Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное учреждение высшего образования

«Новгородский государственный университет имени Ярослава Мудрого»

Кафедра информационных технологий и систем

**Сетевая игра «Червяки.»**

Курсовая работа

по дисциплине:

«Операционные системы»

Разработал:

Студент группы 8091

Шаклеин В. В. \_\_\_\_\_\_

«\_\_».\_\_\_\_\_.2020г.

Проверил:

Ананьев В. В. \_\_\_\_\_\_

«\_\_».\_\_\_\_\_.2020г.

**Великий Новгород**

**2020**

1. ОГЛАВЛЕНИЕ

[ОГЛАВЛЕНИЕ 2](#_Toc42246550)

[ВВЕДЕНИЕ 3](#_Toc42246551)

[ГЛАВА 1. РАЗРАБОТКА ПРОЕКТА. 4](#_Toc42246552)

[1.1 Постановка задачи. 4](#_Toc42246553)

[1.2 Выбор инструментальных средств. 4](#_Toc42246554)

[1.3 Выбор модели. 4](#_Toc42246555)

[1.4 Выбор способа решения проблемы блокирования ввода/вывода. 4](#_Toc42246556)

[1.5 Протокол взаимодействия клиента и сервера. 6](#_Toc42246557)

[1.6 Выбор протокола транспортного уровня OSI-модели. 6](#_Toc42246558)

[1.7 Алгоритм решения задачи. 7](#_Toc42246559)

[Со стороны клиента: 8](#_Toc42246560)

[Со стороны сервера: 9](#_Toc42246561)

[1.8 Вывод по первой главе. 10](#_Toc42246562)

[ГЛАВА 2. РЕАЛИЗАЦИЯ ПРОЕКТА. 11](#_Toc42246563)

[2.1 Структура клиента и сервера. 11](#_Toc42246564)

[2.2 Программный интерфейс сервера. 11](#_Toc42246565)

[Описание работы главного потока: 11](#_Toc42246566)

[Описание работы потока игрока: 12](#_Toc42246567)

[Описание работы потока бота: 12](#_Toc42246568)

[2.3 Описание программной реализации клиента 12](#_Toc42246569)

[2.4 Системные вызовы. 13](#_Toc42246570)

[2.5 Пример работы приложения. 14](#_Toc42246571)

[2.6 Результат работы. 14](#_Toc42246572)

[ЗАКЛЮЧЕНИЕ 16](#_Toc42246573)

[СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ. 16](#_Toc42246574)

1. ВВЕДЕНИЕ

Целью данной работы является создание пошаговой игры многопользовательской игры на подобии игры “Змейка”.

Особенность проектируемой игры заключается в том, что она должна работать под операционной системой Linux, а также быть пошаговой и многопользовательской в отличии от оригинала.

Основной функционал игры и правила игры следующие:

В одной игре может принимать участие от 1 до 8 человек (по факту максимальное количество игроков может быть больше 8, но тестирование при таких значениях не проводилось). Перед началом игры игроки подключаются к серверу и могут заявить о своей готовности или неготовности к началу игры. После того как все подключившиеся игроки заявят о своей готовности игре присоединяются компьютерные игроки. Затем начинается игра. Каждый ход игроки по очереди выбирают, в каком направлении хотят сдвинуть голову своего червя. Когда все игроки сделали выбор происходит сдвижения голов всех червей в выбранном направлении. Если в результате перемещения голова червя занимает клетку, в которой была цифра, происходит увеличении его счёта и длинны на значение данной цифры. Если голова червя пытается занять непустую клетку, в которой находится что-либо кроме еды, червь удаляется с поля. Увеличение длинны червя происходит путём не удаления последней клетки хвоста червя, если его длинна на поле не превышает значения его длинны. После того как все черви исчезнут с поля, всем игрокам показывается таблица результатов игры, а затем через некоторое время все игроки перемещаются обратно в меню ожидания.

Перечень основных функций, предъявляемых к разрабатываемой сетевой игре:

1. Подключение игрока по IP-адресу и порту сервера.
2. Простой и понятный графический интерфейс для игроков.
3. ГЛАВА 1. РАЗРАБОТКА ПРОЕКТА.

1.1 Постановка задачи.

Задачей данной курсовой работы является разработка сетевой пошаговой игры «Змейка», которая рассчитана на несколько игроков. Игра должна работать под управлением операционной системы семейства Unix/Linux. Сетевая часть иры должна быть выполнена на языке С/C++ с помощью сокетов, предоставляемых системной библиотекой <sys/socket.h>.

1.2 Выбор инструментальных средств.

Основным дистрибутивом, использовавшимся при разработке, был Ubuntu 18.04.4 LTS, поскольку он работает в рамках WSL (Windows Subsystem for Linux).

Сетевая часть программы будет реализована с помощью библиотек сокетов Беркли на C++. Протокол передачи данных: TCP.

Для реализации графического интерфейса было решено использовать SDL2, свободную, кроссплатформенную мультимедийную библиотеку, реализующую единый программный интерфейс к графической подсистеме, звуковым устройствам и средствам ввода для широкого спектра платформ.

1.3 Выбор модели.

В сетевой игре, в которой может принять участие несколько игроков, необходима определенная модель, которая обеспечит их взаимодействие в игре. Для этого была выбрана модель взаимодействия компьютеров и программ в сети клиент-сервер. Основная часть игры, заключающаяся в изменении состояния игрового поля, получении команд от игроков, подсчёте поля и переводу игры в различные состояния (ожидания, игры, таблицы результатов) должна работать на одном компьютере, называемом сервер.

1.4 Выбор способа решения проблемы блокирования ввода/вывода.

Обычно операция приема данных из сокета выполняется в блокирующем режиме. Если на сокете нет данных для чтения, то поток, вызвавший операцию чтения, блокируется до их появления. Такое поведение представляет проблему для приложения, которое должно иметь возможность проверять наличие входящих данных на сокете без торможения графического интерфейса. Также проблемы возникают с реализацией игрового сервера, который должен работать сразу с несколькими клиентами. Если сервер, поддерживающий TCP-соединение с несколькими клиентами, вызовет функцию read с одним из сокетов, чтобы проверить получение данных от одного клиента, поток выполнения остановится до тех пор, пока этот клиент не отправит серверу какие-либо данные. Пока сервер будет ждать, он не сможет получать данные от других клиентов, принимать новые соединения и выполнять игровые операции. Для решения этой проблемы есть три универсальных способа:

1. Многопоточность. Можно производить блокирующие вызовы в отдельных потоках выполнения. Тогда для решения поставленной задачи серверу нужно запускать по одному потоку для приема данных от каждого клиента, один поток для приема данных и один для выполнения игровых операции. Это вполне работоспособное решение, но оно требует создания потока для каждого клиента и поэтому плохо масштабируется с увеличением их числа. Кроме того, так как потоки обслуживания клиентов будут выполняться параллельно, то необходим механизм, обеспечивающий безопасность выполнения игровых операций. Наконец, если главный поток сервера попытается отправить данные в сокет в то же самое время, когда другой поток будет принимать данные из этого сокета, он окажется заблокированным и не сможет дальше эмулировать игру. Эти проблемы можно решить, но есть более простые варианты.

2. Неблокирующий ввод/вывод. По умолчанию сокеты действуют в блокирующем режиме. Но также сокеты поддерживают неблокирующий режим работы. Когда сокету, действующему в неблокирующем режиме, предлагается выполнить операцию, которая в ином случае заблокировалась бы, он сразу возвращает управление с результатом -1. Переведя сокеты в неблокирующий режим, игра может выполнять проверку появления новых данных в каждом кадре. Если данные получены, то игра их обработает, если нет, то игра перейдет к выполнению других игровых операций. Но такой способ перестает быть эффективным, когда число опрашиваемых сокетов растет, поскольку процессорное время тратится впустую на многократные и безрезультатные опросы.

3. Функция select. В качестве альтернативы библиотека сокетов дает возможность проверить сразу множество сокетов и выполнить действие, как только один из них будет готов. Для этого используется функция select, которая позволяет отслеживать состояние нескольких файловых дескрипторов одновременно. С помощью этой функции сервер может получить из множества всех используемых сокетов получить множество сокетов с доступными для чтения данными. Далее сервер может осуществить перебор этих сокетов. Если сокет предназначен для приема запросов на соединение, то сервер должен принять это соединение и добавить новый сокет в множество используемых сокетов. Если очередной сокет — это обычный сокет, то сервер должен определить к какому клиенту он относится и обработать данные с него.

