته باشد چهار حالت مختلف پیش می‌آید. این چهار حالت عبارتند از: جهت سر کرم به سمت راست، چپ، پایین یا بالای زمین باشد. یکی از حالت‌ها را توضیح می‌دهیم. اگر جهت سر کرم به سمت راست زمین باشد، دوباره با سه حالت مواجه می‌شویم؛ یا فاصله بین سر و اولین نقطه‌ی شکستگی از مقدار decrease بیشتر یا مساوی یا کمتر است. چنانچه فاصله بیشتر باشد تنها کافی است که مختص طولی سر کرم به اندازه decrease کاهش پیدا کند. اگر فاصله مساوی باشد در فیلد مختصات سر کرم، مختصات اولین نقطه‌ی شکستگی قرار داده می‌شود و اولین نقطه‌ی شکستگی از لیست نقاط شکستگی حذف و راستای سر کرم عمودی می‌شود. اگر فاصله کمتر باشد همان عملیات حالت قبلی (فاصله مساوی) انجام می‌شود با این تفاوت که الگوریتم دوباره اجرا می‌شود و مقدار کاهش طول کرم را برابر تفاوت بین decrease و فاصله بین سر و نقطه‌ی شکستگی اول در اجرای قبلی در نظر می‌گیرد. شکل‌های ۳-۱۰، ۳-۱۱، ۳-۱۲ و ۳-۱۳ به ترتیب حالت‌هایی را نشان می‌دهند که طول کرم بیش از حد آستانه و کرم دارای حداقل یک نقطه‌ی شکستگی است و جهت سر کرم به سمت راست، چپ، پایین یا بالا است.

```

first_breaking_point = breaking_points[0];
if(rasta_head == 'h')
{
    if(head.Xposition>first_breaking_point.Xposition)
    {
        int distance=head.Xposition - first_breaking_point.Xposition;
        if(distance > decrease_per_time)
        {
            head.Xposition -= decrease_per_time ;
            not_to_exit = false ;
        }
        else
        {
            if(distance == decrease_per_time)
            {
                head.Xposition -= decrease_per_time ;
                for(int i=0;i<breaking_points.size()-1;i++)
                    breaking_points[i]=breaking_points[i+1];
                breaking_points.pop_back();
                rasta_head='v';
                not_to_exit=false;

            }
            else
            {
                head.Xposition -= distance;
                for(int i=0;i<breaking_points.size()-1;i++)
                    breaking_points[i]=breaking_points[i+1];
                breaking_points.pop_back();
                rasta_head='v';
                decrease_per_time -= distance;
            }
        }
    }
}

```

۳-۱۰- حالتی که طول کرم بیش از حد آستانه و کرم حداقل دارای یک نقطه شکستگی است و جهت سر کرم به سمت راست است.


```

else
{
    int distance= first_breaking_point.Xposition-head.Xposition;
    if(distance > decrease_per_time)
    {
        head.Xposition += decrease_per_time ;
        not_to_exit = false ;
    }
    else
    {
        if(distance == decrease_per_time)
        {
            head.Xposition += decrease_per_time ;
            for(int i=0;i<breaking_points.size()-1;i++)
                breaking_points[i]=breaking_points[i+1];
            breaking_points.pop_back();
            rasta_head='v';
            not_to_exit=false;
        }
        else
        {
            head.Xposition += distance;
            for(int i=0;i<breaking_points.size()-1;i++)
                breaking_points[i]=breaking_points[i+1];
            breaking_points.pop_back();
            rasta_head='v';
            decrease_per_time -= distance;
        }
    }
}

```

شکل ۳-۱۱- حالتی که طول کرم بیش از حد آستانه و کرم حداقل دارای یک نقطه شکستگی است و جهت سر کرم به سمت چپ است.

```

if(head.Yposition>first_breaking_point.Yposition)
{
    int distance=head.Yposition - first_breaking_point.Yposition;
    if(distance > decrease_per_time)
    {
        head.Yposition -= decrease_per_time ;
        not_to_exit = false ;
    }
    else
    {
        if(distance == decrease_per_time)
        {
            head.Yposition -= decrease_per_time ;
            for(int i=0;i<breaking_points.size()-1;i++)
                breaking_points[i]=breaking_points[i+1];
            breaking_points.pop_back();
            rasta_head='h';
            not_to_exit=false;
        }
        else
        {
            head.Yposition -= distance;
            for(int i=0;i<breaking_points.size()-1;i++)
                breaking_points[i]=breaking_points[i+1];
            breaking_points.pop_back();
            rasta_head='h';
            decrease_per_time -= distance;
        }
    }
}

```

شکل ۳-۱۲- حالتی که طول کرم بیش از حد آستانه و کرم حداقل دارای یک نقطه شکستگی است و جهت سر کرم به سمت پایین است.

```

else
{
    int distance= first_breaking_point.Yposition-head.Yposition;
    if(distance > decrease_per_time)
    {
        head.Yposition += decrease_per_time ;
        not_to_exit = false ;
    }
    else
    {
        if(distance == decrease_per_time)
        {
            head.Yposition += decrease_per_time ;
            for(int i=0;i<breaking_points.size()-1;i++)
                breaking_points[i]=breaking_points[i+1];
            breaking_points.pop_back();
            rasta_head='h';
            not_to_exit=false;

        }
        else
        {
            head.Yposition += distance;
            for(int i=0;i<breaking_points.size()-1;i++)
                breaking_points[i]=breaking_points[i+1];
            breaking_points.pop_back();
            rasta_head='h';
            decrease_per_time -= distance;
        }
    }
}

```

شکل ۳-۱۳- حالتی که طول کرم بیش از حد آستانه و کرم حداقل دارای یک نقطه شکستگی است و جهت سر کرم به سمت بالا است.

