Министерство образования Республики Беларусь

Учреждение образования «Белорусский государственный университет информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

Кафедра программного обеспечения информационных технологий

Дисциплина: Компьютерные системы и сети (КСиС)

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовому проекту

на тему:

«2D ВИДЕОИГРА ПО ЛОКАЛЬНОЙ СЕТИ

В ЖАНРЕ ШУТЕРА»

БГУИР КП 1-40 01 01 114 ПЗ

Студент: гр. 151001 Матюшенко В. А.

Руководитель: асс. Шамына А. Ю.

Минск 2023

СОДЕРЖАНИЕ

ВВЕДЕНИЕ 5

1. АНАЛИЗ ПРОТОТИПОВ, ЛИТЕРАТУРНЫХ СТОЧНИКОВ И ФОРМИРОВАНИЕ ТРЕБОВАНИЙ К ПРОЕКТИРУЕМОМУ ПРОГРАММНОМУ СРЕДСТВУ 6

2. АНАЛИЗ ТРЕБОВАНИЙ К ПРОГРАММНОМУ СРЕДСТВУ И РАЗРАБОТКА ФУНКЦИОНАЛЬНЫХ ТРЕБОВАНИЙ 8

3. ПРОЕКТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА 9

3.1 Проектирование взаимодействия серверной и клиентской частей 9

3.2 Проектирование серверной части 10

3.3 Проектирование клиентской части 11

4. КОНСТРУИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА 12

5. ТЕСТИРОВАНИЕ, ПРОВЕРКА РАБОТОСПОСОБНОСТИ И АНАЛИЗ ПОЛУЧЕННЫХ РЕЗУЛЬТАТОВ 14

6. РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ ПРОГРАММЫ 16

ЗАКЛЮЧЕНИЕ 19

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 20

ПРИЛОЖЕНИЕ 21

Исходный код модуля main серверной части 21

Исходный код модуля main клиентской части 27

# ВВЕДЕНИЕ

На сегодняшний день одним из наиболее востребованных направлением в программировании является разработка компьютерных игр. Так как сейчас наиболее популярными играми являются шутеры с режимом мультиплеера, для получения опыта в этой области в качестве темы моего курсового проекта я выбрал видеоигру по локальной сети в жанре шутера. Моя цель заключается в разработке динамической игры, где персонаж должен постоянно бегать, прыгать, а также взаимодействовать с игровым миром: лазать по лестницам и стрелять во врагов. Помимо этого одной из главных особенностей игры будет являться то, что в неё может играть сразу несколько людей, находящихся в одной локальной сети и подключившихся к одному серверу.

В процессе написания курсового проекта я планирую освоить методы взаимодействия программ по сети. Помимо этого я бы хотел, чтобы программа взаимодействовала с файлами, в которых будет вся информация о картах, анимациях и картинки. Так как логика игры делится на две стороны, а именно клиент и сервер, то мне необходимо построить логику на стороне клиента так, чтобы у игрока была возможность подключаться к серверам с разными картами.

В настоящей пояснительной записке отражены следующие этапы написания курсового проекта:

1. Анализ прототипов, литературных источников и формирование требований к проектируемому программному средству.

2) Анализ требований к программному средству и разработка функциональных требований.

3) Проектирование программного средства.

4) Конструирование программного средства.

5) Тестирование, проверка работоспособности и анализ полученных результатов.

6) Руководство пользователя программы.

# АНАЛИЗ ПРОТОТИПОВ, ЛИТЕРАТУРНЫХ СТОЧНИКОВ И ФОРМИРОВАНИЕ ТРЕБОВАНИЙ К ПРОЕКТИРУЕМОМУ ПРОГРАММНОМУ СРЕДСТВУ

За идею я взял известную компьютерную игру в жанре шутера от первого лица «Call Of Duty 4: Modern Warfare», которая была разработана [американской](https://ru.wikipedia.org/wiki/%D0%A1%D0%BE%D0%B5%D0%B4%D0%B8%D0%BD%D1%91%D0%BD%D0%BD%D1%8B%D0%B5_%D0%A8%D1%82%D0%B0%D1%82%D1%8B_%D0%90%D0%BC%D0%B5%D1%80%D0%B8%D0%BA%D0%B8" \o "Соединённые Штаты Америки) компанией [Infinity Ward](https://ru.wikipedia.org/wiki/Infinity_Ward" \o "Infinity Ward) и издана [Activision](https://ru.wikipedia.org/wiki/Activision" \o "Activision). Помимо прохождения сюжета, игрок может также принять участие в многопользовательской игре по локальной сети или интернету. До сих пор люди проводят временя на некоторых серверах, играя в разных режимах.



Рисунок 1.1 - Список серверов

Помимо лекционного материала, я глубже изучил принцип работы многопользовательской игры [1]. Также я поиграл на нескольких серверах и проанализировал игровой мир, чтобы сформировать список требований для моего программного средства.

При подключении к серверу в случае, когда у клиента нет данных о карте, на которой идёт игра на сервере, клиент автоматически скачивает данные с сервера. После обмена информацией о карте игрок выбирает команду. Во всех режимах (за исключением свободной игры, там каждый играет сам за себя) команды две. Если количество игроков в командах различается больше, чем на один, то происходит автобаланс - последнего подключившегося игрока из команды с большим числом игроков переводят в другую команду, что позволяет уровнять бой. Бой ведётся, пока не истечёт время или не наберётся определённое количество очков. Очки начисляются за убийство врага. После отключения от сервера есть возможность выбрать другой, не выходя из игры.



Рисунок 1.2 - Игровой мир

После проведения анализа, я решил, что моё программное средство будет работать по схожему принципу. Единственное принципиальное отличие заключается в том, что я буду проектировать 2D игру от третьего лица вместо 3D от первого лица, так как в этом проекте основной целью является взаимодействие игроков по сети. Поэтому я планирую опустить графику на второй план, чтобы сосредоточить силы на написание логики сервера и клиента.

На стороне сервера необходимо иметь файлы с данными о карте, включая текстуру (она понадобится исключительно для передачи клиенту). После инициализации сервер должен принимать новых игроков, не прерывая обновление уже играющих. При подключении к серверу клиент должен удостоверится в наличии данных о карте, и при необходимости сервер должен разрешить скачивание недостающих данных. После этого сервер даёт возможность игроку выбрать команду и ввести своё имя. Если игрок ввёл уже существующее имя, к строке с его именем прибавится «\_2» в конце, а сам игрок уведомится об этом. При подключении или отключении какого-нибудь игрока сервер обязан уравнять команды по игрокам.

На стороне клиента должна быть возможность выбрать сервер путём ввода его локального IP адреса и порта. После скачивания недостающих файлов, ввода имени и выбора команды игрок может приступить к игре. Помимо передвижения влево и вправо игрок может прыгать, лазать по лестницам, а также стрелять по врагам. После смерти игрок будет возрождаться не сразу, а после нажатия определённой клавиши. Кроме игрового мира на экране будут отображатся игроки двух команд, пули, счёт и здоровье игрока. Игрок будет иметь возможность в любой момент отключиться от сервера и подключится к другому, не закрывая приложение.