Клиентскую часть своей разрабатываемой игры я решил организовать следующим образом. Получение, отправка данных происходят в том же потоке, что и отрисовка кадра. Подобное решение не является оптимальным, но подходит в данном случае поскольку библиотека SDL2 сама создаёт отдельный поток, который принимает данные с клавиатуры, а управление мышью в игре отсутствует.

Серверная часть моей программы идёт отдельно от клиентской и создаёт отдельный поток для каждого игрока, так как этого требует Задание 5 из документа “Курсовые работы ОС.doc” на которое я ориентировался при выполнении данной работы. Так же во время стадии ожидания игроков используется сокет в неблокирующем режиме, для того чтобы проверять не только новые подключения, но также и то, не пора ли начать игру.

1.5 Протокол взаимодействия клиента и сервера.

Так как игра является пошаговой, а все возможные стадии (меню ожидания, игра, таблица результатов) всегда следуют друг за другом в строго определенном порядке, клиент знает какие данные он получит далее, в данных может содержатся информация о том что стадия изменилась и следующие пакеты будут уже другого типа.

1.6 Выбор протокола транспортного уровня OSI-модели.

Для разработки игры необходимо выбрать какой протокол из TCP и UDP выбрать.

TCP-протокол обеспечивает надежность передаваемых данных. Он посылает подтверждения о приеме данных и в случае, если данные не получены или искажены запрашивает повторную передачу. Также этот протокол обеспечивает правильный порядок принятых данных. Это сложный, требующий больших затрат времени протокол, но он берет на себя заботу о гарантированной доставке пакетов, избавляя программиста от необходимости включать эту функциональную возможность в прикладной протокол.

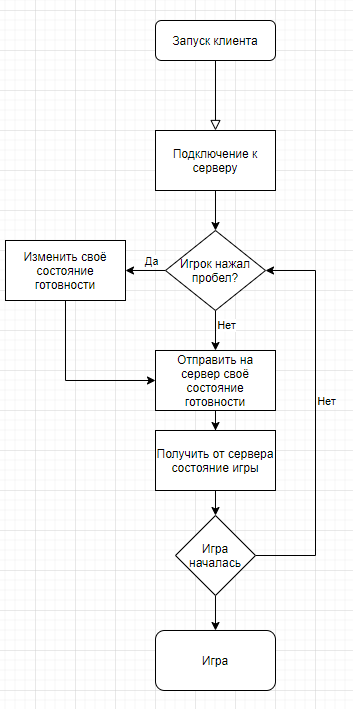
UDP-протокол в отличии от TCP — очень быстрый протокол, поскольку в нем определен самый минимальный механизм, необходимый для передачи данных. В нем есть свои недостатки. Сообщения поступают в любом порядке, и то, которое отправлено первым, может быть получено последним. Доставка сообщений в UDP не гарантируется. Сообщения могут потеряться, или могут быть получены две копии одного и того же сообщения.

Для своей курсовой работы я решил использовать TCP-протокол, поскольку придуманный мной протокол взаимодействия клиента и сервера основывается на том, что сообщения приходят с строго определённом порядке, а основные недостатки TCP относительно UDP нивелируются, тем что игра является пошаговой, а для каждого игрока в любом случае должен создаваться отдельный поток.

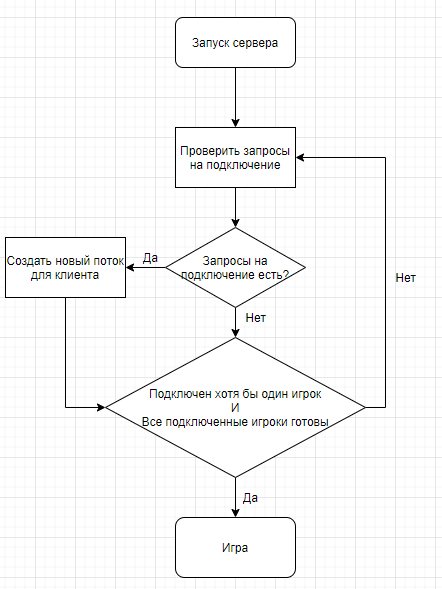
1.7 Алгоритм решения задачи.

Алгоритм начала игры выглядит следующим образом:

Со стороны клиента:



Со стороны сервера:



Таким образом происходит приложение проходит стадию меню ожидания. После этого происходит переход непосредственно на стадию игры. На этой стадии всем игрокам отправляются изменения на игровом поле (в первый раз поле целиком), а также информация о то, чей сейчас ход. После того как первый игрок выберет направление хода, очередь передаётся второму игроку и так далее. После того как все игроки выбрали направление хода происходит перемещение червей. После того как на поле не останется ни одного червя, игра переходит в стадию показа таблицы результатов, а затем, спустя некоторое время обратно в стадию меню ожидания игроков. Игра будет продолжаться, пока сервер не будет вручную закрыть.

1.8 Вывод по первой главе.

Таким образом, для реализации игры «Червяки» основной ОС выбрана Ubuntu, языками программирования процедурной части C и С++, а библиотекой для работы с графикой и пользовательским вводом на стороне клиента SDL2.

Была выбрана модель взаимодействия компьютеров и программ в сети клиент-сервер и параллельный метод обработки клиентов.

Для проблемы блокирования ввода/вывода было решено использовать создания нового потока для каждого клиента.

Был составлен протокол взаимодействия клиентов и сервера.

Используемым протоколом транспортного уровня OSI-модели был выбран TCP.

1. ГЛАВА 2. РЕАЛИЗАЦИЯ ПРОЕКТА.

2.1 Структура клиента и сервера.

Так как основным языком разработки был выбран C, классы как таковые в проекте отсутствуют.

Функционал сервера разнесён по нескольким файлам, а именно:

sem.c – вспомогательные функции для работы с семафорами

server.c – выделяет память, в которой будет храниться состояние игры, создаёт сокет для подключения новых игроков и управляет переключением стадий игры.

serverGameLogic.c – содержит игровую логику, отвечает за изменение игрового состояния

wormLogic.c – содержит инструкции для работы с уже подключившимися игроками, отправляет им состояние сервера, принимает от них ввод, следит за их отключением.

bot.c – содержит логику для компьютерных игроков

Клиент состоит из одного файла client.c

Так же клиент и сервер используют файл shared.h для упрощения изменения структур, передаваемых по сети.

2.2 Программный интерфейс сервера.

Описание работы главного потока:

При запуске сервера создаётся сокет на порте, указанном в параметрах командной строки, который затем переводится в неблокирующее состояние. Также выделяется память для хранения игрового состояния. Затем идёт работа с функциями из файла serverGameLogic.c.

Сначала в функции resetState происходит сброс игрового состояния до стандартного. Затем вызывается функция acceptPlayers, которая ожидает новые подключения на сокете и в случае их появления находит для них слот и создаёт новый поток, а также подсчитывает количество подключенных и готовых игроков. Если хотя бы один игрок подключен и все подключенные игроки готовы, происходит выход из функции acceptPlayers. Затем происходит вызов функции addBots, для незанятых игровых слотов она создаёт потоки ботов. После этого вызывается функция setupGame, которая подготавливает игру к запуску. Затем вызовом функции playGame запускается непосредственно игра. Во время игры главный поток удаляет с поля отключившихся игроков, добавляет на поле новую еду, выбирает нового активного игрока, если предыдущий сделал ход, а так если все игроки сделали ход сдвигает всех червей и удаляет лишние клетки хвоста, а так же врезавшихся во что-либо червей. Когда на поле не остаётся ни одного червя происходит выход из функции playGame. После этого главный поток ожидает время, отведённое на показ таблицы результатов.

