Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет
имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ Информатика и системы управления

КАФЕДРА Программное обеспечение ЭВМ и информационные технологии

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К КУРСОВОЙ РАБОТЕ

НА ТЕМУ:

Многопользовательское игровое приложение
«Two Kings»

Руководитель курсового проекта _____ Рогозин Н.О.
(Подпись, дата) (И.О.Фамилия)

Студенты _____ Аминов Т.С.
(Подпись, дата) (И.О.Фамилия)

_____ Мишин Ф.Р.
(Подпись, дата) (И.О.Фамилия)

Москва, 2020 г.

Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

УТВЕРЖДАЮ
Заведующий кафедрой ___ИУ7__
(Индекс)
_____ И. В. Рудаков
(И.О.Фамилия)
« _____ » _____ 2020 г.

ЗАДАНИЕ
на выполнение курсовой работы

по дисциплине _____Компьютерные сети_____
Студенты группы ИУ7-75Б
_____Аминов Тимур Саидович,_____
_____Мишин Филипп Рафиг оглы_____
(Фамилия, имя, отчество)
Тема курсового проекта Многопользовательское игровое приложение жанра платформер_____
Направленность КП (учебный, исследовательский, практический, производственный, др.)
_____Учебный_____
Источник тематики (кафедра, предприятие, НИР) _____Кафедра_____
График выполнения проекта: 25% к _4_ нед., 50% к _7_ нед., 75% к _11_ нед., 100% к _14_ нед.
Задание Разработать клиент-серверное приложение с передачей данных через собственный протокол прикладного уровня. Обеспечить многопользовательский доступ к приложению. Реализовать протокол для передачи информации об игровой сессии на сервер. Реализовать клиент, подготавливающий данные для отсылки на сервер с помощью протокола. Реализовать сервер, принимающий данные через протокол от одного игрока и отсылающий информацию остальным членам игры.
Оформление курсового проекта:
Расчетно-пояснительная записка на _20-30_ листах формата А4.
Перечень графического (иллюстративного) материала (чертежи, плакаты, слайды и т.п.)
На защиту проекта должна быть представлена презентация, состоящая из 10-20 слайдов. На слайдах должны быть отражены: постановка задачи, использованные методы и алгоритмы, расчетные соотношения, структура комплекса программ, диаграмма классов, интерфейс, характеристики разработанного ПО._____
Дата выдачи задания « » _____ 2020 г.

Руководитель курсового проекта _____ Рогозин Н.О.
(Подпись, дата) (И.О.Фамилия)
Студенты _____ Аминов. Т.С.
(Подпись, дата) (И.О.Фамилия)
_____ Мишин Ф.Р.
(Подпись, дата) (И.О.Фамилия)

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В настоящее время невозможно представить жизнь без Интернета, каждый день миллионы людей отправляют невообразимые объемы информации друг другу. Интернет позволяет людям коммуницировать не выходя из дома: общаться по аудио и видеозвонкам, передавать друг другу сообщения, вместе играть в компьютерные игры, смотреть сериалы и фильмы.

В соответствии с этим реализация игровых приложений через Интернет охватит большую часть аудитории. Для того, чтобы реализовать такое приложение необходимо реализовать клиент-серверное приложение с использованием прикладного протокола.

Основные задачи, которые требуются реализовать в рамках курсового проекта:

1. изучение задач существующих прикладных протоколов в данной предметной области;
2. реализация клиента - игрового приложения;
3. реализация сервера, поддерживающий “общение” между клиентами;
4. реализация прикладного протокола [1].

1. Аналитический раздел

В данном разделе производится постановка задачи, рассматривается предметная область поставленной задачи, проводится анализ протоколов транспортного и прикладного уровней, рассматриваются методы обмена данными между клиентом и сервером.

1.1 Постановка задачи

В соответствии с техническим заданием на курсовую работу требуется разработать клиент-серверное приложение - многопользовательскую игру в жанре платформер. Клиент подготавливает данные для отсылки на сервер через собственный протокол. Сервер принимает данные через данный протокол и отсылает информацию остальным участникам игры.

1.2 Анализ предметной области

Многопользовательская игра — режим компьютерной игры, во время которого играет больше одного человека. Первая многопользовательская игра появилась в 1958 г. для аналоговой ЭВМ. В настоящий же момент времени многопользовательские онлайн-игры пользуются огромной популярностью, и с каждым годом игроком становится все больше.