در صورتی که کرم یک نقطه‌ی افزایشی را بخورد، تابع `increase_lenght()` صدا زده می‌شود تا طول کرم را افزایش دهد. نحوه‌ی عملکرد تابع به این صورت است که اگر راستای سر کرم افقی باشد مختص طولی سر کرم را بسته به این که سر کرم در جهت راست یا چپ زمین است، به اندازه‌ی `increase` افزایش یا کاهش می‌دهد ولی اگر راستای سر کرم عمودی باشد بسته به این که جهت سر کرم به سمت پایین یا بالا زمین بازی است، مختص عرضی سر کرم را افزایش یا کاهش می‌دهد. شکل ۳-۱۴ حالتی را نشان می‌دهد که راستای سر کرم افقی است و شکل ۳-۱۵ حالتی را نشان می‌دهد که راستای سر کرم عمودی است.

```

lenght += increase ;
if(rasta_head == 'h')
{
    if(breaking_points.size()==0)
    {
        if(head.Xposition> tail.Xposition)
        {
            head.Xposition += increase;
        }
        else
        {
            head.Xposition -= increase;
        }
    }
    else
    {
        coordinates first_breaking_point = breaking_points[0];
        if(head.Xposition > first_breaking_point.Xposition)
        {
            head.Xposition+=increase;
        }
        else
        {
            head.Xposition-=increase;
        }
    }
}

```

۳-۱۴- حالتی از تابع `increase_lenght()` که راستای سر کرم افقی است.

```

else
{
    if(breaking_points.size()==0)
    {
        if(head.Yposition> tail.Yposition)
        {
            head.Yposition += increase;
        }
        else
        {
            head.Yposition -= increase;
        }
    }
    else
    {
        coordinates first_breaking_point = breaking_points[0];
        if(head.Yposition > first_breaking_point.Yposition)
        {
            head.Yposition+=increase;
        }
        else
        {
            head.Yposition-=increase;
        }
    }
}

```

۳-۱۵- حالتی از تابع `increase_lenght()` که راستای سر کرم عمودی است.

تابع `move_tail()` دم کرم بازیکن حرکت می‌دهد و تنها زمانی صدا زده می‌شود که کرم بازیکن حداقل یک نقطه‌ی شکستگی داشته باشد. این تابع به این صورت عمل می‌کند که بسته به این که جهت دم کرم به سمت راست، چپ، پایین یا بالا باشد، مختص طولی یا مختص عرضی دم را به اندازه‌ی `speed` افزایش یا کاهش می‌دهد و اگر در اثر این تغییر مختصات دم با مختصات آخرین نقطه‌ی شکستگی برابر شود، آخرین نقطه‌ی شکستگی را از لیست نقاط شکستگی حذف می‌کند و مقدار متغیر `rasta_tail` را به صورت مناسبی تغییر می‌دهد. شکل ۳-۱۶ کد این تابع را نشان می‌دهد.

```
void worm::move_tail()
{
    int breaking_points_number =breaking_points.size();
    if(rasta_tail=='h')
    {
        coordinates the_last_breaking_point =breaking_points[breaking_points_number-1];
        if(the_last_breaking_point.Xposition>tail.Xposition)
            tail.Xposition+=speed ;
        else
            tail.Xposition-=speed ;
        if(tail.Xposition == the_last_breaking_point.Xposition)
        {
            breaking_points.pop_back();
            rasta_tail='v';
        }
    }
    else
    {
        coordinates the_last_breaking_point =breaking_points[breaking_points_number-1];
        if(the_last_breaking_point.Yposition>tail.Yposition)
            tail.Yposition+=speed ;
        else
            tail.Yposition-=speed ;
        if(tail.Yposition == the_last_breaking_point.Yposition)
        {
            breaking_points.pop_back();
            rasta_tail ='h';
        }
    }
}
```

شکل ۳-۱۶- کد تابع `move_tail()`

تابع `self_smash()` چک می‌کند که آیا سر کرم با بدن خودش برخورد دارد یا نه و در صورت تشخیص برخورد تابع `decrease_lenght()` را برای کوچک کردن کرم فراخوانی می‌کند. تابع `self_samsh()` برای تشخیص برخورد، نقاط شکستگی و دم را با شروع از نقطه‌ی شکستگی اول دوبه‌دو با هم در نظر می‌گیرد و بررسی می‌کند که آیا خطی که سر و اولین نقطه‌ی شکستگی کرم را به هم وصل می‌کند، خط وصل کننده این