Описание работы потока игрока:

После создания происходит установка информации о собственном подключении в общем состоянии игры. Затем пока игра находится в состоянии ожидания игроков, от игрока принимается его состояние готовности, а отправляется количество готовых игроков и игроков в общем. Затем, когда стадия игры сменилась на непосредственно игровую, игроку единожды отправляется размер карты. После чего отправляется информация о том менялось ли игровое поле, в случае если да, то для каждого столбца изменился ли он и если да, то в каком состоянии он теперь находится. Затем игроку передаётся информация о его счёте и о том. какой червь сейчас должен делать ход. Наконец, от клиента принимается информация о том, в каком направлении и, если оно является корректным, а червь в данный момент является активным, записывается в глобальное состояние. Это повторяется, пока состояние игры не сменится на отображение таблицы результатов (то есть пока на поле есть черви). Затем, игроку единожды отправляется размер таблицы результатов и все записи в ней, после чего до того как состояние изменится на ожидание игроку отправляется текущее состояние, а от него получается байт синхронизации(этот байт не несёт полезной информации и связан с неудачным технически решением в начале разработки приложения)

Описание работы потока бота:

После создания поток бота, как и поток игрока записывает в глобальное состояние информацию о том, что он подключен и готов. После этого он дожидается старта игры. В процессе игры бот каждый ход с помощью алгоритма Ли ищет путь до ближайшей клетки с едой, если такой путь найден, то он делает ход по нему, а если нет, то движется к дальнейшей от головы свободной клетке(такое движение при не нахождении еды не является оптимальным, однако помогает создать иллюзию интеллекта бота, поскольку исключает ситуации когда он во что-то врезается, несмотря на наличие доступных ходов). После того как бот врежется или игра перейдёт в состояние отображения таблицы результатов, бот обозначает своё отключение в глобальном состоянии и его поток завершается.

2.3 Описание программной реализации клиента

После запуска клиента происходи загрузка ресурсов и инициализация SDL2. Затем клиент подключается к серверу по ip и порту указанным как параметр запуска. Игрок единожды получает свой порядковый номер. Каждый кадр проверятся ввод с клавиатуры и не хочет ли пользователь его закрыть. Затем пока игра находится в состоянии ожидания игроков клиент отправляет на сервер свой статус готовности и получает с него количество готовых игроков и игроков в общем, которые выводит на экран. Затем игра переходит в состояние игрового процесса, клиент единожды получает размер игрового поля. В этом состоянии клиент получает обновления игрового поля и игрового состояния, а отправляет направление движение, выбранное игроком. Затем, когда состояние переходит в отображение итоговой таблицы, клиент единожды получает её размер и очки всех игроков. После этого пока стадия игры не изменится игрок получает от сервера состояние игры и отправляет на него байт синхронизации.

2.4 Системные вызовы.

Для данного проекта использовались следующие системные вызовы:

|  |  |
| --- | --- |
| Системный вызов | Описание |
| socket() | Служит для создания сокета в ОС. Имеет три парметра. Первый указывает, к какому семейству протоколов относится создаваемый сокет, а второй и третий параметры определяют конкретный протокол внутри данного семейства. |
| connect() | Служит для установки логического соединения со стороны клиента. Имеет три параметра: дескриптор активного сокета, через который будет устанавливаться соединение, полный адрес сокета сервера и его длина. |
| read() и write() | Используются для чтения и записи файлов. Могут быть использованы для обмена данных, если в качестве параметров вместо дескрипторов файлов в них задаются дескрипторы сокетов. |
| bind() | Настраивает адрес для созданного сокета. Первый параметр вызова должен содержать дескриптор сокета, для которого производится настройка адреса. Второй и третий параметры задают этот адрес. |
| listen() | Прослушивает соединения на сокете. Позволяет показать готовность принимать соединения и задать лимит входящих соединений. В качестве первого параметра используется дескриптор сокета. Второй параметр определяет максимальную длину очереди входящих соединений. |
| accept() | Извлекает первый запрос на соединение из очереди ожидающих соединений, создаёт новый подключенный сокет и выделяет для сокета новый файловый дескриптор, который и возвращается. В качестве параметров принимает дескриптор слушающего сокета, указатель на структуру адреса и размер структуры адреса. |
| fcntl() | Выполняет управляющие операции над открытыми файлами. Используется с опцией O\_NONBLOCKING для включения неблокирующего режима сокета. |

2.5 Пример работы приложения.

Приведу пример работы приложения с 2 игроками и 1 ботом. Запустив игру, игроки видят меню ожидания. 1-ый игрок нажимает пробел, тем самым меняя свой статус готовности. Оба игрока видят, что теперь один игрок готов, а один нет. Второй игрок так же нажимает пробел, у обоих игроков на экране появляется игровое поле, на котором кроме них присутствует один червь управляемый компьютерным игроком, так же оба игрока видят, что их счёт равен нулю, а выбирать направление должен первый игрок. Когда первый игрок выбирает направление движения(вправо), право выбора передаётся второму игроку. Когда второй игрок также делает свой выбор(вниз), право выбора передаётся компьютерному игроку, который выбирает направление движения автоматически. После этого все черви на поле перемещаются в ранее выбранных направление, а право выбора возвращается первому игроку. Он снова делает свой выбор(вправо), затем свой выбор делает второй игрок(влево) и наконец компьютерный. Вновь происходит движение игроков. Так как в клетке, в которую попал первый игрок, его счёт и длинна увеличиваются, а второй игрок на клетке, так как на клетке, в которую он направлялся находится стена удаляется с поля. Теперь выбор направления делают только первый игрок и бот. После того как первый игрок вновь решает сдвинуться вправо последняя клетка его хвоста не исчезает, потому что после предыдущего хода его длинна равна 4-ём. Теперь первый игрок решает сдвинуться вверх. Он удаляется с поля и бот доигрывает игру в одиночестве, а после того, как и он врезается в препятствие, на экране отображается таблица результатов. Через 5 секунд игроки возвращаются обратно в меню ожидания.

2.6 Результат работы.

В ходе выполнения данной курсовой работы удалось разработать сетевую пошаговую игру на подобии игры “Змейка”, соответствующую поставленным к ней требованиям. Для обеспечения быстрой работы со множеством клиентов был использован неблокирующий режим сокетов и создание отдельного потока для каждого игрока. Сервер и клиент были реализованные как отдельные приложения. Разработанный сервер удалось запустить с 2 игроками и 6 ботами, во время тестирования он также запускался с 1 игроком и 99 ботами и показывал стабильную работу. Удалось реализовать синхронизацию игрового состояния между сервером и клиентами. Игра работает как в локальной сети, так и через интернет. Удалось реализовать отключение клиента на сервере, если с ним произошел перебой связи, и он не отвечает.

1. ЗАКЛЮЧЕНИЕ

Таким образом, результат курсовой работы соответствует требованиям к реализации. Приложение работает на операционных системах семейства Linux. Для построения серверной части приложения использован язык C и компилятор gсс, а для клиентской язык C++ и компилятор g++. Сетевое взаимодействие было построено с помощью протокола TCP. Сервер может одновременно обслуживать несколько клиентов. В целом созданная сетевая игра логически завершена.

1. СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ.
2. Карпов В. Е., Коньков К. А. «Основы операционных систем. Курс лекций» - Учебное пособие, 2005 - 536с.
3. НОУ ИНТУИТ | Учебный курс | Основы операционных систем: [Электронный ресурс]. URL: http://www.intuit.ru.
4. Проект OpenNet: MAN (FreeBSD и Linux): [Электронный ресурс]. URL: http://www.opennet.ru.
5. Beginning Game Programming v2.0: [Электронный ресурс]. URL: <https://lazyfoo.net/tutorials/SDL/>.
6. ПРИЛОЖЕНИЕ А

Изображение выглядит как снимок экрана

Автоматически созданное описаниеИзображение выглядит как снимок экрана

Автоматически созданное описаниеИзображение выглядит как монитор, внутренний, экран, другой

Автоматически созданное описаниеИзображение выглядит как монитор, экран, время, зеленый

Автоматически созданное описаниеИзображение выглядит как монитор, внутренний, экран, компьютер

Автоматически созданное описаниеИзображение выглядит как монитор, внутренний, экран, компьютер

Автоматически созданное описаниеИзображение выглядит как монитор, внутренний, экран, другой

Автоматически созданное описаниеИзображение выглядит как объект, часы, монитор, знак

Автоматически созданное описаниеИзображение выглядит как зеленый, вода

Автоматически созданное описаниеИзображение выглядит как снимок экрана

Автоматически созданное описание

1. ПРИЛОЖЕНИЕ Б

Листинг В1 — Код из файла server.c

#include <sys/types.h>

#include <sys/socket.h>

#include <netinet/in.h>

#include <arpa/inet.h>

#include <string.h>

#include <stdio.h>

#include <errno.h>

#include <unistd.h>

#include <limits.h>

#include <stdlib.h>

#include <pthread.h>

#include <fcntl.h>

#include <sys/sem.h>

#include "sem.h"

#include "../shared.h"

#include "defines.h"

#include "settings.h"

#include "types.h"

#include "serverGameLogic.h"