Игры по сети разделяются следующим образом:

1. через последовательные или параллельные порты;
2. через модем;
3. через локальную сеть и интернет, по протоколам IPX или TCP/IP. (3D-шутеры, стратегии в реальном времени и т.д.);
4. онлайн-игры:
    a. через собственный клиент: Ultima Online, Аллоды Онлайн;
    b. браузерные игры: Tale, Бойцовский клуб, Magic, Livegames;
    c. через электронную почту;
    d. через специальный веб-сайт: Мафия;

е. через IRC (например, викторины);

5. через Bluetooth (игры для мобильных телефонов).

По принципу организации связи между компьютерами сетевые игры делятся следующим образом:

1. «равный с равным» (peer-to-peer). В этом режиме связи нет четко выделенного главного компьютера; от каждого игрока информация передается всем остальным компьютерам. Каждый из компьютеров имеет достоверную информацию об игровом мире. Один из компьютеров обычно является ведущим, его роль ограничивается заданием темпа игры и управлением игрой (смена уровня, изменение настроек игры). Если ведущий выходит, роль ведущего может взять на себя любой другой компьютер;
2. звездообразная связь. Архитектура напоминает «равный с равным», однако вся связь ведётся через один центральный компьютер. Является переходным между «равный с равным» и «клиент-сервер»;
3. клиент-сервер. Один из компьютеров (сервер) содержит полную и достоверную информацию об игровом мире. Остальным компьютерам (клиентам) передаётся лишь та доля информации, которая позволяет вести игру и адекватно отображать игровой мир;
4. многосерверная модель. Многосерверная модель разработана для того, чтобы уменьшить проблему задержек, присущую обычной клиент-серверной модели. В этом режиме каждый компьютер является и клиентом, и сервером.

1.3 Протоколы транспортного уровня

На транспортном уровне стека TCP/IP используются два основных протокола: TCP и UDP.

1.3.1 Протокол TCP

Протокол TCP (Transmission Control Protocol) обеспечивает надёжную отправку сегментов. Под надёжной доставкой подразумевается автоматическая повторная пересылка недошедших сегментов. Каждый сегмент маркируется при помощи специального поля — порядкового номера. После отправки некоторого количества сегментов, TCP на отправляющем узле ожидает подтверждения от получающего, в котором указывается порядковый номер следующего сегмента, который адресат желает получить. В случае, если такое подтверждение не получено, отправка автоматически повторяется. После некоторого количества неудачных попыток, TCP считает, что адресат не доступен, и сессия разрывается.

Каждый сегмент данных обрабатывается индивидуально. То есть, как минимум, он будет запакован в индивидуальный пакет. Пакеты идут по сети и промежуточные маршрутизаторы в общем случае уже ничего не знают о том, что запаковано в эти пакеты. Часто пакеты с целью балансировки нагрузки могут идти по сети разными путями, через разные промежуточные устройства, с разной скоростью. Таким образом получатель, декапсулировав их, может получить сегменты не в том порядке, в котором они отправлялись. TCP автоматически пересоберёт их в нужном порядке используя всё то же поле порядковых номеров и передаст после склейки на уровень приложений.

Перед началом передачи полезных данных, TCP позволяет убедиться в том, что получатель существует, слушает нужный отправителю порт и готов принимать данные. Для этого устанавливается сессия при помощи механизма тройного рукопожатия. Далее, в рамках сессии передаются нужные данные. После завершения передачи сессия закрывается, тем самым получатель извещается о том, что данных больше не будет, а отправитель извещается о том, что получатель извещён.

Контроль за скоростью передачи позволяет корректировать скорость отправки данных в зависимости от возможностей получателя. Например, если быстрый сервер отправляет данные медленному телефону, то сервер будет

передавать данные с допустимой для телефона скоростью.

Благодаря механизму скользящего окна, TCP может работать с сетями разной надёжности. Механизм плавающего окна позволяет менять количество пересылаемых байтов, на которые надо получать подтверждение от адресата. Чем больше размер окна, тем больший объём информации будет передан до получения подтверждения. Для надёжных сетей подтверждения можно присылать редко, чтобы не добавлять трафика, поэтому размер окна в таких сетях автоматически увеличивается. Если же TCP видит, что данные теряются, размер окна автоматически уменьшается. Это связано с тем, что если было передано, например, 3 килобайта информации и не получено подтверждения, то неизвестно, какая конкретно часть из них не дошла и приходится пересылать все 3 килобайта заново. Таким образом, для ненадёжных сетей, размер окна должен быть минимальным. Механизм скользящего окна позволяет TCP постоянно менять размер окна — увеличивать его пока всё нормально и уменьшать, когда сегменты не доходят. Таким образом, в любой момент времени размер окна будет более или менее адекватен состоянию сети.