دو نقطه (دو نقطه در نظر گرفته شده) به یکدیگر را قطع می‌کند یا نه. اگر خطوط یاد شده یکدیگر را قطع کنند، یعنی سر کرم به بدن خودش برخورد کرده است و باید کرم کوچک شود. شکل ۳-۱۷ کد این تابع را نشان می‌دهد.

```

465
466 void worm::self_smash()
467 {
468     char X=head.Xposition;
469     char Y=head.Yposition;
470     if(breaking_points.size()>=3)
471     {
472         if(rasta_head == 'h')
473         {
474             if(head.Xposition>breaking_points[0].Xposition)
475             {
476                 for(int i=0 ;i<breaking_points.size();i=i+2) {...}
519             }
520             else
521             {
522                 for(int i=0 ;i<breaking_points.size();i=i+2) {...}
565             }
566         }
567     }
568     else
569     {
570         if(head.Yposition > breaking_points[0].Yposition)
571         {
572             for(int i=0 ;i<breaking_points.size();i=i+2) {...}
614         }
615         else
616         {
617             for(int i=0 ;i<breaking_points.size();i=i+2) {...}
660         }
661     }
662 }
663 }
```

شکل ۳-۱۷- کد تابع self_smash()

۳-۳- پیاده‌سازی کلاس کاربر

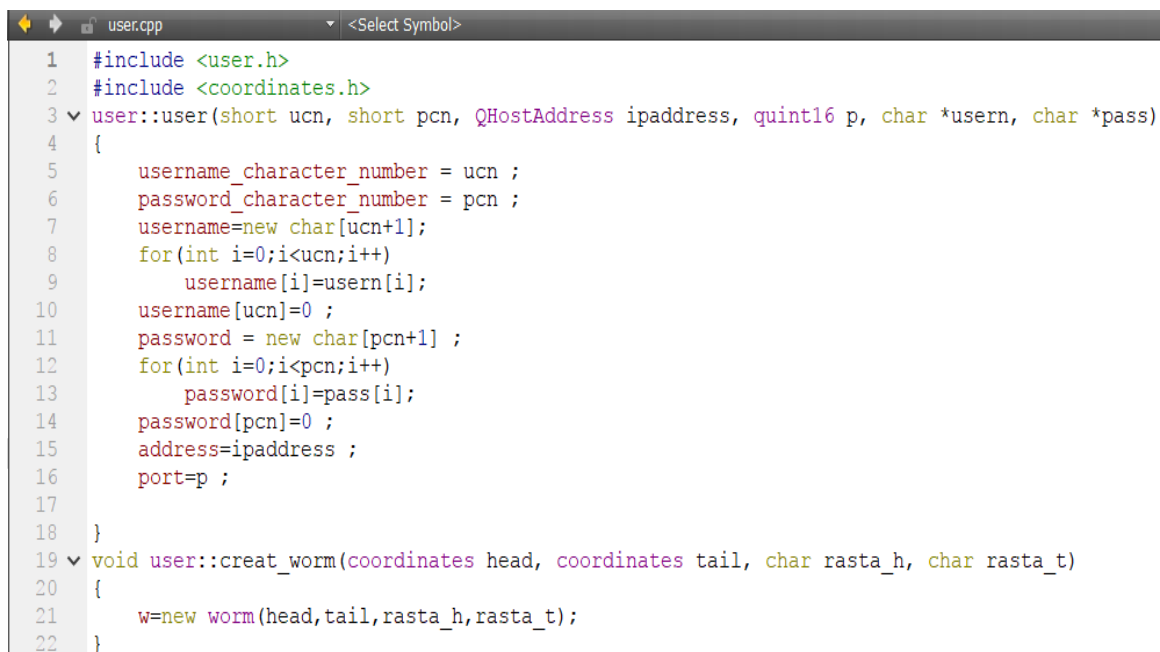
کلاس کاربر برای مدل کردن کاربر استفاده می‌شود. آدرس IP و شماره پورتی که کاربر با آن‌ها بسته‌های خود را برای سرور ارسال کرده است، در دو متغیر address و port نگهداری می‌شوند تا سرور بتواند با استفاده از

آنها بسته‌های داده مورد نیاز را برای کاربر ارسال کند. این کلاس تنها یک تابع به نام `create_worm()` دارد که مسئول ایجاد یک کرم برای کاربر است. شکل ۳-۱۸ نحوه تعریف این کلاس و شکل ۳-۱۹ کد تابع `creat_worm()` را نشان می‌دهد.

```
#include <worm.h>
#include <QHostAddress>
#include <string>
#ifndef USER_H
#define USER_H
class user{
public:
    short username_character_number;
    short password_character_number ;
    char * username ;
    char * password ;
    worm * w;
    QHostAddress address ;
    quint16 port ;
public:
    user(short ucn,short pcn,QHostAddress ipaddress,quint16 p,char * usern,char * pass);
    void creat_worm(coordinates head,coordinates tail,char rasta_h,char rasta_t);
};

#endif // USER_H
```

شکل ۳-۱۸- نحوه تعریف کلاس کاربر در فایل سرآیند `user.h`



```
1 #include <user.h>
2 #include <coordinates.h>
3 user::user(short ucn, short pcn, QHostAddress ipaddress, quint16 p, char *usern, char *pass)
4 {
5     username_character_number = ucn ;
6     password_character_number = pcn ;
7     username=new char[ucn+1];
8     for(int i=0;i<ucn;i++)
9         username[i]=usern[i];
10    username[ucn]=0 ;
11    password = new char[pcn+1] ;
12    for(int i=0;i<pcn;i++)
13        password[i]=pass[i];
14    password[pcn]=0 ;
15    address=ipaddress ;
16    port=p ;
17 }
18
19 void user::creat_worm(coordinates head, coordinates tail, char rasta_h, char rasta_t)
20 {
21     w=new worm(head,tail,rasta_h,rasta_t);
22 }
```