#include "wormLogic.h"

#include "bot.h"

int setUpServer(char \*port)

{

  int sockfd;

  struct sockaddr\_in servaddr;

  bzero(&servaddr, sizeof(servaddr));

  servaddr.sin\_family = AF\_INET;

  servaddr.sin\_port = htons(atoi(port));

  servaddr.sin\_addr.s\_addr = htonl(INADDR\_ANY);

  sockfd = socket(PF\_INET, SOCK\_STREAM, 0);

  if (bind(sockfd, (struct sockaddr \*)&servaddr, sizeof(servaddr)) < 0)

  {

    servaddr.sin\_port = 0;

    if (bind(sockfd, (struct sockaddr \*)&servaddr, sizeof(servaddr)) < 0)

    {

      perror(NULL);

      close(sockfd);

      exit(1);

    }

  }

  socklen\_t servlen = sizeof(servaddr);

  listen(sockfd, 5);

  getsockname(sockfd, (struct sockaddr \*)&servaddr, &servlen);

  printf("Listening on port: %d\n", ntohs(servaddr.sin\_port));

  fcntl(sockfd, F\_SETFL, fcntl(sockfd, F\_GETFL, 0) | O\_NONBLOCK);

  return sockfd;

}

int main(int argc, char \*argv[])

{

  if (argc < 2)

  {

    printf("Please specify port!\n");

    return -1;

  }

  int sockfd = setUpServer(argv[1]);

  int semId = semget(IPC\_PRIVATE, PLAYERS\_SEM\_OFFSET + MAX\_PLAYERS, 0600 | IPC\_CREAT);

  for (int i = 0; i < PLAYERS\_SEM\_OFFSET + MAX\_PLAYERS; i++)

  {

    semUnlock(semId, i);

  }

  struct GameState \*gameState = malloc(sizeof(struct GameState));

  struct Map \*map = malloc(sizeof(struct Map));

  struct PlayerState \*playersState = malloc(sizeof(struct PlayerState) \* MAX\_PLAYERS);

  bzero(playersState, sizeof(\*playersState) \* MAX\_PLAYERS);

  while (1)

  {

    resetState(semId, gameState, playersState, map);

    if (acceptPlayers(sockfd, semId, gameState, playersState, map) != 0)

    {

      printf("Socket failed!\n");

      return -1;

    }

    addBots(BOTS, semId, gameState, playersState, map);

    printf("All players are ready!\n");

    setupGame(semId, playersState);

    playGame(semId, gameState, playersState, map);

    printf("Game finished\n");

    sleep(SCOREBOARD\_TIME);

  }

}

Листинг В2 — Код из файла sem.h

#pragma once

#define GAME\_STATE\_SEM 0

#define MAP\_SEM 1

#define PLAYERS\_SEM\_OFFSET 2

void semUnlock(int semId, int n);

void semLock(int semId, int n);

Листинг В3 — Код из файла sem.c

#include "sem.h"

#include "stdio.h"

#include <sys/sem.h>

void sem(int semId, int n, int d)

{

  struct sembuf op;

  op.sem\_op = d;

  op.sem\_flg = 0;

  op.sem\_num = n;

  semop(semId, &op, 1);

}

void semUnlock(int semId, int n)

{

  //printf("u-%d-%d\n", semId, n);

  sem(semId, n, 1);

}

void semLock(int semId, int n)

{

  //printf("l-%d-%d\n", semId, n);

  sem(semId, n, -1);

}

Листинг В4 — Код из файла settings.h

#pragma once

#define MAP\_SIZE 16

#define MAX\_PLAYERS 8

#define BOTS 8

#define START\_SNAKE\_LEN 3

#define FOOD\_TARGET 4

#define SCOREBOARD\_TIME 5

#define BOT\_SPEED\_UP 1

#define TICKRATE 24

Листинг В5 — Код из файла types.h

#pragma once

#include "settings.h"

struct GameState

{

  char gamePhase;

  char playersConnected;

  char playersReady;

  char activeWorm;

  char foodOnMap;

};

struct PlayerState

{

  char connected;

  char ready;

  int length;

  char direction;

  char prevDirection;

};

struct Map

{

  int field[MAP\_SIZE][MAP\_SIZE];

  int rowVersions[MAP\_SIZE];

  int mapVersion;

};

struct WormThreadArgs

{

  int sockfd;

  int semId;

  char ind;

  struct PlayerState \*playersState;

  struct GameState \*gameState;

  struct Map \*map;

};

Листинг В6 — Код из файла defines.h

#pragma once

#define MAX\_WORM\_LEN (MAP\_SIZE \* MAP\_SIZE)

#define FOOD\_OFFSET (2 + (MAP\_SIZE \* MAP\_SIZE) \* (MAX\_PLAYERS))

Листинг В7 — Код из файла shared.h

#pragma once

struct WaitingState

{

  char gamePhase;

  char playersConnected;

  char playersReady;

};

struct InGameState

{

  char gamePhase;

  char activeWorm;

  int score;

};

struct ScoreboardState

{

  char gamePhase;

  char players;

};

struct ScoreboardRecord

{

  char ind;

  int score;

};

enum GamePhases

{

  WAITING\_FOR\_PLAYERS,

  IN\_PROGRESS,

  SCOREBOARD

};

Листинг В8 — Код из файла serverGameLogic.h

#pragma once

#include <netinet/in.h>

#include "string.h"

#include "settings.h"

#include "types.h"

void resetState(int semId, struct GameState \*gameState, struct PlayerState \*playersState, struct Map \*map);

int acceptPlayers(int sockfd, int semId, struct GameState \*gameState, struct PlayerState \*playersState, struct Map \*map);

void addBots(int count, int semId, struct GameState \*gameState, struct PlayerState \*playersState, struct Map \*map);

void setupGame(int semId, struct PlayerState \*playersState);

void playGame(int semId, struct GameState \*gameState, struct PlayerState \*playersState, struct Map \*map);

Листинг В9 — Код из файла serverGameLogic.c

#include "serverGameLogic.h"

#include "stdlib.h"

#include "unistd.h"

#include <pthread.h>

#include "math.h"

#include "sem.h"

#include "../shared.h"

#include "defines.h"

#include "settings.h"

#include "wormLogic.h"

#include "bot.h"

void loadMap(int map[MAP\_SIZE][MAP\_SIZE])

{

  for (int i = 0; i < MAP\_SIZE; i++)

  {

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

    {

      if (i == 0 || j == 0 || i == MAP\_SIZE - 1 || j == MAP\_SIZE - 1)

      {

        map[i][j] = 1;

      }

      else

      {

        map[i][j] = 0;

      }

    }

  }

  int inRow = ceil(sqrt(MAX\_PLAYERS));

  float offset;

  if (inRow > 1)

  {

    offset = 1.0 \* (MAP\_SIZE - 3) / (inRow - 1);

  }

  else

  {

    offset = 0;

  }

  for (int i = 0; i < MAX\_PLAYERS; i++)

  {

    int y = round(offset \* (i / inRow));

    int x = round(offset \* (i % inRow));

    map[1 + y][1 + x] = 2 + MAX\_WORM\_LEN \* i;

  }

}

void resetState(int semId, struct GameState \*gameState, struct PlayerState \*playersState, struct Map \*map)

{

  for (int i = 0; i < MAX\_PLAYERS; i++)

  {

    semLock(semId, i + PLAYERS\_SEM\_OFFSET);

  }

  semLock(semId, MAP\_SEM);

  semLock(semId, GAME\_STATE\_SEM);

  gameState->gamePhase = WAITING\_FOR\_PLAYERS;

  gameState->activeWorm = 0;

  gameState->foodOnMap = 0;

  for (int i = 0; i < MAX\_PLAYERS; i++)

  {

    playersState[i].direction = 0;

    playersState[i].prevDirection = 0;

    playersState[i].length = 0;

    playersState[i].ready = 0;

  }

  bzero(map->rowVersions, sizeof(\*map->rowVersions) \* MAP\_SIZE);

  map->mapVersion = 0;

  loadMap(map->field);

  for (int i = 0; i < MAX\_PLAYERS; i++)

  {

    semUnlock(semId, i + PLAYERS\_SEM\_OFFSET);

  }

  semUnlock(semId, MAP\_SEM);

  semUnlock(semId, GAME\_STATE\_SEM);

}

int acceptPlayers(int sockfd, int semId, struct GameState \*gameState, struct PlayerState \*playersState, struct Map \*map)