# АНАЛИЗ ТРЕБОВАНИЙ К ПРОГРАММНОМУ СРЕДСТВУ И РАЗРАБОТКА ФУНКЦИОНАЛЬНЫХ ТРЕБОВАНИЙ

Исходя из поставленной задачи и сформулированных требований, сервер должен выполнять следующие функции:

- возможность настроить сервер на определённую карту;

- подключение новых игроков без прерывания обновления других;

- проверить при подключении наличие всех файлов на клиенте;

- автораспределение при подключении или отключении игрока и разнице игроков в командах более чем один;

- проверить на уникальность введённое имя игрока и при обратном прибавить к строке с именем «\_2»;

- приём данных о нажатых клавишах, обновление позиции и отправка полученной игровой ситуации всем игрокам;

- реализация передвижения игрока влево и вправо, прыжка, лазанья по лестницам, стрельбы, получения урона, возрождения по готовности;

- реалистичное обновление положений объектов.

На стороне клиента обязаны быть реализованными следующие функции:

- возможность выбора сервера;

- поддерживать любые карты и любые текстуры;

- отображать всех игроков, пули, счёт и шкалу здоровья;

- отправка данных о нажатых клавишах серверу, приём игровой ситуации и её отображение.

# ПРОЕКТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА

## 3.1 Проектирование взаимодействия серверной и клиентской частей

Для того, чтобы у игроков был выбор, можно реализовать одинаковые сервера, но доверить пользователю привязку их к разным картам. Подключение клиента к серверу будет включать следующие этапы.

Сначала сервер проверяет наличие на клиенте файла с информацией о карте, текстуры и заднего фона. Если у клиента отсутствует какой-либо файл, то производится скачивание недостающего файла с сервера. После этого сервер показывает текущее количество игроков в командах и даёт возможность клиенту выбрать за какую команду играть. В случае, когда разница между количествами игроков в командах больше одного, происходит автобаланс, о чём сообщается клиенту. Далее клиент вводит своё имя, а сервер проверяет его на уникальность. Если это имя уже использовано, то к строке с именем прибавляется «\_2», а клиент об этом уведомляется. После этого клиент становится новым игроком - его данные о нажатых клавишах принимаются сервером, обрабатываются и результат высылается всем игрокам. Если на каком-то из этапов, включая этап самой игры, соединение прерывается, то после получения уведомления о данной ситуации клиент может начать подключение сначала. Алгоритм подключения клиента к серверу показан на рисунке 3.1.

На этапах подключения будет использоваться протокол TCP, так как он обеспечивает целостность данных, что здесь является обязательным. Если во время ввода данных задержка на приём и передачу незначительна, то во время самой игры необходимо эту задержку минимизировать. Для передачи данных во время игры будет использоваться протокол UDP.

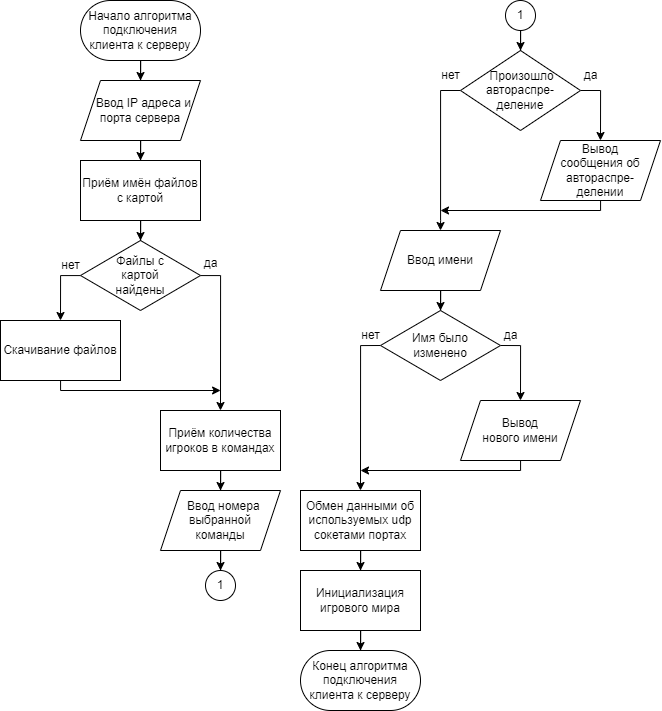


Рисунок 3.1 - Алгоритм подключения клиента к серверу

## 3.2 Проектирование серверной части

Инициализация сервера заключается в следующем:

- ввод пользователя путей к файлам с данными о карте;

- считывание данных о карте с XML файла и получение из них двумерной матрицы объектов;

- создание массива клиентов и массива пуль;

- создание двух потоков: один для подключения новых клиентов, другой для обновления уже подключившихся.

Для обновления положения персонажей сервер должен знать исключительно объектную характеристику карты, тоесть где какой объект находится. Чтобы при подключении нового клиента или появлении новой пули не создавать объект, а при отключении клиента или взрыва пули не удалять его, можно использовать массив созданных объектов, переинициализируя его вместо создания. Подключение новых клиентов может затормозить обновление играющих, особенно если высылаются файлы. По этой причине для каждой задачи создан отдельный поток.

Пакет, пересылаемый сервером клиенту, содержит следующие значения: счёт первой команды, счёт второй команды, количество клиентов, массив пакетов клиента, кооличество пуль, массив пакетов пули, индекс клиента.

Пакет клиента содержит в себе: команду, две координаты, здоровье, состояние (бег, прыжок и так далее) и другие вспомогательные переменные для корректного показа анимаций. Пакет пули содержит её координаты вспомогательную переменную для корректного показа анимаций. Индекс клиента для каждого клиента уникален. Он является индексом элемента массива пакетов клиента и указывает на элемент, которым «управляет» данный клиент.

## 3.3 Проектирование клиентской части

Инициализация клиента содержит следующие этапы:

- считывание данных об анимациях персонажа и пули с XML файла;

- загрузка картинки для шкалы здоровья и шрифта для счёта;

- создание массива игроков и массива пуль.

На стороне клиента также как и на сервере необходимо заранее подготовить два массива для хранения информации об игроках и пулях. После подключения к серверу загружается карта.

Пакет, пересылаемый клиентом серверу, содержит в себе битовую маску нажатых клавиш.

4. КОНСТРУИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА

Языком программирования программного средства был выбран язык C++. Данный язык широко используется при разработке игр, поскольку позволяет повысить производительость, сравнительно с другими языками. Дополнительно я использовал стороннюю библиотеку SFML-2.5.1 [2], чтобы упростить работу с созданием окна, прорисовкой карты и аминаций, а также проигрыванием музыки и звуков. В этой библиотеки также есть модуль для работы с сетями. Для считывания XML файлов я подключил модуль с парсером TinyXML-2 [3]. В качестве среды разработки была выбрана среда Visual Studio 2022.