1.3.2 Протокол UDP

Протокол UDP (User datagram protocol) не обеспечивает надежную доставку сообщений и установку соединения. Основное назначение UDP — это максимально быстрая доставка, то есть UDP — это наиболее тонкая возможная прослойка между сетевым уровнем и уровнем приложений.

Протокол UDP может сегментировать данные, полученные с уровня приложений и адресовать работающие приложения при помощи портов.

Никаких сессий, плавающего размера окна, упорядочивания датаграмм в UDP нет. Приложениям, использующим UDP требуется быстрая доставка данных.

1.3.3 Выводы

В результате проведенного анализа протоколов транспортного уровня было решено выбрать TCP, т.к. он обеспечивает надежную доставку сегментов и упорядочивание сегментов при получении, что является приоритетными задачами при выполнении курсовой работы.

1.4 Протоколы прикладного уровня

Протокол прикладного уровня — протокол верхнего (7-го) уровня сетевой модели OSI, обеспечивает взаимодействие сети и пользователя. Уровень разрешает приложениям пользователя иметь доступ к сетевым службам, таким, как обработчик запросов к базам данных, доступ к файлам, пересылке электронной почты. Также отвечает за передачу служебной информации, предоставляет приложениям информацию об ошибках и формирует запросы к уровню представления. Пример: HTTP, POP3, SMTP.

В соответствии с заданием на курсовую работу необходимо разработать собственный протокол прикладного уровня, который обеспечит представление игровых данных удобном для отправления виде, а также будет правильно обрабатывать полученные от сервера данные.

1.5 Сокеты Беркли

Сокеты Беркли [2] обеспечивают возможность обмена данными между клиентом и сервером. Все современные операционные системы имеют ту или иную реализацию интерфейса сокетов Беркли, так как это стало стандартным интерфейсом для подключения к сети Интернет.

Создание простейшего TCP-сервера состоит из следующих шагов:

1. создание TCP-сокетов вызовом функции socket();
2. привязывание сокета к прослушиваемому порту вызовом функции bind();
3. подготовка сокета к прослушиванию на предмет соединений (создание прослушиваемого сокета) при помощи вызова listen();

4. принятие входящих соединений через вызов accept(). Это блокирует сокет до получения входящего соединения, после чего возвращает дескриптор сокета для принятого соединения. Первоначальный дескриптор остаётся прослушиваемым дескриптором, а accept() может быть вызван вновь для этого сокета в любое время (пока он открыт);
5. соединение с удаленным хостом, которое может быть создано при помощи send() и recv() или write() и read();
6. итоговое закрытие каждого открытого сокета, который больше не нужен, происходит при помощи close().

Создание TCP-клиента происходит следующим образом:

1. создание TCP-сокета вызовом socket();
2. соединение с сервером при помощи connect(), передача структуры sockaddr_in с sin_family с указанными PF_INET или PF_INET6, sin_port для указания порта прослушивания (в байтовом порядке), и sin_addr для указания IPv4 или IPv6 адреса прослушиваемого сервера (также в байтовом порядке);
3. взаимодействие с сервером при помощи send() и recv() или write() и read();
4. завершение соединения и сброс информации при вызове close().

1.6 Выводы

В данном разделе был произведен выбор протокола транспортного уровня, рассмотрены задачи прикладных протоколов, используемых в данной предметной области, и методы обмена данными между клиентом и сервером.

2. Конструкторский раздел

В данном разделе рассмотрены общие требования к созданию ПО, диаграмма классов, схема работы сервера.

Программный комплекс в данной курсовой работе состоит из 3-х частей:

1. сервер - осуществляет инициализацию игровых сессий, пересылку сообщений от клиента-источника (игрок, чья очередь ходить) всем клиентам-слушателям;
2. клиент - игровое приложение, которое отображает графическую информацию, регистрирует действия пользователя, а также отсылает действия пользователя остальным игрокам;
3. протокол прикладного уровня - осуществляет перевод данных в json-формат, позволяющий просто извлекать данные из битовых строк.