{

  struct sockaddr\_in cliaddr;

  socklen\_t clilen = sizeof(cliaddr);

  while (1)

  {

    int newsockfd = accept(sockfd, (struct sockaddr \*)&cliaddr, &clilen);

    if (newsockfd == 0)

    {

      return -1;

    }

    else if (newsockfd > 0)

    {

      for (int i = 0; i < MAX\_PLAYERS; i++)

      {

        if (!playersState[i].connected)

        {

          pthread\_t newWormThread;

          struct WormThreadArgs \*newWormThreadArgs = malloc(sizeof(struct WormThreadArgs));

          newWormThreadArgs->sockfd = newsockfd;

          newWormThreadArgs->playersState = playersState;

          newWormThreadArgs->gameState = gameState;

          newWormThreadArgs->semId = semId;

          newWormThreadArgs->ind = (char)i;

          newWormThreadArgs->map = map;

          semLock(semId, PLAYERS\_SEM\_OFFSET + i);

          pthread\_create(&newWormThread, NULL, wormTask, (void \*)newWormThreadArgs);

          break;

        }

      }

    }

    semLock(semId, GAME\_STATE\_SEM);

    gameState->playersConnected = 0;

    gameState->playersReady = 0;

    for (int i = 0; i < MAX\_PLAYERS; i++)

    {

      semLock(semId, PLAYERS\_SEM\_OFFSET + i);

      if (playersState[i].connected)

      {

        gameState->playersConnected++;

      }

      if (playersState[i].ready)

      {

        gameState->playersReady++;

      }

      semUnlock(semId, PLAYERS\_SEM\_OFFSET + i);

    }

    if (gameState->playersConnected > 0 && gameState->playersConnected == gameState->playersReady)

    {

      gameState->gamePhase = IN\_PROGRESS;

      semUnlock(semId, GAME\_STATE\_SEM);

      return 0;

    }

    semUnlock(semId, GAME\_STATE\_SEM);

    usleep(1000000 / TICKRATE);

  }

  return -1;

}

void addBots(int count, int semId, struct GameState \*gameState, struct PlayerState \*playersState, struct Map \*map)

{

  for (int i = 0; i < MAX\_PLAYERS && count > 0; i++)

  {

    if (!playersState[i].connected)

    {

      pthread\_t newWormThread;

      struct WormThreadArgs \*newWormThreadArgs = malloc(sizeof(struct WormThreadArgs));

      newWormThreadArgs->playersState = playersState;

      newWormThreadArgs->gameState = gameState;

      newWormThreadArgs->semId = semId;

      newWormThreadArgs->ind = (char)i;

      newWormThreadArgs->map = map;

      semLock(semId, PLAYERS\_SEM\_OFFSET + i);

      pthread\_create(&newWormThread, NULL, botTask, (void \*)newWormThreadArgs);

      count--;

    }

  }

}

void setupGame(int semId, struct PlayerState \*playersState)

{

  for (int i = 0; i < MAX\_PLAYERS; i++)

  {

    semLock(semId, i + PLAYERS\_SEM\_OFFSET);

    if (playersState[i].ready)

    {

      playersState[i].length = START\_SNAKE\_LEN;

    }

    semUnlock(semId, i + PLAYERS\_SEM\_OFFSET);

  }

  for (int i = 0; i < MAX\_PLAYERS; i++)

  {

    semLock(semId, PLAYERS\_SEM\_OFFSET + i);

    playersState[i].ready = 1;

    semUnlock(semId, PLAYERS\_SEM\_OFFSET + i);

  }

}

char randFood()

{

  int r = rand() % 512 + 1;

  int n = 9;

  while (r > 0)

  {

    n--;

    r /= 2;

  }

  return 1 + n;

}

char gameLoop(int semId, struct GameState \*gameState, int map[MAP\_SIZE][MAP\_SIZE], struct PlayerState playersState[MAX\_PLAYERS], int rowVersions[MAP\_SIZE], int \*mapVersion)

{

  for (int i = 0; i < MAX\_PLAYERS; i++)

  {

    semLock(semId, i + PLAYERS\_SEM\_OFFSET);

  }

  semLock(semId, MAP\_SEM);

  semLock(semId, GAME\_STATE\_SEM);

  char allPlayersMadeTurn = 1;

  char playersOnField = 0;

  for (int i = 0; i < MAX\_PLAYERS; i++)

  {

    if (playersState[i].ready)

    {

      playersOnField++;

      if (!playersState[i].direction)

      {

        allPlayersMadeTurn = 0;

      }

    }

  }

  if (playersOnField == 0)

  {

    gameState->gamePhase = SCOREBOARD;

  }

  else

  {

    char toRemoveC = 0;

    char toRemove[MAX\_PLAYERS];

    if (allPlayersMadeTurn)

    {

      char lenIncrease[MAX\_PLAYERS];

      char madeMove[MAX\_PLAYERS];

      bzero(madeMove, sizeof(madeMove));

      bzero(lenIncrease, sizeof(lenIncrease));

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

      {

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

        {

          if (map[i][j] > 1 && map[i][j] < FOOD\_OFFSET)

          {

            int snakeId = (map[i][j] - 2) / MAX\_WORM\_LEN;

            if (!playersState[snakeId].ready)

              continue;

            int segmentInd = (map[i][j] - 2) % MAX\_WORM\_LEN;

            if (segmentInd != 0 || !madeMove[snakeId])

            {

              map[i][j]++;

            }

            if (segmentInd == 0 && !madeMove[snakeId])

            {

              madeMove[snakeId] = 1;

              int targetX, targetY;

              char direction = playersState[snakeId].direction;

              if (direction == 1)

              {

                targetX = i;

                targetY = j - 1;

              }

              else if (direction == 2)

              {

                targetX = i + 1;

                targetY = j;

              }

              else if (direction == 3)

              {

                targetX = i;

                targetY = j + 1;

              }

              else if (direction == 4)

              {

                targetX = i - 1;

                targetY = j;

              }

              if (targetX < 0 || targetX >= MAP\_SIZE || targetY < 0 || targetY >= MAP\_SIZE)

              {

                playersState[snakeId].ready = 0;

                toRemove[toRemoveC] = snakeId;

                toRemoveC++;

              }

              else if (map[targetX][targetY])

              {

                if (map[targetX][targetY] < FOOD\_OFFSET)

                {

                  if (map[targetX][targetY] > 1)

                  {

                    int targetSnakeId = (map[targetX][targetY] - 2) / MAX\_WORM\_LEN;

                    int targetSegmentId = (map[targetX][targetY] - 2) % MAX\_WORM\_LEN;

                    if (targetSegmentId + 1 >= playersState[targetSnakeId].length && (direction == 3 || direction == 2))

                    {

                      map[targetX][targetY] = 2 + MAX\_WORM\_LEN \* snakeId;

                      (\*mapVersion)++;

                      rowVersions[targetX]++;

                      continue;

                    }

                  }

                  playersState[snakeId].ready = 0;

                  toRemove[toRemoveC] = snakeId;

                  toRemoveC++;

                }

                else

                {

                  gameState->foodOnMap--;

                  lenIncrease[snakeId] = map[targetX][targetY] - FOOD\_OFFSET;

                  map[targetX][targetY] = 2 + MAX\_WORM\_LEN \* snakeId;

                  (\*mapVersion)++;

                  rowVersions[targetX]++;

                }

              }

              else

              {

                map[targetX][targetY] = 2 + MAX\_WORM\_LEN \* snakeId;

                (\*mapVersion)++;

                rowVersions[targetX]++;

              }

            }

            else if (segmentInd + 1 >= playersState[snakeId].length)

            {

              map[i][j] = 0;

              (\*mapVersion)++;

              rowVersions[i]++;

            }

          }

        }

      }

      for (int i = 0; i < MAX\_PLAYERS; i++)

      {

        if (playersState[i].ready)

        {

          playersState[i].length += lenIncrease[i];

          playersState[i].prevDirection = playersState[i].direction;

          playersState[i].direction = 0;

        }

      }

    }

    if (allPlayersMadeTurn || playersState[gameState->activeWorm].direction || !playersState[gameState->activeWorm].ready)

    {

      for (int i = 1; i <= MAX\_PLAYERS; i++)

      {

        int ind = (gameState->activeWorm + i) % MAX\_PLAYERS;

        if (playersState[ind].ready && !playersState[ind].direction)

        {

          gameState->activeWorm = ind;

          break;