Для упрощения задачи пользовательский интефейс серверной части будет реализован в консольном режиме. Интефейс клиентской части также будет в консольном режиме, но после подключения к серверу появится окно, в котором будет визуализироваться игровой мир.

Чтобы структурировать код, серверная часть была поделена на следующие модули:

- Consts. Здесь объявлены необходимые константы, такие как размеры массивов клиентов и пуль, размеры игрока и пули как объектов, коды состояний игрока и коды используемых в игре клавиш. Переменные, определяющие скорость передвижения, коэффициент свободного падения и прочие переменные, которые характеризуют движение объекта, было решено хранить как поле класса игрока, так как в будущем могут быть добавлены бонусы, позволяющие ускорять передвижение объекта или изменять гравитацию;

- Level. Здесь объявлен класс, содержащий информацию о карте. Конструктор с помощью парсера TinyXML-2 считывает информацию о карте и создаёт двумерную матрицу объектов;

- Bullet. Здесь объявлен класс, который характеризует пулю. В этом классе реализован метод, обрабатывающий коллизию пули с объектами карты и игроками;

- Player. Здесь объявлен класс, который характеризует игрока. В этом классе реализованы методы переинициализации, обновления состояния игрока и обработки коллизии с картой;

- Client. Здесь объявлен класс, который характеризует клиента. Одним из его полей является класс Player. В этом классе реализован метод, который формирует пакеты с текущим состоянием игрока для передачи другим, и метод для распаковки принятого пакета. Здесь также создаются TCP и UDP сокеты;

- main. Здесь происходит инициализация и создаются два потока (для подключения новых клиентов и для обновления уже подключившихся). Для каждого из потоков реализована своя функция.

Для структурирования кода клиентская часть была также поделена на модули:

- Consts. Здесь объявлены те же константы, что и на сервере;

- Level. Здесь объявлен класс, содержащий информацию о карте. Конструктор с помощью парсера TinyXML-2 считывает информацию о карте и создаёт вектор спрайтов, которой используется для отображения карты;

- Animation. Здесь объявлены классы Animation и AnimationManager. Первый содержит в себе векторы с координатами анимаций движения в одну сторону и другую. В этом классе реализован метод, обновляющий анимацию. Второй класс с помощью парсера TinyXML-2 считывает информацию об анимациях и загружает спрайты. Этот класс содержит словарь со значениями типа Animation. AnimationManager отвечает за проигрывание нужной анимации;

- Bullet. Здесь объявлен класс, который характеризует анимацию пулю. Одним из его полей является класс AnimationManager. В этом классе реализован метод выбора соответствующей состоянию пули анимации;

- Player. Здесь объявлен класс, который характеризует анимацию игрока. Одним из его полей является класс AnimationManager. В этом классе реализован метод выбора соответствующей состоянию игроку анимации;

- HealthBar. Здесь объявлен класс, который визуализирует количество здоровья игрока;

- Score. Здесь объявлен класс, который визуализирует счёт;

- main. Здесь происходит инициализация и осуществляется подключение к серверу с последующим обновлением состояния игры.

# 5. ТЕСТИРОВАНИЕ, ПРОВЕРКА РАБОТОСПОСОБНОСТИ И АНАЛИЗ ПОЛУЧЕННЫХ РЕЗУЛЬТАТОВ

Тестирование программного средства было разбито на две части: тестирование серверной части и тестирование клиентской части. Специфики тестирований и результаты отражены в следующих таблицах.

Таблица 5.1 - Тестирование серверной части

|  |  |  |
| --- | --- | --- |
| Специфика тестирования | Ожидаемый результат | Полученный результат |
| Подключение клиента | Подключение по известному алгоритму. Далее сервер начинает отправлять пакеты с игровой ситуацией | Тест пройден |
| Отключение клиента | Сервер забывает клиента и продолжает работать | Тест пройден |
| Отсутствие файла на стороне клиента | Сервер успешно пересылает нужный файл | Тест пройден |
| Приём данных от клиентов и отправка данных клиентам | Сервер устанавливает состояние игрока по полученной битовой маске, обновляет состояние всех игроков с учётом прошедшего времени, отправляет всем клиентам полученную игровую ситуацию | Тест пройден |
| Взаимодействие игрока с игровым миром | Сервер корректно обрабатывает коллизию, прыжки, лазание по лестницам, выстрелы, а также возрождает игрока в определённом месте, когда тот нажмёт нужную клавишу | Тест пройден |
| Взаимодействие пули с игровым миром | Сервер корректно обрабатывает коллизию и с объектами карты, и с игроками. При коллизи с игроком, чья команда отличается от команды игрока, выпустившего пулю, у первого уменьшается здоровье | Тест пройден |
| Ошибочная ситуация при загрузке файлов | Сервер сообщает, в чём заключается ошибка, и прекращает работу | Тест пройден |

Таблица 5.2 - Тестирование клиентской части

|  |  |  |
| --- | --- | --- |
| Специфика тестирования | Ожидаемый результат | Полученный результат |
| Подключение к серверу | Подключение по известному алгоритму. Далее клиент начинает получать пакеты с игровой ситуацией | Тест пройден |
| Отключение сервера | Прекращение игрового процесса. Клиент переход на начальный этап алгоритма | Тест пройден |
| Отсутствие файла | Клиет успешно принимает нужный файл. Если при передаче произошла ошибка, то файл удаляется | Тест пройден |
| Приём данных от сервера и отправка данных на сервер | Клиент принимает пакет с игровой ситуацией, визуализирует её, создаёт битовую маску нажатых клавиш и отправляет её серверу | Тест пройден |
| Визуализация игровой ситуации | Клиент корректно визуализирует карту, всех игроков, пули, счёт и шкалу здоровья | Тест пройден |
| Ошибочная ситуация при загрузке файлов | Клиент сообщает, в чём заключается ошибка, и прекращает работу | Тест пройден |

Проведённые тесты показали корректное выполнение всех требуемых действий. В ходе итогового тестирования не было выявлено каких-либо ошибок или некорректной работы программного средства.

6. РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ ПРОГРАММЫ

Сначала нужно удостоверится, что сервер и клиенты находятся в одной локальной сети. Для того, чтобы можно было начать игру, необходимо настроить сервер и подключить к нему клиентов. На стороне сервера пользователю необходимо удостоверится в наличии 4 фалов: в папке files должны быть файл карты с расширение tmx и файл настройки тайлсета с расширение tsx; здесь же в папке images файл тайлсета с расширением png и файл картинки заднего фона с расширением png. После запуска сервера нужно ввести пути трёх файлов: файл карты с расширение tmx, файл тайлсета с расширением png, файл картинки заднего фона с расширением png. Теперь сервер настроен на проведение игры на заданной карте. Этому будет свидетельствовать появление значений IP адреса и порта, которые позже введут клиенты. При подключении и отключении игроков будет выводится соответствующая информация.