شکل ۳-۱۹- کد تابع `creat_worm()`

۳-۴- پیاده‌سازی کلاس‌های دیوار و زمین بازی

کلاس دیوار انواع مختلف دیوار برای زمین‌های بازی غیر ساده را مدل سازی می‌کند و شامل هیچ‌گونه عملیاتی نمی‌شود. کلاس زمین بازی نیز زمین بازی را مدل می‌کند و لیستی از دیوارهای زمین را در خود نگهداری می‌کند. شکل ۳-۲۰ نحوه‌ی تعریف کلاس دیوار و کلاس زمین بازی را نشان می‌دهد.

```

1  #include <coordinates.h>
2  #include <vector>
3  using namespace std;
4  #ifndef PLAYGROUND_H
5  #define PLAYGROUND_H
6
7  class wall{
8  public:
9      short type;
10     coordinates top_left ;
11 };
12
13 class playground{
14 public:
15     vector<wall> walls ;
16 };
17
18 #endif // PLAYGROUND_H
19

```

شکل ۳-۲۰- نحوه‌ی تعریف کلاس‌های دیوار و زمین بازی در فایل سرآیند playground.h

۳-۵- پیاده‌سازی کلاس بازی

کلاس بازی، یک بازی در حال انجام را مدل می‌کند. شکل ۳-۲۲ نحوه‌ی تعریف این کلاس را نشان می‌دهد. تابع add_user() یک کاربر را دریافت و به انتهای لیست کاربران حاضر در بازی اضافه می‌کند. شکل ۳-۲۱ کد این تابع را نشان می‌دهد.

```

53 void game::add_user(user u)
54 {
55     users.push_back(u) ;
56 }

```

شکل ۳-۲۱- کد تابع add_user()

```

1  #ifndef GAME_H
2  #define GAME_H
3
4  #include <QObject>
5  #include <user.h>
6  #include <vector>
7  #include <coordinates.h>
8  #include <QTimer>
9  #include <playground.h>
10 #include <string>
11 class game : public QObject
12 {
13     Q_OBJECT
14 public:
15     int id ;
16     vector<user> users ;
17     vector<coordinates> increasing_points ;
18     QTimer * timer ;
19     playground * ground ;
20     explicit game(QObject *parent = 0,int id_game=0,playground * gr=NULL);
21
22     void move_worm(string username,char jahat);
23     void add_user(user u);
24     ~game();
25     void delete_user(int index);
26
27     signals:
28         void state_calculated(int game_id);
29 private slots:
30     void calculate_state();
31
32 };
33
34 #endif // GAME_H

```

شکل ۳-۲۲- نحوه‌ی تعریف کلاس بازی در فایل سرآیند game.h

تابع `move()` کرم کاربر حاضر در بازی با نام کاربری `username` را در جهت `jahat` حرکت می‌دهد. بعد از حرکت دادن کرم چک می‌کند که کرم با انجام این حرکت نقطه‌ی افزایشی‌ایی خورده است یا نه و اگر کرم یک نقطه‌ی افزایشی خورده باشد، به شرط این‌که نقطه‌ی افزایشی روی بدن بازیکن دیگری نباشد طول کرم بازیکن را با فراخوانی تابع `increase_lenght()` خود کرم افزایش می‌دهد و نقطه‌ی افزایشی جدیدی به جای نقطه‌ی مصرف شده تولید می‌کند. دلیل بررسی این شرط این است که اگر نقطه‌ی افزایشی روی بدن کرم بازیکن دیگر باشد و کرم حرکت داده شده نقطه‌ی افزایشی را خورده باشد، درواقع کرم حرکت داده شده به بدن کرم دیگر برخورد کرده است و باید طولش کاهش یابد نه افزایش. بعد از بررسی نقاط افزایشی، تابع چک می‌کند که کرم مورد نظر با انجام حرکت جدید به دیوارهای حاشیه‌ای زمین برخورد کرده است یا نه و در صورت برخورد طول کرم را کاهش می‌دهد. شکل ۳-۲۳ در صفحه بعد کد تابع `move()` را نشان می‌دهد.


```

57 void game::move_worm(string username, char jahat)
58 {
59     //this function moves the user worm according received jahat
60     cout <<"moving worm"<<endl;
61     int index ;
62     for(int i=0;i<users.size();i++) { ... }
63     users[index].w->move(jahat);
64     char X=users[index].w->head.Xposition;
65     char Y=users[index].w->head.Yposition;
66     if(ground->walls.size()==0)
67     {
68         for(int i=0;i<4;i++)
69         {
70             if(increasing_points[i].Xposition == X && increasing_points[i].Yposition == Y)
71             {
72                 if((increasing_points[i].Xposition>0 && increasing_points[i].Xposition<5)
73                 if((increasing_points[i].Yposition>0 && increasing_points[i].Yposition<5)
74                 for(int j=0;j<users.size();j++) { ... }
75                 coordinates inc_point ;
76                 int Xvalue;
77                 int Yvalue;
78                 users[index].w->increase_lenght();
79                 if(i==0) { ... }
80                 if(i==1) { ... }
81                 if(i== 2) { ... }
82                 if(i==3) { ... }
83             }
84         }
85         X=users[index].w->head.Xposition;
86         Y=users[index].w->head.Yposition;
87         cout <<users[index].username <<" " <<(int) X << " " <<(int)Y<<endl ;
88         if((int)X==0 || (int)X== 27)
89         {
90             users[index].w->decrease_lenght();
91         }
92     }
93     else
94     {
95         if((int)Y== 0 || (int)Y== 41)
96         {
97             users[index].w->decrease_lenght();
98         }
99     }
100 }
101 }
102 }