        }

      }

    }

    for (int i = 0; i < MAX\_PLAYERS; i++)

    {

      if (!playersState[i].connected && playersState[i].ready)

      {

        playersState[i].ready = 0;

        toRemove[toRemoveC] = i;

        toRemoveC++;

      }

    }

    for (int k = 0; k < toRemoveC; k++)

    {

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

      {

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

        {

          if (map[i][j] > 1 && map[i][j] < FOOD\_OFFSET)

          {

            int snakeId = (map[i][j] - 2) / MAX\_WORM\_LEN;

            if (snakeId == toRemove[k])

            {

              map[i][j] = 0;

              (\*mapVersion)++;

              rowVersions[i]++;

            }

          }

        }

      }

    }

    int a = 0;

    while (FOOD\_TARGET - gameState->foodOnMap > 0 && a < 10000)

    {

      int x = rand() % MAP\_SIZE;

      int y = rand() % MAP\_SIZE;

      if (!map[x][y])

      {

        map[x][y] = FOOD\_OFFSET + randFood();

        (\*mapVersion)++;

        rowVersions[x]++;

        gameState->foodOnMap++;

      }

      a++;

    }

    for (int i = 0; i < MAP\_SIZE && FOOD\_TARGET - gameState->foodOnMap > 0; i++)

    {

      for (int j = 0; j < MAP\_SIZE && FOOD\_TARGET - gameState->foodOnMap > 0; j++)

      {

        if (!map[i][j])

        {

          map[i][j] = FOOD\_OFFSET + randFood();

          (\*mapVersion)++;

          rowVersions[i]++;

          gameState->foodOnMap++;

        }

      }

    }

  }

  for (int i = 0; i < MAX\_PLAYERS; i++)

  {

    semUnlock(semId, i + PLAYERS\_SEM\_OFFSET);

  }

  semUnlock(semId, MAP\_SEM);

  semUnlock(semId, GAME\_STATE\_SEM);

  return (playersOnField > 0);

}

void playGame(int semId, struct GameState \*gameState, struct PlayerState \*playersState, struct Map \*map)

{

  while (gameLoop(semId, gameState, map->field, playersState, map->rowVersions, &map->mapVersion))

  {

    usleep(1000000 / TICKRATE);

  }

}

Листинг В10 — Код из файла wormLogic.h

#pragma once

void \*wormTask(void \*targs);

Листинг В11 — Код из файла wormLogic.c

#include "wormLogic.h"

#include <sys/socket.h>

#include <unistd.h>

#include "stdlib.h"

#include "sem.h"

#include "../shared.h"

#include "defines.h"

#include "settings.h"

#include "types.h"

void encodeMapRow(char encoded[MAP\_SIZE], int map[MAP\_SIZE][MAP\_SIZE], int r)

{

  for (int i = 0; i < MAP\_SIZE; i++)

  {

    if (map[r][i] <= 1)

    {

      encoded[i] = map[r][i];

    }

    else if (map[r][i] < FOOD\_OFFSET)

    {

      encoded[i] = (map[r][i] - 2) / MAX\_WORM\_LEN + 2;

    }

    else

    {

      encoded[i] = -(map[r][i] - FOOD\_OFFSET);

    }

  }

}

void disconnect(int sockfd, int semId, struct PlayerState \*myState, int ind)

{

  semLock(semId, PLAYERS\_SEM\_OFFSET + ind);

  myState->connected = 0;

  semUnlock(semId, PLAYERS\_SEM\_OFFSET + ind);

  close(sockfd);

}

void \*wormTask(void \*targs)

{

  struct WormThreadArgs \*args = targs;

  int sockfd = args->sockfd;

  int semId = args->semId;

  char ind = args->ind;

  struct PlayerState \*playersState = args->playersState;

  struct PlayerState \*myState = playersState + ind;

  struct GameState \*gameState = args->gameState;

  struct Map \*map = args->map;

  myState->connected = 1;

  myState->ready = 0;

  semUnlock(semId, PLAYERS\_SEM\_OFFSET + ind);

  if (!write(sockfd, &ind, sizeof(ind)))

  {

    disconnect(sockfd, semId, myState, ind);

    return NULL;

  }

  while (1)

  {

    struct WaitingState waitingState;

    do

    {

      char status;

      if (!read(sockfd, &status, sizeof(status)))

      {

        disconnect(sockfd, semId, myState, ind);

        return NULL;

      }

      semLock(semId, PLAYERS\_SEM\_OFFSET + ind);

      if (status)

      {

        myState->ready = 1;

      }

      else

      {

        myState->ready = 0;

      }

      semUnlock(semId, PLAYERS\_SEM\_OFFSET + ind);

      semLock(semId, GAME\_STATE\_SEM);

      waitingState.gamePhase = gameState->gamePhase;

      waitingState.playersConnected = gameState->playersConnected;

      waitingState.playersReady = gameState->playersReady;

      semUnlock(semId, GAME\_STATE\_SEM);

      if (!write(sockfd, &waitingState, sizeof(waitingState)))

      {

        disconnect(sockfd, semId, myState, ind);

        return NULL;

      }

    } while (waitingState.gamePhase == WAITING\_FOR\_PLAYERS);

    char mapSize = MAP\_SIZE;

    if (!write(sockfd, &mapSize, sizeof(mapSize)))

    {

      disconnect(sockfd, semId, myState, ind);

      return NULL;

    }

    struct InGameState inGameState;

    char mapRowBuff[MAP\_SIZE];

    int mapRowVersions[MAP\_SIZE];

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

    {

      mapRowVersions[i] = -1;

    }

    int mapVersion = -1;

    do

    {

      semLock(semId, MAP\_SEM);

      char status;

      if (map->mapVersion > mapVersion)

      {

        mapVersion = map->mapVersion;

        status = 1;

        if (!write(sockfd, &status, sizeof(status)))

        {

          semUnlock(semId, MAP\_SEM);

          disconnect(sockfd, semId, myState, ind);

          return NULL;

        }

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

        {

          if (map->rowVersions[i] > mapRowVersions[i])

          {

            mapRowVersions[i] = map->rowVersions[i];

            status = 1;

            if (!write(sockfd, &status, sizeof(status)))

            {

              semUnlock(semId, MAP\_SEM);

              disconnect(sockfd, semId, myState, ind);

              return NULL;

            }

            encodeMapRow(mapRowBuff, map->field, i);

            if (!write(sockfd, &mapRowBuff, sizeof(mapRowBuff)))

            {

              semUnlock(semId, MAP\_SEM);

              disconnect(sockfd, semId, myState, ind);

              return NULL;

            }

          }

          else

          {

            status = 0;

            if (!write(sockfd, &status, sizeof(status)))

            {

              semUnlock(semId, MAP\_SEM);

              disconnect(sockfd, semId, myState, ind);

              return NULL;

            }

          }

        }

        semUnlock(semId, MAP\_SEM);

      }

      else

      {

        status = 0;

        if (!write(sockfd, &status, sizeof(status)))

        {

          semUnlock(semId, MAP\_SEM);

          disconnect(sockfd, semId, myState, ind);

          return NULL;

        }

        semUnlock(semId, MAP\_SEM);

      }

      semLock(semId, GAME\_STATE\_SEM);

      inGameState.gamePhase = gameState->gamePhase;

      inGameState.activeWorm = gameState->activeWorm;

      semUnlock(semId, GAME\_STATE\_SEM);

      semLock(semId, ind + PLAYERS\_SEM\_OFFSET);

      inGameState.score = myState->length - START\_SNAKE\_LEN;

      semUnlock(semId, ind + PLAYERS\_SEM\_OFFSET);

      if (!write(sockfd, &inGameState, sizeof(inGameState)))

      {

        disconnect(sockfd, semId, myState, ind);

        return NULL;

      }

      char c;

      if (!read(sockfd, &c, sizeof(c)))

      {

        disconnect(sockfd, semId, myState, ind);

        return NULL;

      }

      if (!myState->direction && c && (!myState->prevDirection || myState->prevDirection == c || myState->prevDirection % 2 != c % 2))

      {

        semUnlock(semId, ind + PLAYERS\_SEM\_OFFSET);

        semLock(semId, GAME\_STATE\_SEM);

        if (gameState->activeWorm == ind)

        {

          semUnlock(semId, GAME\_STATE\_SEM);

          semLock(semId, ind + PLAYERS\_SEM\_OFFSET);

          myState->direction = c;

          semUnlock(semId, ind + PLAYERS\_SEM\_OFFSET);