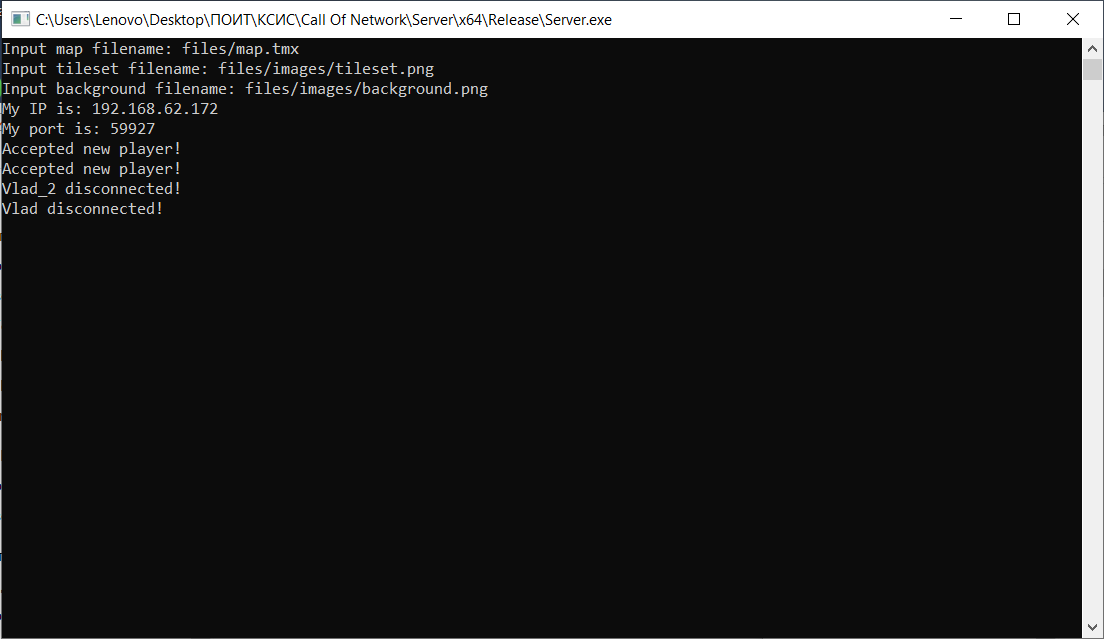


Рисунок 5.1 - Пользовательский интерфейс сервера после приёма и отключения двух клиентов

На стороне клиента необходимо ввести IP адрес и порт сервера, на который осуществится подключение. Если у клиента отсутствуют некоторые файлы, они автоматически скачаются и уведомят пользователя. После получения информации о количествах человек в командах, пользователь может выбрать команду. Если его перенесло в другую команду из-за автобаланса, появится соответствующее уведомление. Далее пользователю предлогается ввести имя. В случае совпадения его имени с именем уже подключённого клиента к строке имени прибавится «\_2» и выведется соответствующее сообщение. После этого появится окно с игровым миром. Пользователь может перейти к игре.