```

شکل ۳-۲۳- کد تابع move()

در پایان اسلات‌های زمانی که توسط تایمر زمان‌بندی می‌شوند تابع calculate_state() فراخوانی می‌شود. این تابع وضعیت کرم‌ها را نسبت به یکدیگر در نظر می‌گیرد و در صورت برخورد سر یک کرم به بدن کرم دیگر، طول کرم برخورد کننده را کاهش می‌دهد. به دلیل حالت‌های محتمل زیاد، این تابع پیچیده‌ترین و طولانی‌ترین تابع در میان تابع‌های برنامه سرور است. این تابع بوسیله دو حلقه for تودرتو کار خود را انجام می‌دهد. حلقه اول بازیکن اول را از لیست بازیکنان و حلقه دوم بازیکن دوم را از لیست بازیکنان انتخاب می‌کند و کدهای درون حلقه دوم بررسی می‌کنند که آیا سر کرم بازیکن اول (بازیکن با اندیس i) به بدن کرم بازیکن دوم (بازیکن با

اندیس j) برخورد داشته است یا نه. برای تشخیص برخورد این تابع بررسی می‌کند که آیا خطی که سر کرم بازیکن اول را به اولین نقطه‌ی شکستگی یا دم کرم بازیکن اول (بسته به این که کرم نقطه‌ی شکستگی دارد یا نه) متصل می‌کند با یکی از خطوط متصل کننده‌ی دو نقطه‌ی اصلی متوالی کرم بازیکن دوم تقاطع دارد یا خیر و در صورت تقاطع طول کرم بازیکن اول را کاهش می‌دهد. شکل ۳-۲۴ قسمتی از کد این تابع را نمایش می‌دهد.

```

362 char fby ;
363 for(int i=0 ;i<users_number;i++)
364 {
365     if(users[i].w->lenght != 0)
366     {
367         X=users[i].w->head.Xposition ;
368         Y=users[i].w->head.Yposition ;
369         if(users[i].w->breaking_points.size()==0) {...}
374         else {...}
379         for(int j=0;j<users_number;j++)
380         {
381             if(j!=i && users[j].w->lenght!=0)
382             {
383                 if(users[i].w->rasta_head == 'h')
384                 {
385
386                     if(X > fbx)
387                     {
388                         if(users[j].w->breaking_points.size()==0) {...}
480                         else
481                         {
482                             if(users[j].w->rasta_head == 'h')
483                             {
484                                 if(users[j].w->breaking_points.size()==1) {...}
610                                 else {...}
751                             }
752                             else {...}
889                         }
890                     }
891                     else
892                     {
893                         if(users[j].w->breaking_points.size()==0) {...}
985                         else
986                         {
987                             if(users[j].w->rasta_head == 'h')
988                             {
989                                 if(users[j].w->breaking_points.size()==1) {...}
1117                                else {...}
1257                            }
1258                            else {...}
1395                        }
1396                    }
1397                }

```

شکل ۳-۲۴- قسمتی از کد تابع calculate_state()

۳-۶- پیاده‌سازی کلاس سرور

کلاس سرور بزرگترین کلاس برنامه و دربردارنده‌ی همه کلاس‌های قبلی است. شکل ۳-۲۵ نحوه‌ی تعریف این کلاس را نشان می‌دهد.

```

1  #ifndef SERVER_H
2  #define SERVER_H
3
4  #include <QObject>
5  #include <QUdpSocket>
6  #include <QHostAddress>
7  #include <vector>
8  #include <user.h>
9  #include <game.h>
10 #include <playground.h>
11 #include <QSqlDatabase>
12 #include <QTimer>
13 using namespace std;
14 class server : public QObject
15 {
16     Q_OBJECT
17 public:
18     vector<game *> games ;
19     vector<user> authenticated_users ;
20     QUdpSocket * _socket ;
21     playground * playgrounds ;
22     vector<user> waiting_for_playing;
23     QSqlDatabase db ;
24     QTimer start_new_game ;
25     int the_highest_availble_id ;
26     vector<int> removed_game_ids;
27 public:
28     explicit server(QObject *parent = 0);
29     ~server();
30
31
32 public slots:
33     void ready_read();
34     void send_state(int id_game);
35     void add_new_game();
36
37 };
38
39 #endif // SERVER_H
40
```

شکل ۳-۲۵- نحوه‌ی تعریف کلاس سرور در فایل سرآیند server.h

در اثر دریافت داده از شبکه، برای تحلیل داده‌ی رسیده و انجام عملیات مناسب تابع ready_read() فراخوانی می‌شود. این تابع داده دریافتی را از روی سوکت می‌خواند و در یک آرایه کپی می‌کند. سپس بایت اول آرایه