        }

        else

        {

          semUnlock(semId, GAME\_STATE\_SEM);

        }

      }

      else

      {

        semUnlock(semId, ind + PLAYERS\_SEM\_OFFSET);

      }

    } while (inGameState.gamePhase == IN\_PROGRESS);

    if (inGameState.gamePhase == WAITING\_FOR\_PLAYERS) continue;

    char sync = 1;

    struct ScoreboardState scoreboardState;

    semLock(semId, GAME\_STATE\_SEM);

    scoreboardState.gamePhase = gameState->gamePhase;

    scoreboardState.players = gameState->playersReady;

    if (!write(sockfd, &scoreboardState, sizeof(scoreboardState)))

    {

      semUnlock(semId, GAME\_STATE\_SEM);

      disconnect(sockfd, semId, myState, ind);

      return NULL;

    }

    semUnlock(semId, GAME\_STATE\_SEM);

    for (int i = 0; i < MAX\_PLAYERS; i++)

    {

      semLock(semId, PLAYERS\_SEM\_OFFSET + i);

      if (playersState[i].length > 0)

      {

        struct ScoreboardRecord record;

        record.ind = i;

        record.score = playersState[i].length - START\_SNAKE\_LEN;

        if (!write(sockfd, &record, sizeof(record)))

        {

          semUnlock(semId, PLAYERS\_SEM\_OFFSET + i);

          disconnect(sockfd, semId, myState, ind);

          return NULL;

        }

      }

      semUnlock(semId, PLAYERS\_SEM\_OFFSET + i);

    }

    if (!read(sockfd, &sync, sizeof(sync)))

    {

      disconnect(sockfd, semId, myState, ind);

      return NULL;

    }

    while (scoreboardState.gamePhase == SCOREBOARD)

    {

      semLock(semId, GAME\_STATE\_SEM);

      scoreboardState.gamePhase = gameState->gamePhase;

      scoreboardState.players = gameState->playersReady;

      if (!write(sockfd, &scoreboardState, sizeof(scoreboardState)))

      {

        semUnlock(semId, GAME\_STATE\_SEM);

        disconnect(sockfd, semId, myState, ind);

        return NULL;

      }

      semUnlock(semId, GAME\_STATE\_SEM);

      if (!read(sockfd, &sync, sizeof(sync)))

      {

        disconnect(sockfd, semId, myState, ind);

        return NULL;

      }

    }

  }

}

Листинг В10 — Код из файла bot.h

#pragma once

#include "settings.h"

void \*botTask(void \*targs);

Листинг В11 — Код из файла bot.c

#include "bot.h"

#include <stdlib.h>

#include <unistd.h>

#include "sem.h"

#include "types.h"

#include "defines.h"

#include "../shared.h"

const int WALL = -1;

const int BLANK = -2;

const int TARGET = -3;

void encodeMap(char \*x, char \*y, int encoded[MAP\_SIZE][MAP\_SIZE], int map[MAP\_SIZE][MAP\_SIZE], char ind)

{

  for (int i = 0; i < MAP\_SIZE; i++)

  {

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

    {

      if (map[i][j] == 0)

      {

        encoded[i][j] = BLANK;

      }

      else if (map[i][j] == 1)

      {

        encoded[i][j] = WALL;

      }

      else if (map[i][j] < FOOD\_OFFSET)

      {

        int snakeId = (map[i][j] - 2) / MAX\_WORM\_LEN;

        int segmentInd = (map[i][j] - 2) % MAX\_WORM\_LEN;

        if (snakeId == ind && segmentInd == 0)

        {

          (\*x) = j;

          (\*y) = i;

        }

        encoded[i][j] = WALL;

      }

      else

      {

        encoded[i][j] = TARGET;

      }

    }

  }

}

char makeTurn(char sx, char sy, int encoded[MAP\_SIZE][MAP\_SIZE])

{

  int tx = -1, ty = -1;

  int mx = 0, my = 0;

  int dx[4] = {1, 0, -1, 0};

  int dy[4] = {0, 1, 0, -1};

  int d, x, y, k;

  char stop;

  d = 0;

  encoded[sy][sx] = 0;

  char targetFound = 0;

  do

  {

    stop = 1;

    for (y = 0; y < MAP\_SIZE && !targetFound; ++y)

      for (x = 0; x < MAP\_SIZE && !targetFound; ++x)

        if (encoded[y][x] == d)

        {

          for (k = 0; k < 4 && !targetFound; ++k)

          {

            int iy = y + dy[k], ix = x + dx[k];

            if (iy >= 0 && iy < MAP\_SIZE && ix >= 0 && ix < MAP\_SIZE)

            {

              if (encoded[iy][ix] == BLANK)

              {

                stop = 0;

                encoded[iy][ix] = d + 1;

                mx = ix;

                my = iy;

              }

              else if (encoded[iy][ix] == TARGET)

              {

                targetFound = 1;

                tx = ix;

                ty = iy;

                encoded[iy][ix] = d + 1;

              }

            }

          }

        }

    d++;

  } while (!stop && !targetFound);

  if (tx == -1 || ty == -1)

  {

    tx = mx;

    ty = my;

  }

  int len = encoded[ty][tx];

  x = tx;

  y = ty;

  d = len;

  int px[MAP\_SIZE \* MAP\_SIZE];

  int py[MAP\_SIZE \* MAP\_SIZE];

  while (d > 0)

  {

    px[d] = x;

    py[d] = y;

    d--;

    for (k = 0; k < 4; ++k)

    {

      int iy = y + dy[k], ix = x + dx[k];

      if (iy >= 0 && iy < MAP\_SIZE && ix >= 0 && ix < MAP\_SIZE &&

          encoded[iy][ix] == d)

      {

        x = x + dx[k];

        y = y + dy[k];

        break;

      }

    }

  }

  if (px[1] > sx)

  {

    return 3;

  }

  else if (px[1] < sx)

  {

    return 1;

  }

  else if (py[1] < sy)

  {

    return 4;

  }

  else

  {

    return 2;

  }

}

void \*botTask(void \*targs)

{

  struct WormThreadArgs \*args = targs;

  int semId = args->semId;

  char ind = args->ind;

  struct PlayerState \*playersState = args->playersState;

  struct PlayerState \*myState = playersState + ind;

  struct GameState \*gameState = args->gameState;

  struct Map \*map = args->map;

  free(targs);

  myState->connected = 1;

  myState->ready = 1;

  semUnlock(semId, PLAYERS\_SEM\_OFFSET + ind);

  semLock(semId, GAME\_STATE\_SEM);

  gameState->playersConnected++;

  gameState->playersReady++;

  semUnlock(semId, GAME\_STATE\_SEM);

  struct WaitingState waitingState;

  do

  {

    semLock(semId, GAME\_STATE\_SEM);

    waitingState.gamePhase = gameState->gamePhase;

    semUnlock(semId, GAME\_STATE\_SEM);

  } while (waitingState.gamePhase == WAITING\_FOR\_PLAYERS);

  struct InGameState inGameState;

  char isAlive = 1;

  do

  {

    semLock(semId, GAME\_STATE\_SEM);

    inGameState.gamePhase = gameState->gamePhase;

    semUnlock(semId, GAME\_STATE\_SEM);

    semLock(semId, ind + PLAYERS\_SEM\_OFFSET);

    if (!myState->direction && myState->ready)

    {

      semUnlock(semId, ind + PLAYERS\_SEM\_OFFSET);

      semLock(semId, GAME\_STATE\_SEM);

      if (BOT\_SPEED\_UP || gameState->activeWorm == ind)

      {

        semUnlock(semId, GAME\_STATE\_SEM);

        semLock(semId, MAP\_SEM);

        int encodedMap[MAP\_SIZE][MAP\_SIZE];

        char x, y;

        encodeMap(&x, &y, encodedMap, map->field, ind);

        semUnlock(semId, MAP\_SEM);

        semLock(semId, ind + PLAYERS\_SEM\_OFFSET);

        myState->direction = makeTurn(x, y, encodedMap);

        semUnlock(semId, ind + PLAYERS\_SEM\_OFFSET);

      }

      else

      {

        semUnlock(semId, GAME\_STATE\_SEM);

      }

    }

    else

    {

      isAlive = myState->ready;

      semUnlock(semId, ind + PLAYERS\_SEM\_OFFSET);

    }

    usleep(1000000 / TICKRATE);