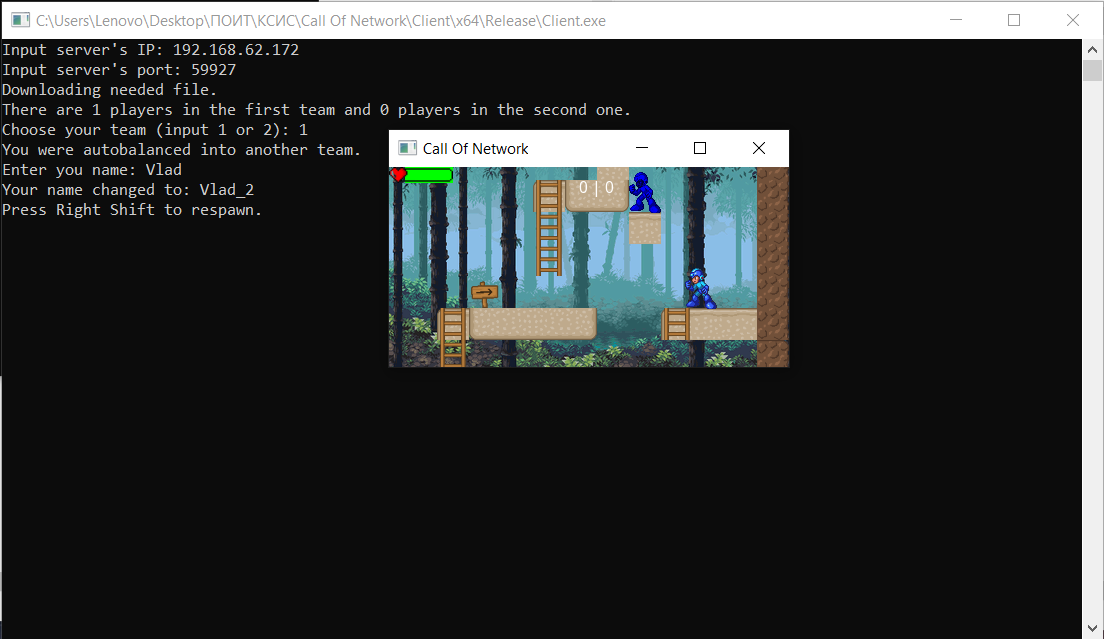


Рисунок 5.2 - Пользовательский интерфейс клиента после подключения к серверу

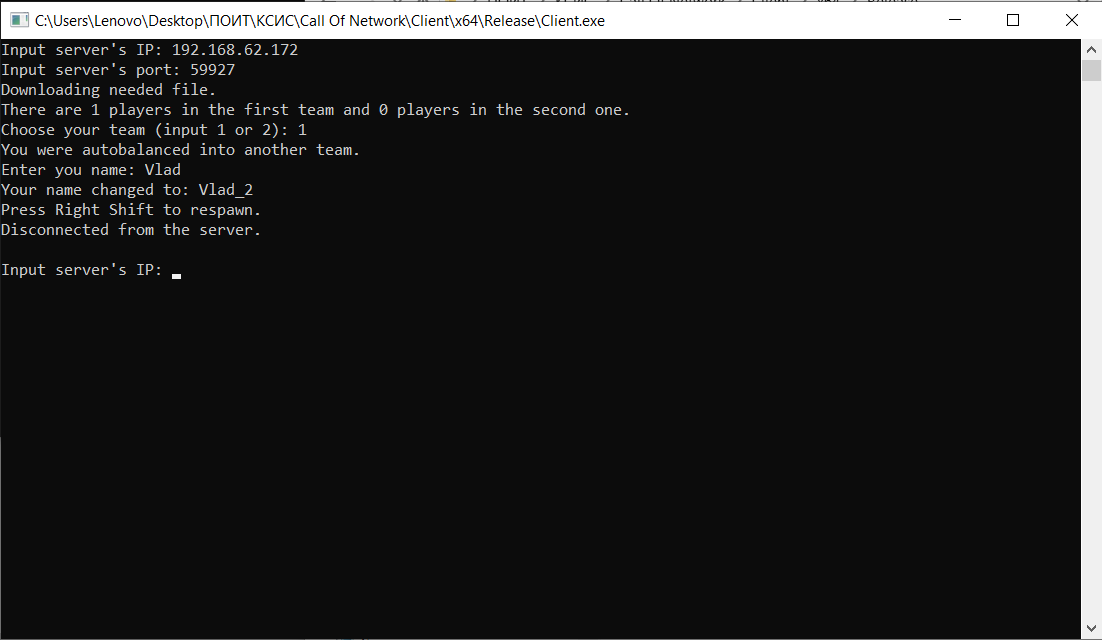


Рисунок 5.3 - Пользовательский интерфейс клиента после отключения от сервера

Управление персонажем осуществляется клавишами стрелками: влево и вправо - бег влево и вправо соответственно; вверх - прыжок. Если персонаж стоит около лестницы, то стрелками вверх и вниз он может начать подъём и спуск соответственно. Чтобы слезть с лестницы, нужно выйти за её границы используя клавиши стрелки. После этого персонаж подпрыгнет, если не оказался на поверхности. Для стрельбы нужно нажать клавишу пробел. Пуля полетит в ту сторону, в которую смотрит персонаж. Когда персонаж на лестнице, он не может стрелять. Пули наносят урон только персонажам противоположной команды. После смерти или подключения персонаж может возродится по нажатию клавиши правый шифт. Шкала здоровья находится слева вверху и отображает количество здоровья следующим образом: чем длинее и зеленее шкала, тем больше здоровья; чем короче и краснее шкала, тем меньше здоровья. Счёт показан посередине сверху. Первое число - количество убийств первой камандой, второе число - количество убийств второй камандой. Участники первой команды показаны тёмно-синим цветом, второй - зелёным. Сам игрок показан светло-синим цветом.



Рисунок 5.4 - Игровой мир



Рисунок 5.5 - Игровой мир со стороны другого клиента

# ЗАКЛЮЧЕНИЕ

В пояснительно записке я отразил все этапы разработки моего программного средства. На каждом из этих этапов я постарался показать ход моих мыслей и объяснить причины принятых решений. В результате работы над курсовым проектом было сконструировано и разработано программное средство, являющееся 2D видеоигрой по локальной сети в жанре шутера. Разработанное программное средство в полной мере выполняет все операции, заданные в условии курсового проекта, работает корректно и без сбоев.

После анализа приведённого аналога и выявления необходимых требований к моему программному средству, были спроектированы серверная и клиентская части. Для создания программного средства было проведено ознакомление с различными современными технологиями, освоены некоторые библиотеки, а именно SFML и TinyXML-2. В ходе тестирования было показано, что программное средство не подведёт пользователя ни при каких обстоятельствах.

К сожалению, из-за кратких сроков пришлось отложить реализацию нескольких режимов игры, бонусов, которые бы изменяли высоту прыжка или скорость бега персонажа, подсчёт пуль и остальных идей. Но несмотря на это, проделанная работа принесла мне много опыта, необходимого в моей будущей профессии.

Работа была разделена на этапы, такие как анализ существующих решений, постановка требований к проектируемому программному средству, разработка алгоритма и его схемы, конструирование программного средства, отладка и тестирование. После последовательного выполнения вышеперечисленных этапов разработки было получено исправно работающее программное средство.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. <https://youtu.be/0LxyqYV86Fk> [Электронный ресурс] – Как устроен мультиплеер - подробно | Networking.
2. [https://www.sfml-dev.org/documentation/2.5.1](https://www.sfml-dev.org/documentation/2.5.1/) [Электронный ресурс] –Документация по библиотеке SFML-2.5.1.
3. <https://github.com/leethomason/tinyxml2> [Электронный ресурс] – репозиторий с модулями и документацией TinyXML-2.

# ПРИЛОЖЕНИЕ

## Исходный код модуля main серверной части

#include <SFML/Graphics.hpp>

#include <SFML/Network.hpp>

#include <thread>

#include <iostream>

#include <fstream>

#include "Consts.hpp"

#include "Level.hpp"

#include "Bullet.hpp"

#include "Player.hpp"

#include "Client.hpp"

using namespace sf;

void newClient(Level\* level, Client\*\* clients, unsigned char\* team1, unsigned char\* team2, unsigned char\* clientsNumber);

void updateClients(Level\* level, Client\*\* clients, unsigned char\* team1, unsigned char\* team2, unsigned char\* clientsNumber);

int main()

{

std::string mapFilename = "", tilesetFilename = "", backgroundFilename = "";

while (backgroundFilename == "")

{

bool f = true;

std::string path;

while (f)

{

if (mapFilename == "")

{

std::cout << "Input map filename: ";

}

else

{

if (tilesetFilename == "")

{

std::cout << "Input tileset filename: ";

}

else

{

std::cout << "Input background filename: ";

}

}

std::cin >> path;

std::ifstream file(path, std::ios::binary);

if (file.good())

{

file.close();

f = false;

}

else

{

std::cout << "No file found! Try again.\n";

}

}

if (mapFilename == "")

{

mapFilename = path;

}

else

{

if (tilesetFilename == "")

{

tilesetFilename = path;

}

else

{

backgroundFilename = path;

}

}

}

Level level(mapFilename, tilesetFilename, backgroundFilename);

mapFilename = "";

tilesetFilename = "";

backgroundFilename = "";

Client\*\* clients = new Client\*[CLIENTS\_SIZE];

for (int i = 0; i < CLIENTS\_SIZE; i++)

{

clients[i] = new Client(level);

}

unsigned char clientsNumber = 0, team1 = 0, team2 = 0;

std::thread tNewClient(newClient, &level, clients, &team1, &team2, &clientsNumber);

std::thread tUpdateClients(updateClients, &level, clients, &team1, &team2, &clientsNumber);

tNewClient.join();

tUpdateClients.join();

return 0;

};

/////////////////////////////////////////////////////////////////////////////////////////////////

void newClient(Level\* level, Client\*\* clients, unsigned char\* team1, unsigned char\* team2, unsigned char\* clientsNumber)

{

TcpListener listener;

Packet rPacket, sPacket;

listener.listen(Socket::AnyPort);

std::cout << "My IP is: " << IpAddress::getLocalAddress().toString() << "\nMy port is: " << listener.getLocalPort();

while (true)

{

for (int i = 0; i < CLIENTS\_SIZE; i++)

{

switch (clients[i]->stage)

{

case Stage::Error:

{

rPacket.clear();

std::cout << "\n" << clients[i]->name << " disconnected!";

clients[i]->disconnect();

break;

}

case Stage::Connection:

{

if (listener.accept(clients[i]->tcpSocket) == Socket::Done)

{

std::cout << "\nAccepted new player!";

if (listener.isBlocking())

{

listener.setBlocking(false);

}

clients[i]->tcpSocket.setBlocking(false);

clients[i]->filename = level->mapFilename;

clients[i]->stage = Stage::FileAsk;

}

break;

}

case Stage::FileAsk:

{

Socket::Status s = clients[i]->tcpSocket.receive(rPacket);

if (s == Socket::Disconnected || s == Socket::Error)

{

clients[i]->stage = Stage::Error;

break;

}

if (s == Socket::Done)