برای پی بردن به نوع پیام دریافتی می‌خواند. از بایت اول کاراکتر C باشد، یعنی که پیام نوع C دریافت کرده‌ایم. پیام نوع C برای احراز اصالت کاربران با نام کاربری و کلمه عبور جدید به کار می‌رود. در صورت دریافت پیام نوع C، سرور نام کاربری و کلمه عبور را از پیام استخراج می‌کند و سپس اگر کاربر دیگری با نام کاربری مساوی با نام کاربری استخراج شده در پایگاه داده خود نداشته باشد، یک پیام نوع C مبنی بر موفقیت آمیز بودن احراز اصالت را برای کاربر مورد نظر ارسال می‌کند و در غیر اینصورت در قالب یک پیام نوع C دیگر به کاربر اعلام می‌دارد نمی‌تواند از این نام کاربری جدید استفاده کند. شکل ۳-۲۶ قسمتی از تابع `ready_read()` را نشان می‌دهد که در اثر دریافت پیام نوع C اجرا می‌شود.

```
if(received_data[0]=='c')
{
    int username_chactrer_number=received_data[1];
    char incoming_username[username_chactrer_number+1];
    int index=0;
    for(int i=2;i<username_chactrer_number+2;i++)
    {
        incoming_username[index]=received_data[i];
        index++;
    }
    incoming_username[username_chactrer_number]=0;
    int password_character_number=received_data[username_chactrer_number+2];
    char incoming_password[password_character_number+1] ;
    index= 0;
    for(int i=username_chactrer_number+3;i<username_chactrer_number+3+password_character_number;i++)
    {
        incoming_password[index]=received_data[i];
        index++;
    }
    incoming_password[password_character_number]=0;
    cout <<"new user with username:"<<" "<<incoming_username<<" "<<"and password:"<<" "<<incoming_password<<endl ;
    if(db.open())
    {
        QSqlQuery q;
        q.prepare("select * from user_table where username=:u ");
        QString u(incoming_username);
        q.bindValue(0,u);
        q.exec();
        q.next();
        if(q.isValid()) { ... }
        else { ... }
    }
}
```

شکل ۳-۲۶-قسمتی از `ready_read()` که در اثر دریافت پیام نوع C اجرا می‌شود.

اگر پیام دریافتی از نوع a باشد، عملیات مورد نیاز مانند حالت قبل (نوع C) است با این تفاوت که وجود کاربری با نام کاربری و کلمه عبور مساوی با نام کاربری و کلمه عبور استخراج شده از داده یعنی عملیات احراز اصالت

موفقیت‌آمیز است و در غیر این صورت یعنی عملیات احراز اصالت با شکست مواجه شده است. شکل ۳-۲۷ قسمتی از کد تابع `ready_read()` را نشان می‌دهد که در اثر دریافت پیام نوع C اجرا می‌شود.

```
if(received_data[0]=='a')
{
    int username_character_number=received_data[1];
    char incoming_username[username_character_number+1];
    int index=0;
    for(int i=2;i<username_character_number+2;i++)
    {
        incoming_username[index]=received_data[i];
        index++;
    }
    incoming_username[username_character_number]=0;
    int password_character_number=received_data[username_character_number+2];
    char incoming_password[password_character_number+1];
    index=0;
    for(int i=username_character_number+3;i<username_character_number+3+password_character_number;i++)
    {
        incoming_password[index]=received_data[i];
        cout <<(int)incoming_password[index]<<endl;
        index++;
    }
    incoming_password[password_character_number]=0;
    cout <<"user with username:"<<" "<<incoming_username<<" "<<"and password:"<<" "<<QString(incoming_password)
    if(db.open())
    {
        QSqlQuery q;
        q.prepare("select * from user_table where username=:u and password=:p ");
        QString u(incoming_username);
        QString p(incoming_password);
        q.bindValue(0,u);
        q.bindValue(1,p);
        q.exec();
        q.next();
        if(q.isValid()) { ... }
        else { ... }
    }
}
```

شکل ۳-۲۷-قسمتی از تابع `ready_read()` که در اثر دریافت پیام نوع a اجرا می‌شود.

اگر پیام دریافتی از نوع p باشد، سرور پس از استخراج نام کاربری از پیام دریافتی شروع به جست‌وجوی کاربر در لیست کاربران احراز اصالت شده می‌کند. پس از پیدا کردن کاربر مورد نظر، کاربر را از لیست کاربران احراز اصالت شده خارج و به انتهای لیست کاربران منتظر شروع بازی جدید اضافه می‌کند. شکل ۳-۲۸ قطعه کد اجرایی از تابع `ready_read()` را نشان می‌دهد که در اثر دریافت پیام نوع p اجرا می‌شود.