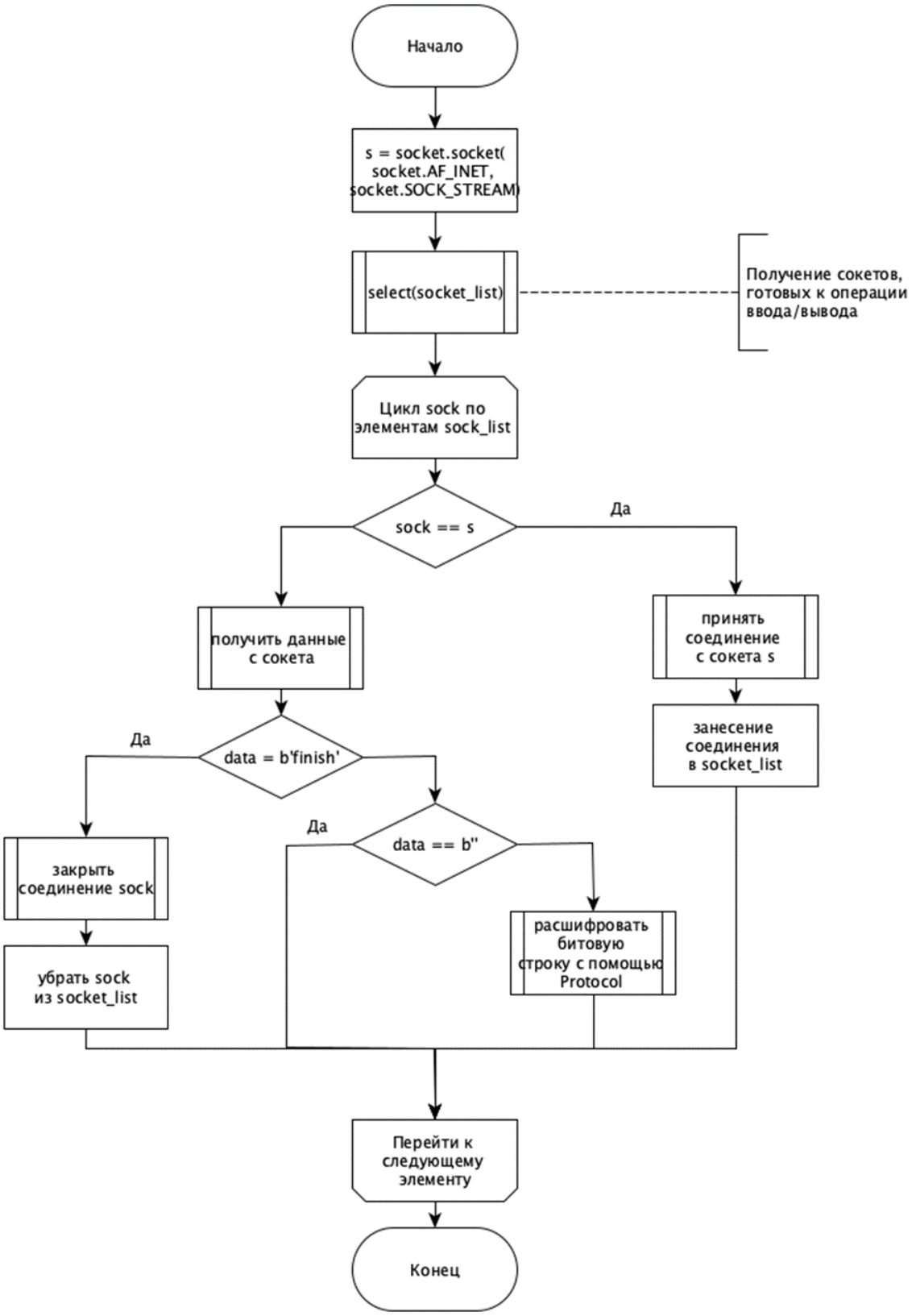
2.1 Сервер

Программа состоит из следующих основных процедур:

- инициализация;
- добавление игрока в сессию;
- рассылка сообщений между игроками внутри сессии;
- обработка сообщения от игрока.

На рисунке 2.1 представлена схема алгоритма работы сервера.

Рисунок 2.1. Схема алгоритма работы сервера.



2.2 Клиент

Программа состоит из следующих основных процедур:

- инициализация (включает подключение игрока к серверу и ожидания сообщения о запуске игры);
- передвижение персонажей посредством игроков;
- передвижение обоих игроков;
- отображение основного меню;
- отображение локаций и объектов.

2.3 Протокол прикладного уровня

Протокол конвертирует входные данные в битовую строку для передачи данных по сети. Также протокол позволяет извлекать данные из битовой строки.