{

rPacket.clear();

sPacket << clients[i]->filename;

clients[i]->tcpSocket.send(sPacket);

sPacket.clear();

clients[i]->stage = Stage::FileAnswer;

}

break;

}

case Stage::FileAnswer:

{

Socket::Status s = clients[i]->tcpSocket.receive(rPacket);

if (s == Socket::Disconnected || s == Socket::Error)

{

clients[i]->stage = Stage::Error;

break;

}

if (s == Socket::Done)

{

bool need;

rPacket >> need;

rPacket.clear();

if (need)

{

clients[i]->tcpSocket.setBlocking(true);

std::ifstream file(clients[i]->filename, std::ios::binary);

file.seekg(0, std::ios::end);

int fileSize = file.tellg();

file.close();

sPacket << fileSize;

clients[i]->tcpSocket.send(sPacket);

sPacket.clear();

file.open(clients[i]->filename, std::ios::binary);

char buffer[1024];

while (file.good())

{

file.read(buffer, sizeof(buffer));

clients[i]->tcpSocket.send(buffer, file.gcount());

}

clients[i]->tcpSocket.setBlocking(false);

}

if (clients[i]->filename == level->backgroundFilename)

{

clients[i]->stage = Stage::TeamAsk;

break;

}

if (clients[i]->filename == level->tilesetTsxFilename)

{

clients[i]->filename = level->backgroundFilename;

}

if (clients[i]->filename == level->tilesetFilename)

{

clients[i]->filename = level->tilesetTsxFilename;

}

if (clients[i]->filename == level->mapFilename)

{

clients[i]->filename = level->tilesetFilename;

}

clients[i]->stage = Stage::FileAsk;

}

break;

}

case Stage::TeamAsk:

{

Socket::Status s = clients[i]->tcpSocket.receive(rPacket);

if (s == Socket::Disconnected || s == Socket::Error)

{

clients[i]->stage = Stage::Error;

break;

}

if (s == Socket::Done)

{

sPacket << \*team1;

sPacket << \*team2;

clients[i]->tcpSocket.send(sPacket);

sPacket.clear();

clients[i]->stage = Stage::TeamAnswer;

}

break;

}

case Stage::TeamAnswer:

{

Socket::Status s = clients[i]->tcpSocket.receive(rPacket);

if (s == Socket::Disconnected || s == Socket::Error)

{

clients[i]->stage = Stage::Error;

break;

}

if (s == Socket::Done)

{

rPacket >> clients[i]->team;

rPacket.clear();

bool autoBalanced = false;

if (clients[i]->team)

{

(\*team1)++;

if (\*team1 - \*team2 > 1)

{

(\*team1)--;

(\*team2)++;

clients[i]->team = false;

autoBalanced = true;

}

}

else

{

(\*team2)++;

if (\*team2 - \*team1 > 1)

{

(\*team2)--;

(\*team1)++;

clients[i]->team = true;

autoBalanced = true;

}

}

sPacket << autoBalanced;

clients[i]->tcpSocket.send(sPacket);

sPacket.clear();

clients[i]->stage = Stage::NameAsk;

}

break;

}

case Stage::NameAsk:

{

Socket::Status s = clients[i]->tcpSocket.receive(rPacket);

if (s == Socket::Disconnected || s == Socket::Error)

{

clients[i]->stage = Stage::Error;

break;

}

if (s == Socket::Done)

{

rPacket >> clients[i]->name;

rPacket.clear();

bool anotherName = false;

for (int j = 0; j < CLIENTS\_SIZE; j++)

{

if (i != j && clients[i]->name == clients[j]->name)

{

anotherName = true;

clients[i]->name = clients[i]->name + "\_2";

break;

}

}

sPacket << anotherName;

sPacket << clients[i]->name;

clients[i]->tcpSocket.send(sPacket);

sPacket.clear();

clients[i]->stage = Stage::PortAsk;

}

break;

}

case Stage::PortAsk:

{

Socket::Status s = clients[i]->tcpSocket.receive(rPacket);

if (s == Socket::Disconnected || s == Socket::Error)

{

clients[i]->stage = Stage::Error;

break;

}

if (s == Socket::Done)

{

rPacket >> clients[i]->clientPort;

rPacket.clear();

clients[i]->udpSocket.setBlocking(false);

clients[i]->udpSocket.bind(Socket::AnyPort);

clients[i]->ip = clients[i]->tcpSocket.getRemoteAddress();

clients[i]->serverPort = clients[i]->udpSocket.getLocalPort();

sPacket << clients[i]->serverPort;

clients[i]->tcpSocket.send(sPacket);

sPacket.clear();

clients[i]->stage = Stage::Playing;

(\*clientsNumber)++;

}

break;

}

}

}

}

}

/////////////////////////////////////////////////////////////////////////////////////////////////

void updateClients(Level\* level, Client\*\* clients, unsigned char\* team1, unsigned char\* team2, unsigned char\* clientsNumber)

{

Packet rPacket, sPacket;

Bullet\*\* bullets = new Bullet \* [BULLETS\_SIZE];

for (int i = 0; i < BULLETS\_SIZE; i++)

{

bullets[i] = new Bullet(NULL\_VECTOR2f, 10, false, true);

}

Vector2f playersCoord[CLIENTS\_SIZE];

Vector2i bulletHitVec;

unsigned char bulletsNumber = 0, clientIndex = 0, score1 = 0, score2 = 0;

Clock clock;

signed \_\_int32 time;

while (true)

{

if (clock.getElapsedTime().asMilliseconds() > MSPF)

{

time = clock.getElapsedTime().asMilliseconds();

clock.restart();

if (time > MSPF \* 2)

{

time = MSPF \* 2;

}

for (int i = 0; i < CLIENTS\_SIZE; i++)

{

if (clients[i]->stage == Stage::Playing)

{

Socket::Status s = clients[i]->tcpSocket.receive(rPacket);

if (s == Socket::Disconnected || s == Socket::Error)

{

rPacket.clear();

std::cout << "\n" << clients[i]->name << " disconnected!";

clients[i]->disconnect();

if (clients[i]->team)

{

(\*team1)--;

}

else

{

(\*team2)--;

}

(\*clientsNumber)--;

break;

}

IpAddress ip;

if (clients[i]->udpSocket.receive(rPacket, ip, clients[i]->serverPort) == Socket::Done)

{

clients[i]->player.receivePacket(&rPacket);

rPacket.clear();

}

}

if (clients[i]->stage == Stage::Playing)

{

clients[i]->player.update(time, \*level);

if (clients[i]->player.respawn)

{

clients[i]->newPlayer(\*level, 100);

}

if (clients[i]->player.shoot)

{

for (int j = 0; j < BULLETS\_SIZE; j++)

{

if (!bullets[j]->isAlive)

{

bullets[j]->newBullet(clients[i]->getBulletVec(), 10, clients[i]->player.left, clients[i]->team);

bulletsNumber++;

break;

}

}

}

playersCoord[i] = clients[i]->player.getVec();

}

else

{

playersCoord[i] = NULL\_VECTOR2f;