اگر پیام دریافتی از نوع n باشد، سرور نام کاربری، شناسه زمین و جهت حرکت را از پیام استخراج می‌کند و در لیست بازی‌های خود دنبال بازی با شناسه استخراج شده می‌گردد و پس از پیدا کردن بازی مورد نظر، تابع `move()` از شی بازی را با دادن پارامترهای نام کاربری و جهت حرکت فراخوانی می‌کند. شکل ۳-۲۹ قسمتی از کد تابع `ready_read()` را نشان می‌دهد که در اثر دریافت پیام نوع n اجرا می‌شود.

```

if(received_data[0]=='p')
{
    int username_chactrer_number=received_data[1];
    char incoming_username[username_chactrer_number+1];
    int index=0;
    for(int i=2;i<username_chactrer_number+2;i++)
    {
        incoming_username[index]=received_data[i];
        index++;
    }
    incoming_username[username_chactrer_number]=0;
    int ground_id =received_data[username_chactrer_number+2];
    cout <<"player"<<" " <<incoming_username<<" " <<"choosed playground 1"<<endl ;
    string incoming_user(incoming_username);
    for(int i=0;i<authenticated_users.size();i++)
    {
        string a_u(authenticated_users[i].username) ;
        if(incoming_user == a_u)
        {
            if(ground_id == 1)
            {
                waiting_for_playing.push_back(authenticated_users[i]);
                for(int j=i;j<authenticated_users.size()-1;j++)
                    authenticated_users[j]=authenticated_users[j+1];
                authenticated_users.pop_back();
                break ;
            }
        }
    }
}
}

```

شکل ۳-۲۸- قسمتی از کد ready_read() که در اثر دریافت پیام نوع p اجرا می‌شود.

```

if(received_data[0]=='n')
{
    int username_chactrer_number=received_data[1];
    char incoming_username[username_chactrer_number+1];
    int index=0;
    for(int i=2;i<username_chactrer_number+2;i++)
    {
        incoming_username[index]=received_data[i];
        index++;
    }
    incoming_username[username_chactrer_number]=0;
    int game_id =received_data[username_chactrer_number+2];
    char jahat= received_data[username_chactrer_number+3];
    cout <<"user"<<" " <<incoming_username<<" " <<"moved to " <<" " <<jahat<<endl ;
    cout <<received_data<<endl ;
    for(int i=0;i<games.size();i++)
    {
        if(games[i]->id == game_id)
        {
            string incoming_user(incoming_username);
            for(int j=0;j<games[i]->users.size();j++)
            {
                string p_u(games[i]->users[j].username);
                if(incoming_user == p_u)
                {
                    if(games[i]->users[j].w->lenght != 0)
                    {
                        games[i]->move_worm(games[i]->users[j].username,jahat);
                    }
                    break ;
                }
            }
            break ;
        }
    }
}
}

```

شکل ۳-۲۹- قطعه‌ای از کد ready_read() که در اثر دریافت پیام نوع n اجرا می‌شود.

در پایان اسلات‌های زمانی مشخصی که توسط تایمرهای بازی‌های موجود در لیست بازی‌ها زمان‌بندی می‌شوند، تابع `send_state()` فراخوانی می‌شود. این تابع مسئول فرستادن وضعیت بروزشده‌ی کرم‌های بازی برای تک‌تک کاربران حاضر در زمین بازی است. این تابع ابتدا یک بار لیست کاربران حاضر در بازی را برای پیدا کردن کاربران بازنده پویش می‌کند و هر بار که یک کاربر بازنده پیدا می‌کند، یک پیام S نوع دوم برای کاربر ارسال و کاربر را از لیست کاربران حاضر در بازی حذف می‌کند. پس از اتمام پویش اول تنها کاربرانی در لیست کاربران بازی باقی می‌مانند که یا برنده‌اند و یا هنوز بازی آن‌ها تمام نشده است. حال دوباره لیست کاربران بازی پویش می‌شود. اگر فقط یک کاربر در لیست باقی‌مانده باشد، آن کاربر برنده است و برایش یک پیام S نوع اول ارسال و بازی از لیست بازی‌های سرور حذف می‌شود. ولی اگر تعداد کاربران باقی‌مانده بیش از یک باشد برای تک‌تک کاربران یک پیام S نوع سه حاوی اطلاعات بروزشده‌ی کرم خود بازیکن و بازیکنان حاضر دیگر ارسال می‌شود. شکل ۳-۳۰ کد این تابع را نمایش می‌دهد.

```
void server::send_state(int id_game)
{
    for(int i=0;i<games.size();i++)
    {
        if(games[i]->id == id_game)
        {
            int sending_message_size = 2 ;
            int iterator=games[i]->users.size();
            int index =0 ;
            for(int j=0;j<iterator;j++)
            {
                if(games[i]->users[index].w->lenght == 0)
                {
                    char sending_data[2];
                    sending_data[0]='s';
                    sending_data[1]='o';
                    _socket->writeDatagram(sending_data,2,games[i]->users[index].address,games[i]->users[index].port);
                    delete games[i]->users[index].username ;
                    delete games[i]->users[index].password ;
                    delete games[i]->users[index].w ;
                    games[i]->users[index] = games[i]->users.back();
                    games[i]->users.pop_back();

                }
                else
                {
                    sending_message_size +=(games[i]->users[index].w->breaking_points.size()*2)+5;
                    index ++ ;
                }
            }
            if(games[i]->users.size() <=1) {...}
            else {...}
        }
    }
}
```