2.4 Структура передаваемых сообщений

Сообщения в игре передаются с помощью типа словарь. В любом сообщении есть поле “type”, определяющее тип передаваемого сообщения. В зависимости от значения этого поля остальная часть сообщения может меняться.

При передаче сообщений между клиентами и сервером используются следующие типы сообщений:

1. connect (клиент) - сообщение о намерении клиента подключиться к игровой сессии. Дополнительные параметры в данном случае не передаются;
2. connect (server) - ответ на запрос соединения от клиента. Дополнительно передаются параметры player_id, которые определяют номер игрока в сессии и номер сессии на сервере соответственно. Во всех последующих от клиента сообщениях эти

параметры будут передаваться для более удобного определения отправителя сообщения;

3. start – загрузка карты.
4. position- сообщение хранит информацию о текущем расположение персонажа и текущей анимации.

2.5 Выводы

В данном разделе рассмотрены общие требования к созданию ПО, диаграммы классов и схема работы сервера.

3. Технологический раздел

В данном разделе рассмотрен выбор средств программной реализации, рассмотрена реализация серверной и клиентской части, описан интерфейс программы.

3.1 Выбор стека технологий

Операционной системой для сервера была выбрана ОС Linux, а конкретно дистрибутив Ubuntu из-за бесплатности и популярности в таком типе задач.

Для клиентской части был выбран язык python [3], так как изначально приложение было написано на нем. Для серверной части также был выбран язык python из-за большого количества библиотек, в том числе API Linux.

Для организации командной разработки была выбрана система контроля версий git из-за нативной поддержки Linux, был создан репозиторий на веб-сервисе github.

Для взаимодействия между клиентом и сервером [4] были использованы сокеты Беркли, так как это распространенный способ передачи информации по сети.

3.2 Реализация серверной части

При разработке сервера необходимо реализовать организацию игровых сессий и их контроль.

Сервер реализован в однопоточном виде - в основном цикле прослушиваются все сокеты, и далее последовательно идет обработка пришедших на них данных.

Полный листинг сервера приведен в приложении А.

3.2.3 Развертывание сервера

Для корректного взаимодействия между клиентской и серверной программой была произведена настройка серверного ПК и маршрутизатора:

1. домашней сети был присвоен глобальный статический ipv4-адрес;
2. в рамках локальной сети серверному ПК (в данном случае в его роли выступил рабочий ноутбук) был присвоен статический ipv4-адрес 192.168.0.104;
3. маршрутизатор [4] был настроен на прослушивание порта 9090 и передачу пакетов на соответствующий порт серверного пк (в данном случае порт был тоже 9090);
4. на серверном пк была запущена программа-сервер, которая с помощью сокета прослушивает порт 9090 и принимает сообщения от клиентов;
5. программа-клиент была настроена так, чтобы связываться с сокетом на сервере по адресу сети сервера и порту 9090.

3.3 Реализация протокола прикладного уровня

В данном приложении перед протоколом стоит задача конвертации данных из словаря python в байтовую строку, пригодную для передачи по сети, и наоборот - перевод сетевого сообщения в словарь.

В качестве промежуточного этапа между словарем и байтовой строкой был использован JSON, для более простого приведения типов, так как без JSON понадобилось бы вручную определять типы полей.

Протокол реализован как статический класс со следующими методами:

@staticmethod

def getByteStrFromData(data: dict) -> bytes:

Сериализует словарь в байтовую строку.

Параметры:

- data: словарь с данными для отправки сообщения по сети

@staticmethod

def getDataFromByteStr(byte_str: bytes) -> list:

Параметры:

- byte_str: байтовая строка, полученная по сети

Полный код прикладного протокола приведен в приложении Б.

3.4 Реализация клиентской части

В данном случае игровая логика реализована полностью в клиентской части за исключением организации игровых сессий. Сетевой код клиентской части игры реализован в классе NetWork. Взаимодействие с другими классами приложения реализовано с помощью сигналов и слотов QT (которые являются реализацией паттерна подписчик-издатель), то есть по событийной модели. По пришествию сообщения от сервера возбуждается событие, соответствующее типу сообщения.

3.5 Инструкция по запуску программного обеспечения

Требования:

1. интерпретатор python 3;
2. установленная (с помощью команды pip или исходников) библиотека cocos2d [5];
3. установленная (с помощью команды pip или исходников) библиотека socket.

Для игры в корневой папке приложения запустите с помощью python3 файл main.py

В меню выбираете “Новая игра” и сервер подбирает игровую сессию.