}

}

for (int i = 0; i < BULLETS\_SIZE; i++)

{

if (bullets[i]->isAlive)

{

bulletHitVec = bullets[i]->update(time, \*level, playersCoord);

if (bulletHitVec != NULL\_VECTOR2I)

{

if (clients[bulletHitVec.y]->team != bullets[i]->team && clients[bulletHitVec.y]->player.isAlive)

{

bullets[i]->isAlive = false;

if (clients[bulletHitVec.y]->player.hit(bulletHitVec.x))

{

if (bullets[i]->team)

{

score1++;

}

else

{

score2++;

}

}

bulletsNumber--;

}

}

else

{

if (!bullets[i]->isAlive)

{

bulletsNumber--;

}

}

}

}

sPacket << score1;

sPacket << score2;

sPacket << \*clientsNumber;

for (int i = 0; i < CLIENTS\_SIZE; i++)

{

if (clients[i]->stage == Stage::Playing)

{

clients[i]->createPacket(&sPacket);

}

}

sPacket << bulletsNumber;

for (int i = 0; i < BULLETS\_SIZE; i++)

{

if (bullets[i]->isAlive)

{

bullets[i]->createPacket(&sPacket);

}

}

clientIndex = 0;

for (int i = 0; i < CLIENTS\_SIZE; i++)

{

if (clients[i]->stage == Stage::Playing)

{

Packet pPacket = sPacket;

pPacket << clientIndex;

clients[i]->udpSocket.send(pPacket, clients[i]->ip, clients[i]->clientPort);

clientIndex++;

}

}

sPacket.clear();

}

}

}

## Исходный код модуля main клиентской части

#include <SFML/Graphics.hpp>

#include <SFML/Network.hpp>

#include <iostream>

#include <fstream>

#include "Consts.hpp"

#include "Level.hpp"

#include "Animation.hpp"

#include "Player.hpp"

#include "Bullet.hpp"

#include "HealthBar.hpp"

#include "Score.hpp"

using namespace sf;

int main()

{

enum Stage { Connection, FileAsk, FileAnswer, TeamAsk, TeamAnswer, NameAsk, NameAnswer, PortAsk, PortAnswer, Playing, Error };

IpAddress serverIp;

unsigned short serverPort, myPort;

Stage stage = Stage::Connection;

TcpSocket tcpSocket;

UdpSocket udpSocket;

Packet rPacket, sPacket;

float offsetX, offsetY;

RenderWindow window;

int windowWidthHalf = 0, windowHeightHalf = 0, mapWidth = 0, mapHeight = 0;

View view;

Level level;

Score score("files/OpenSans-Regular.ttf");

HealthBar healthBar("files/images/healthBar.png");

AnimationManager playerAnimationManager("files/images/megaman.png", "files/megaman.xml");

AnimationManager bulletAnimationManager("files/images/bullet.png", "files/bullet.xml");

Player\*\* players = new Player\*[CLIENTS\_SIZE];

for (int i = 0; i < CLIENTS\_SIZE; i++)

{

players[i] = new Player(playerAnimationManager);

}

Bullet\*\* bullets = new Bullet\*[BULLETS\_SIZE];

for (int i = 0; i < BULLETS\_SIZE; i++)

{

bullets[i] = new Bullet(bulletAnimationManager);

}

unsigned char playersNumber = 0, bulletsNumber = 0, myIndex = 0, playerState = 0, score1 = 0, score2 = 0;

std::string mapFilename, tilesetFilename, backgroundFilename, myName;

Clock clock;

signed \_\_int32 time;

while (true)

{

switch (stage)

{

case Stage::Error:

{

window.close();

rPacket.clear();

tcpSocket.disconnect();

new(&tcpSocket) TcpSocket;

std::cout << "Disconnected from the server.\n\n";

stage = Stage::Connection;

break;

}

case Stage::Connection:

{

std::fflush(stdin);

std::fflush(stdout);

std::cout << "Input server's IP: ";

std::string sServerIp;

std::cin >> sServerIp;

serverIp = IpAddress(sServerIp);

std::cout << "Input server's port: ";

std::cin >> serverPort;

Socket::Status s = tcpSocket.connect(serverIp, serverPort, seconds(2));

if (s == Socket::Disconnected || s == Socket::Error)

{

stage = Stage::Error;

break;

}

if (s == Socket::Done)

{

mapFilename = "";

tilesetFilename = "";

backgroundFilename = "";

myName = "";

stage = Stage::FileAsk;

sPacket << true;

tcpSocket.send(sPacket);

sPacket.clear();

}

break;

}

case Stage::FileAsk:

{

Socket::Status s = tcpSocket.receive(rPacket);

if (s == Socket::Disconnected || s == Socket::Error)

{

stage = Stage::Error;

break;

}

if (s == Socket::Done)

{

bool need = true;

std::string path;

if (mapFilename == "")

{

rPacket >> mapFilename;

path = mapFilename;

}

else

{

if (tilesetFilename == "")

{

rPacket >> tilesetFilename;

path = tilesetFilename;

}

else

{

// Use myName to store the path to tileset.tsx because it will not be needed

if (myName == "")

{

rPacket >> myName;

path = myName;

}

else

{

rPacket >> backgroundFilename;

path = backgroundFilename;

}

}

}

rPacket.clear();

std::ifstream file(path, std::ios::binary);

if (file.good())

{

file.close();

need = false;

}

else

{

std::cout << "Downloading needed file.\n";

stage = Stage::FileAnswer;

}

sPacket << need;

tcpSocket.send(sPacket);

sPacket.clear();

if (!need)

{

sPacket << true;

tcpSocket.send(sPacket);

sPacket.clear();

if (backgroundFilename != "")

{

stage = Stage::TeamAsk;

}

}

}

break;

}

case Stage::FileAnswer:

{

Socket::Status s = tcpSocket.receive(rPacket);

if (s == Socket::Disconnected || s == Socket::Error)

{

stage = Stage::Error;

break;

}

if (s == Socket::Done)

{

int fileSize;

rPacket >> fileSize;

rPacket.clear();

std::string path;

if (backgroundFilename != "")

{

path += backgroundFilename;

}

else

{

if (myName != "")

{

path += myName;

}

else

{

if (tilesetFilename != "")

{

path += tilesetFilename;

}

else

{

path += mapFilename;

}

}

}

char buffer[1024];

std::size\_t recieved = 0;

std::ofstream file(path, std::ios::binary);

while (true)

{

Socket::Status s = tcpSocket.receive(buffer, sizeof(buffer), recieved);

if (s == Socket::Disconnected || s == Socket::Error)

{

file.close();

std::remove(path.c\_str());

stage = Stage::Error;

break;

}

if (s == Socket::Done)

{

file.write(buffer, recieved);

fileSize -= recieved;

if (fileSize <= 0)

{

file.close();

if (backgroundFilename == "")

{

stage = Stage::FileAsk;

}

else

{

stage = Stage::TeamAsk;

}

sPacket << true;

tcpSocket.send(sPacket);

sPacket.clear();

break;

}

}

}

}

break;