شکل ۳-۳۰- کد تابع `send_state()`

در پایان اسلات‌های زمانی مشخصی که توسط خود تایمر کلاس سرور زمان‌بندی می‌شوند، تابع `add_new_game()` فراخوانی می‌شود. این تابع لیست کاربران منتظر برای شروع بازی جدید را بررسی می‌کند و چنانچه حداقل ۲ کاربر منتظر در لیست وجود داشته باشند، این تابع یک بازی جدید با یک شناسه جدید ایجاد می‌کند، کرم‌های بازیکنان را مقداردهی اولیه می‌کند، بازیکنان منتظر را از لیست بازیکنان در حال انتظار خارج و به لیست بازیکنان حاضر در بازی جدید اضافه می‌کند، بازی جدید را به لیست بازی‌های خود اضافه می‌کند و شناسه بازی و وضعیت اولیه کرم‌ها را در قالب پیام نوع `g` برای تک‌تک بازیکنان ارسال می‌کند. لازم به ذکر است حداکثر ۴ بازیکن می‌توانند در یک بازی با هم بازی کنند. شکل ۳-۳۱ قسمتی از کد این تابع را نشان می‌دهد که پیام نوع `g` را برای تک‌تک بازیکنان ارسال می‌کند.

```
games.push_back(new_game);
char sending_data[(new_game->users.size()*4)+3];
sending_data[0]='g' ;
sending_data[1]=new_game_id;
sending_data[2]=new_game->users.size() ;
for(int i=0;i<new_game->users.size();i++)
{
    sending_data[3]=new_game->users[i].w->tail.Xposition;
    sending_data[4]=new_game->users[i].w->tail.Yposition;
    sending_data[5]=new_game->users[i].w->head.Xposition;
    sending_data[6]=new_game->users[i].w->head.Yposition;
    int sending_data_index =7 ;
    for(int j=0;j<new_game->users.size();j++)
    {
        if(j!= i)
        {
            sending_data[sending_data_index]=new_game->users[j].w->tail.Xposition;
            sending_data_index++;
            sending_data[sending_data_index]=new_game->users[j].w->tail.Yposition;
            sending_data_index ++ ;
            sending_data[sending_data_index]=new_game->users[j].w->head.Xposition;
            sending_data_index ++ ;
            sending_data[sending_data_index]=new_game->users[j].w->head.Yposition;
            sending_data_index ++ ;
        }
    }
    _socket->writeDatagram(sending_data, (new_game->users.size()*4)+3,new_game->users[i].address,new_game->users[i].port);
    cout <<"new game added successfully"<<endl ;
}
connect(games[games.size()-1],SIGNAL(state_calculated(int )),this,SLOT(send_state(int )));
games[games.size()-1]->timer->start(2000);
```

شکل ۳-۳۱-قسمتی از کد تابع `add_new_game` که برای تک‌تک بازیکنان پیام نوع `g` ارسال می‌کند.

فصل چهارم: نتیجه‌گیری و پیشنهادها

۴-۱- نتیجه‌گیری

در مقدمه در مورد بازی کرم‌ها و قواعد آن مطالبی را بیان نمودیم و گفتیم هدف از انجام این پروژه چیست و با پروژه‌های مشابه انجام شده در این زمینه چه تفاوتی دارد. همچنین توضیحاتی در مورد روند انجام کار ارائه دادیم. در فصل مقدمات در مورد تکنولوژی‌های به کار رفته در توسعه پروژه توضیحات مختصری ارائه نمودیم. در فصل طراحی، پرتکل ارتباطی بین بازیکنان و سرور را طراحی کردیم. این پرتکل توضیح می‌داد که در هر مرحله باید چه پیامی با چه ساختاری بین بازیکن و سرور منتقل شود. همچنین در این فصل کلاس‌ها را شناسایی نمودیم و در مورد صفات و عملیات‌های هر یک توضیح مختصری ارائه دادیم. در فصل پیاده‌سازی یکی یکی در مورد نحوه‌ی پیاده‌سازی کلاس‌ها با زبان ++c و در محیط Qt Creator توضیحات مختصری ارائه کردیم و کدهای کلاس‌ها را در قالب شکل‌های مختلف نشان دادیم.

۴-۲- پیشنهادات

در همه کلاس‌ها برای نگهداری تعداد زیادی از یک شی از ساختمان داده‌ی آماده‌ی `vector` استفاده کردیم. پیشنهاد می‌شود برای نسخه‌های بعدی یک ساختمان داده‌ی اختصاصی برای برنامه سرور به جای ساختمان داده‌ی `vector` توسعه داده شود. همچنین برنامه سرور فعلی فقط از زمین بازی ساده-زمین بازی که فقط دیوارهای حاشیه‌ای دارد و دیوارهای داخلی ندارد- پشتیبانی می‌کند به همین دلیل پیشنهاد می‌شود قابلیت انجام بازی در زمین‌های بازی غیرساده نیز در نسخه‌های بعدی به برنامه سرور اضافه شود. در پرتکل ارتباطی این حالت در نظر نگرفته شده است که بازیکن در حین انجام بازی بدون این که بازی را باخته باشد از بازی خارج شود. این مسئله برای زمانی که بازیکن احراز اصالت شده و هنوز زمینش را انتخاب نکرده نیز صادق است. پیشنهاد دیگر این است حالت‌های گفته شده در پرتکل ارتباطی مد نظر قرار گیرند و برای آن‌ها پیام مناسبی طراحی و در برنامه سرور پیاده‌سازی گردد.

منابع و مأخذ

[۱] www.wikipedia.org

[۲] www.p30.download.com

[۳] شبکه‌های کامپیوتری، نوشته‌ی آندرواس.تننباوم و دیوید ج.وودرال، ویراست پنجم، ترجمه دکتر قدرت‌الله سپیدنام و مهندس عین‌الله جعفرنژاد قمی، جلد دوم