3.6 Примеры работы программы

На рисунках 3.1 и 3.2 приведены скриншоты программы

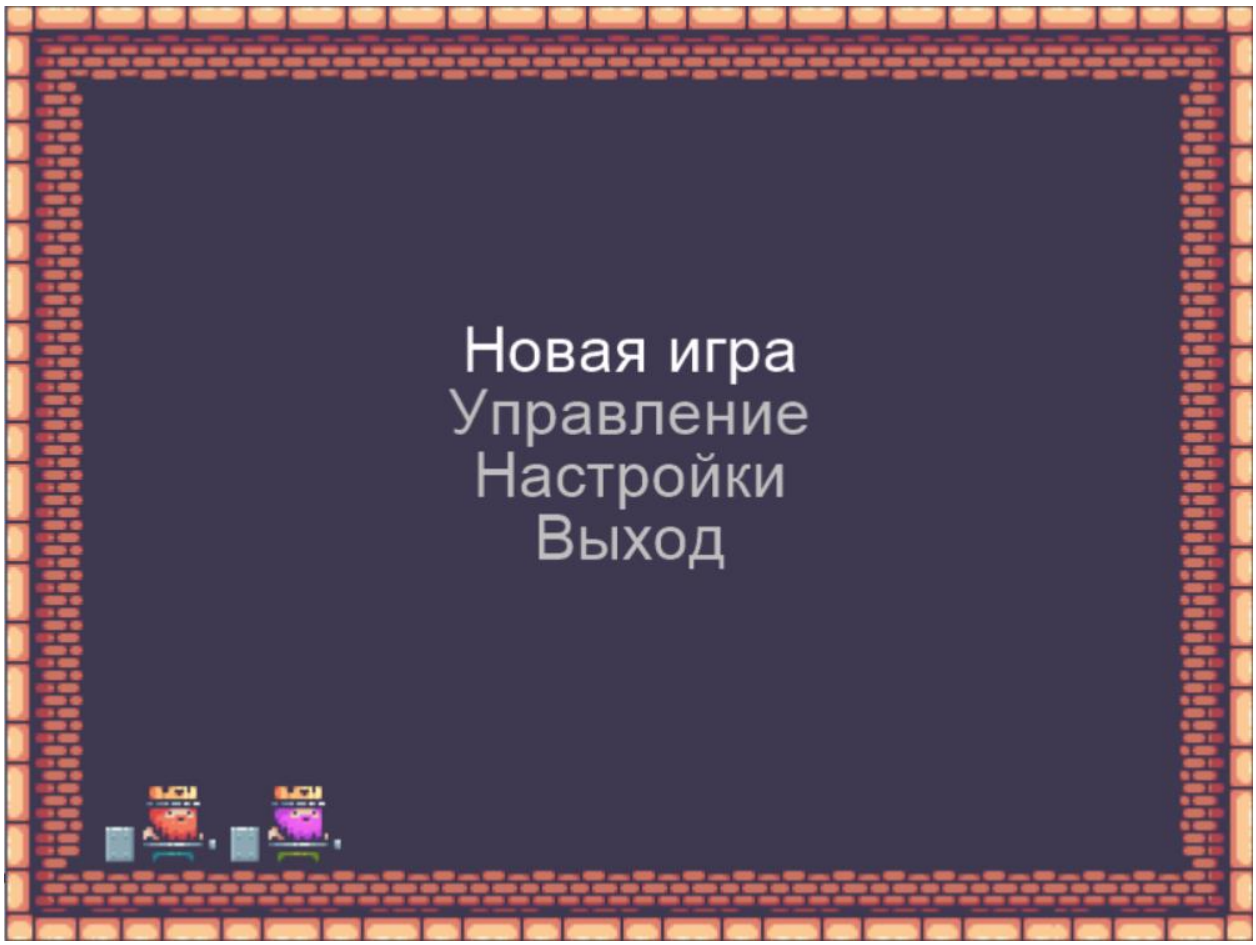


Рис. 3.2. Главное меню игры

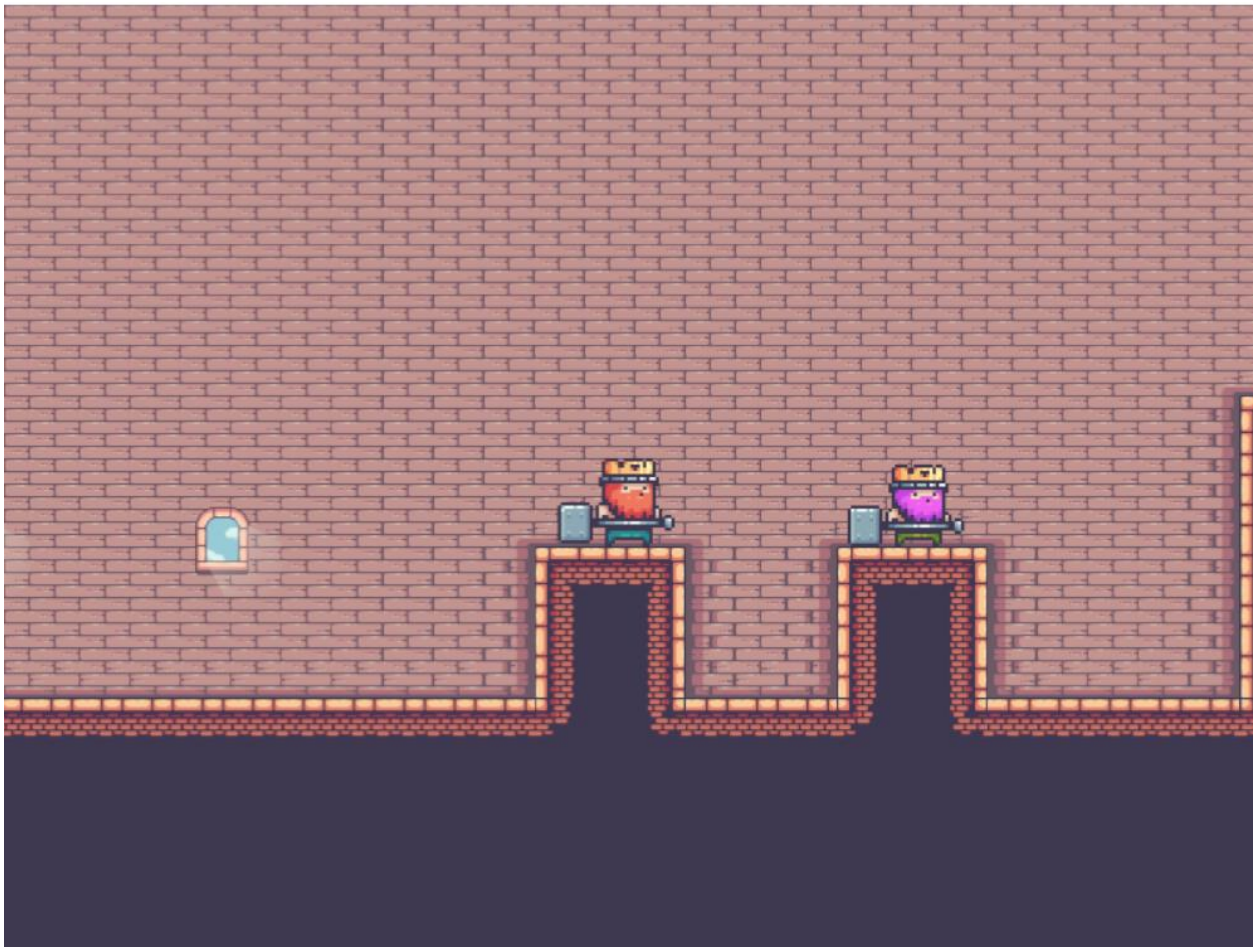


Рис. 3.2. Скриншот игрового процесса

3.6 Выводы

В данном разделе были рассмотрены: технологический стек, краткая документация по игре (описаны сетевые функции и методы и в некоторых случаях приведен код), а также инструкция по запуску и примеры работы программы.

ЗАКЛЮЧЕНИЕ

В ходе работы был формализован необходимый функционал, проведен анализ существующих прикладных протоколов, спроектировано и реализовано клиент-серверное приложение, а также прикладной протокол.

В дальнейшем в качестве усовершенствования приложения будет добавлен многопоточный сервер для обработки сообщений от клиентов параллельно.

Список использованной литературы

1. В. Олифер, Н. Олифер. Компьютерные сети. Принципы, технологии, протоколы. – Издательский дом "Москва", 2017.
2. Джеймс Куроуз, Кит Росс. Компьютерные сети. Нисходящий подход. – Издательский дом "Москва", 2016.
3. Документация по языку программирования Python [Электронный ресурс]. – Режим доступа: https://docs.python.org/, свободный – (дата обращения 24.10.20).
4. Таненбаум Э. С. Компьютерные сети:[пер. с англ.]. – Издательский дом" Питер", 2012.
5. Документация по библиотеке cocos2d [Электронный ресурс]. – Режим доступа: https://docs.cocos.com/creator/manual/en/, свободный – (дата обращения 24.10.20).