}

case Stage::TeamAsk:

{

Socket::Status s = tcpSocket.receive(rPacket);

if (s == Socket::Disconnected || s == Socket::Error)

{

stage = Stage::Error;

break;

}

if (s == Socket::Done)

{

unsigned char uTeam = 0;

int iTeam;

rPacket >> uTeam;

iTeam = uTeam;

std::cout << "There are " << iTeam << " players in the first team and ";

rPacket >> uTeam;

rPacket.clear();

iTeam = uTeam;

std::cout << iTeam << " players in the second one. \nChoose your team (input 1 or 2): ";

std::cin >> iTeam;

while (iTeam != 1 && iTeam != 2)

{

std::cout << "Incorrect input! Try again: ";

std::cin >> iTeam;

}

bool team = iTeam == 1;

sPacket << team;

tcpSocket.send(sPacket);

sPacket.clear();

stage = Stage::TeamAnswer;

}

break;

}

case Stage::TeamAnswer:

{

Socket::Status s = tcpSocket.receive(rPacket);

if (s == Socket::Disconnected || s == Socket::Error)

{

stage = Stage::Error;

break;

}

if (s == Socket::Done)

{

bool autoBalanced;

rPacket >> autoBalanced;

rPacket.clear();

if (autoBalanced)

{

std::cout << "You were autobalanced into another team.\n";

}

stage = Stage::NameAsk;

}

break;

}

case Stage::NameAsk:

{

std::cout << "Enter you name: ";

std::cin >> myName;

while (myName.length() > 8)

{

if (myName.length() > 8)

{

std::cout << "Name is too long! Try again: ";

}

std::cin >> myName;

}

sPacket << myName;

tcpSocket.send(sPacket);

sPacket.clear();

stage = Stage::NameAnswer;

break;

}

case Stage::NameAnswer:

{

Socket::Status s = tcpSocket.receive(rPacket);

if (s == Socket::Disconnected || s == Socket::Error)

{

stage = Stage::Error;

break;

}

if (s == Socket::Done)

{

bool anotherName;

rPacket >> anotherName;

if (anotherName)

{

std::cout << "Your name changed to: ";

rPacket >> myName;

std::cout << myName + "\n";

}

rPacket.clear();

stage = Stage::PortAsk;

}

break;

}

case Stage::PortAsk:

{

udpSocket.setBlocking(false);

udpSocket.bind(Socket::AnyPort);

myPort = udpSocket.getLocalPort();

sPacket << myPort;

tcpSocket.send(sPacket);

sPacket.clear();

stage = Stage::PortAnswer;

break;

}

case Stage::PortAnswer:

{

Socket::Status s = tcpSocket.receive(rPacket);

if (s == Socket::Disconnected || s == Socket::Error)

{

stage = Stage::Error;

break;

}

if (s == Socket::Done)

{

rPacket >> serverPort;

rPacket.clear();

tcpSocket.setBlocking(false);

playerState = 0;

sPacket << playerState;

udpSocket.send(sPacket, serverIp, serverPort);

sPacket.clear();

offsetX = 0;

offsetY = 0;

new(&window) RenderWindow(VideoMode(400, 200), "Call Of Network");

windowWidthHalf = window.getSize().x / 2;

windowHeightHalf = window.getSize().y / 2;

new(&view) View(FloatRect(0, 0, window.getSize().x, window.getSize().y));

new(&level) Level(mapFilename, tilesetFilename, backgroundFilename);

mapWidth = level.mapWidth \* level.tileWidth;

mapHeight = level.mapHeight \* level.tileHeight;

new(&clock) Clock();

std::cout << "Press Right Shift to respawn.\n";

stage = Stage::Playing;

}

break;

}

case Stage::Playing:

{

Event event;

while (window.pollEvent(event))

{

if (event.type == Event::Closed)

{

window.close();

stage = Stage::Error;

}

}

Socket::Status s = tcpSocket.receive(rPacket);

if (s == Socket::Disconnected || s == Socket::Error)

{

stage = Stage::Error;

break;

}

IpAddress ip;

if (udpSocket.receive(rPacket, ip, myPort) == Socket::Done)

{

rPacket >> score1;

rPacket >> score2;

rPacket >> playersNumber;

for (int i = 0; i < playersNumber; i++)

{

players[i]->receivePacket(&rPacket);

}

rPacket >> bulletsNumber;

for (int i = 0; i < bulletsNumber; i++)

{

bullets[i]->receivePacket(&rPacket);

}

rPacket >> myIndex;

rPacket.clear();

playerState = 0;

if (Keyboard::isKeyPressed(Keyboard::Left))

{

playerState = playerState | KEY\_LEFT;

}

if (Keyboard::isKeyPressed(Keyboard::Right))

{

playerState = playerState | KEY\_RIGHT;

}

if (Keyboard::isKeyPressed(Keyboard::Up))

{

playerState = playerState | KEY\_UP;

}

if (Keyboard::isKeyPressed(Keyboard::Down))

{

playerState = playerState | KEY\_DOWN;

}

if (Keyboard::isKeyPressed(Keyboard::Space))

{

playerState = playerState | KEY\_SPACE;

}

if (Keyboard::isKeyPressed(Keyboard::RShift))

{

playerState = playerState | KEY\_RSHIFT;

}

sPacket << playerState;

udpSocket.send(sPacket, serverIp, serverPort);

sPacket.clear();

}

if (clock.getElapsedTime().asMilliseconds() > MSPF)

{

time = clock.getElapsedTime().asMilliseconds();

clock.restart();

if (time > MSPF \* 2)

{

time = MSPF \* 2;

}

for (int i = 0; i < playersNumber; i++)

{

players[i]->update(time, i == myIndex);

}

for (int i = 0; i < bulletsNumber; i++)

{

bullets[i]->update(time);

}

if (players[myIndex]->x <= windowWidthHalf) {

offsetX = windowWidthHalf - players[myIndex]->x;

}

else

{

if (players[myIndex]->x >= mapWidth - windowWidthHalf) {

offsetX = mapWidth - windowWidthHalf - players[myIndex]->x;

}

}

if (players[myIndex]->y <= windowHeightHalf) {

offsetY = windowHeightHalf - players[myIndex]->y;

}

else

{

if (players[myIndex]->y >= mapHeight - windowHeightHalf) {

offsetY = mapHeight - windowHeightHalf - players[myIndex]->y;

}

}

window.clear();

view.setCenter(players[myIndex]->x + offsetX, players[myIndex]->y + offsetY);

window.setView(view);

level.draw(window, view.getCenter());

for (int i = 0; i < playersNumber; i++)

{

if (i != myIndex)

{

players[i]->draw(window);

}

}

players[myIndex]->draw(window);

for (int i = 0; i < bulletsNumber; i++)

{

bullets[i]->draw(window);

}

healthBar.update(players[myIndex]->health);

healthBar.draw(window, view.getCenter() - Vector2f(200, 100));

score.update(score1, score2);

score.draw(window, view.getCenter() - Vector2f(0, 100));

window.display();

}

break;

}

}

}

}