Приложение А. Исходный код сервера

Листинг А.1. Сервер, обрабатывающий клиентские соединения.

```
import select, socket, sys
from protocol import MyProtocol

server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server.setblocking(False)
server.bind(('', 9090))
server.listen(4)
socket_list = [server]
clients = []
positions = []
animations = []
max_clients = 2

while socket_list:
    sockets_to_read, _, _ = select.select(socket_list, [], [])
    for s in sockets_to_read:
        if s is server:
            print('here 0')
            connection, client_address = s.accept()
            connection.setblocking(0)
            socket_list.append(connection)
        else:
            if s not in clients:
                clients.append(s)
                positions.append([0, 0])
                animations.append('none')
            data = s.recv(10000)
            if data:
                mes = MyProtocol.getDataFromByteStr(data)
                if mes['type'] == 'connect':
                    player_id = len(clients)
                    answer = {'type': 'connected', 'player_id': player_id}
                    answer_bit = MyProtocol.getByteStrFromData(answer)
                    s.send(answer_bit)
                elif mes['type'] == 'ask':
                    if mes['question'] == 'players_connected':
                        answer = {'type': 'ask', 'players_connected': len(clients)}
                        answer_bit = MyProtocol.getByteStrFromData(answer)
                        s.send(answer_bit)
                    if mes['question'] == 'game_ready':
                        answer = {'type': 'ask', 'question': 'game_ready'}
```

```
            if len(clients) == max_clients:
                answer['answer'] = 'yes'
            else:
                answer['answer'] = 'no'
            answer_bit = MyProtocol.getByteStrFromData(answer)
            s.send(answer_bit)
    elif mes['type'] == 'start':
        answer = {'type': 'start'}
        answer_bit = MyProtocol.getByteStrFromData(answer)
        for client in clients:
            client.send(answer_bit)
    elif mes['type'] == 'position':
        positions[mes['player_id'] - 1] = mes['pos']
        animations[mes['player_id'] - 1] = mes['anim']

        answer = {'type': 'position'}
        for i in range(len(clients)):
            if i != mes['player_id'] - 1:
                answer[f'pos_{i + 1}'] = positions[i]
                answer[f'anim_{i + 1}'] = animations[i]

        answer_bit = MyProtocol.getByteStrFromData(answer)
        s.send(answer_bit)
```

Приложение Б. Исходный код прикладного протокола

Листинг Б.1. Прикладной протокол.

```
import json
from sys import getsizeof

example_dict = {'id': 1, 'session_id': 2, 'unit': 'army', 'coord_from': [1, 1], 'co
ord_to': [10, 10]}

class MyProtocol:
    __size: int = 0
    __data: dict

    __coding: str = 'utf-8'

    __bytes_start: bytes = b'START'
    __bytes_end: bytes = b'END'
    __bytes_sep: bytes = b':'

    def __init__(self, data: dict):
        self.__data = data
        self.__size = getsizeof(data)

    def __str__(self):
        return 'Object of class MYProtocol \n' + \
               'data = ' + str(self.__data) + '\n'

    def getSize(self):
        return self.__size

    def getData(self):
        return self.__data

    @staticmethod
    def getByteStrFromData(data: dict) -> bytes:
        res_str = MyProtocol.__bytes_start + \
                  MyProtocol.__bytes_sep + \
                  json.dumps(data).encode(MyProtocol.__coding) + \
                  MyProtocol.__bytes_sep + \
                  MyProtocol.__bytes_end
        return res_str

    @staticmethod
    def getDataFromByteStr(byte_str: bytes) -> dict:
```

```
            if MyProtocol.__bytes_start == byte_str[:len(MyProtocol.__bytes_start
)] and \
                    MyProtocol.__bytes_end == byte_str[-
len(MyProtocol.__bytes_end):]:

                byte_dict = byte_str[len(MyProtocol.__bytes_start) + 1:len(byte_str)
- len(MyProtocol.__bytes_end) - 1]
                res = json.loads(byte_dict)

                return res

            else:
                is_MyProtocol = False

            if not is_MyProtocol:
                print('Unknown protocol')
                return {}
```