  } while (inGameState.gamePhase == IN\_PROGRESS && isAlive);

  semLock(semId, ind + PLAYERS\_SEM\_OFFSET);

  myState->connected = 0;

  semUnlock(semId, ind + PLAYERS\_SEM\_OFFSET);

}

Листинг В12 — Код из файла LTexture.h

#include <SDL2/SDL.h>

#include <SDL2/SDL\_ttf.h>

#include <string>

//Texture wrapper class

class LTexture

{

public:

    //Initializes variables

    LTexture();

    //Deallocates memory

    ~LTexture();

    //Creates image from font string

    bool loadFromRenderedText(std::string textureText, TTF\_Font \*font, SDL\_Color textColor);

    //Deallocates texture

    void free();

    //Set color modulation

    void setColor(Uint8 red, Uint8 green, Uint8 blue);

    //Set blending

    void setBlendMode(SDL\_BlendMode blending);

    //Set alpha modulation

    void setAlpha(Uint8 alpha);

    //Renders texture at given point

    void render(int x, int y, SDL\_Rect \*clip = NULL, double angle = 0.0, SDL\_Point \*center = NULL, SDL\_RendererFlip flip = SDL\_FLIP\_NONE);

    //Gets image dimensions

    int getWidth();

    int getHeight();

    void init(SDL\_Renderer \*renderer);

private:

    //The actual hardware texture

    SDL\_Texture \*mTexture;

    //Image dimensions

    int mWidth;

    int mHeight;

    SDL\_Renderer \*gRenderer;

};

Листинг В13 — Код из файла LTexture.cpp

#include "LTexture.h"

LTexture::LTexture()

{

    //Initialize

    mTexture = NULL;

    mWidth = 0;

    mHeight = 0;

}

LTexture::~LTexture()

{

    //Deallocate

    free();

}

void LTexture::init(SDL\_Renderer \*renderer)

{

    this->gRenderer = renderer;

}

bool LTexture::loadFromRenderedText(std::string textureText, TTF\_Font \*font, SDL\_Color textColor)

{

    //Get rid of preexisting texture

    free();

    //Render text surface

    SDL\_Surface \*textSurface = TTF\_RenderText\_Solid(font, textureText.c\_str(), textColor);

    if (textSurface == NULL)

    {

        printf("Unable to render text surface! SDL\_ttf Error: %s\n", TTF\_GetError());

    }

    else

    {

        //Create texture from surface pixels

        mTexture = SDL\_CreateTextureFromSurface(gRenderer, textSurface);

        if (mTexture == NULL)

        {

            printf("Unable to create texture from rendered text! SDL Error: %s\n", SDL\_GetError());

        }

        else

        {

            //Get image dimensions

            mWidth = textSurface->w;

            mHeight = textSurface->h;

        }

        //Get rid of old surface

        SDL\_FreeSurface(textSurface);

    }

    //Return success

    return mTexture != NULL;

}

void LTexture::free()

{

    //Free texture if it exists

    if (mTexture != NULL)

    {

        SDL\_DestroyTexture(mTexture);

        mTexture = NULL;

        mWidth = 0;

        mHeight = 0;

    }

}

void LTexture::setColor(Uint8 red, Uint8 green, Uint8 blue)

{

    //Modulate texture rgb

    SDL\_SetTextureColorMod(mTexture, red, green, blue);

}

void LTexture::setBlendMode(SDL\_BlendMode blending)

{

    //Set blending function

    SDL\_SetTextureBlendMode(mTexture, blending);

}

void LTexture::setAlpha(Uint8 alpha)

{

    //Modulate texture alpha

    SDL\_SetTextureAlphaMod(mTexture, alpha);

}

void LTexture::render(int x, int y, SDL\_Rect \*clip, double angle, SDL\_Point \*center, SDL\_RendererFlip flip)

{

    //Set rendering space and render to screen

    SDL\_Rect renderQuad = {x, y, mWidth, mHeight};

    //Set clip rendering dimensions

    if (clip != NULL)

    {

        renderQuad.w = clip->w;

        renderQuad.h = clip->h;

    }

    //Render to screen

    SDL\_RenderCopyEx(gRenderer, mTexture, clip, &renderQuad, angle, center, flip);

}

int LTexture::getWidth()

{

    return mWidth;

}

int LTexture::getHeight()

{

    return mHeight;

}

Листинг В14 — Код из файла colors.h

#include <SDL2/SDL.h>

#define DARK 79, 91, 98, 255

#define BLACK 0, 10, 18, 255

#define BG 0x21, 0x21, 0x21, 255

SDL\_Color TWHITE = {255, 255, 255};

SDL\_Color TGREEN = {118, 210, 117};

SDL\_Color TRED = {240, 85, 69};

//first wall, then worms

unsigned char MAP\_COLORS[][3] = {

    {0x21, 0x21, 0x21},

    {0x8d, 0x70, 0xc9},

    {0x66, 0xb8, 0x58},

    {0xc8, 0x58, 0x8c},

    {0xbc, 0xaa, 0x41},

    {0x4b, 0xb2, 0xb5},

    {0xce, 0x4d, 0x42},

    {0xc5, 0x7c, 0x44},

    {0xff, 0xae, 0xdb},

    {0xb, 0x92, 0x0},

    {0xb9, 0x4e, 0xd4},

    {0x4b, 0xd1, 0x52},

    {0x63, 0x7e, 0x3c},

    {0x8f, 0x10, 0x9b},

    {0x7b, 0xcf, 0x39},

    {0x4d, 0x3d, 0xbd},

    {0xa6, 0xc2, 0x1},

    {0x5f, 0x56, 0xdd},

    {0x22, 0xb9, 0x3d},

    {0xd8, 0x4d, 0xcf},

    {0x0, 0xa0, 0x3b},

    {0xff, 0x75, 0xee},

    {0x30, 0x88, 0x0},

    {0x32, 0x70, 0xfa},

    {0xa8, 0xd5, 0x49},

    {0xa9, 0x78, 0xff},

    {0xc6, 0xce, 0x42},

    {0x0, 0x53, 0xc7},

    {0xdc, 0xa5, 0x0},

    {0x86, 0x7c, 0xff},

    {0xe7, 0xc3, 0x39},

    {0x1, 0x5c, 0xbe},

    {0xff, 0xb7, 0x32},

    {0xce, 0x86, 0xff},

    {0x1, 0xc7, 0x7c},

    {0xdb, 0x1, 0x90},

    {0x4c, 0xdf, 0x95},

    {0xa9, 0x0, 0x8f},

    {0xa6, 0xd5, 0x65},

    {0x8b, 0x21, 0x8d},

    {0x65, 0x86, 0x0},

    {0xd4, 0x0, 0x7f},

    {0x1, 0x88, 0x44},

    {0xb1, 0x0, 0x78},

    {0x0, 0x66, 0x1a},

    {0xfa, 0x9f, 0xff},

    {0x43, 0x73, 0x0},

    {0x63, 0x3f, 0x9a},

    {0xfb, 0x89, 0x19},

    {0x2, 0x78, 0xc9},

    {0xe3, 0x4e, 0x1c},

    {0x1, 0xc9, 0xb9},

    {0xbe, 0x0, 0x14},

    {0x58, 0xd6, 0xec},

    {0xec, 0x3a, 0x37},

    {0x60, 0xd7, 0xda},

    {0xff, 0x3f, 0x54},

    {0x0, 0x8f, 0x65},

    {0xd7, 0x0, 0x66},

    {0xc, 0x60, 0x25},

    {0xff, 0x41, 0x68},

    {0x87, 0xd7, 0xaa},

    {0xbe, 0x0, 0x4f},

    {0xbd, 0xce, 0x77},

    {0x8d, 0x9b, 0xff},

    {0xfd, 0xba, 0x4c},

    {0x80, 0xab, 0xff},

    {0xa8, 0x39, 0x0},

    {0x1, 0x77, 0xb3},

    {0xa9, 0x58, 0x0},

    {0xc8, 0xa3, 0xff},

    {0x80, 0x7f, 0x0},

    {0xff, 0x91, 0xd8},

    {0xf, 0x5f, 0x3f},

    {0xff, 0x61, 0x95},

    {0x45, 0x59, 0x1f},

    {0xff, 0x72, 0xb3},

    {0x97, 0x79, 0x0},

    {0x6c, 0x42, 0x7b},

    {0xe3, 0xc3, 0x6e},

    {0x93, 0x25, 0x69},

    {0xfd, 0xb9, 0x68},

    {0x67, 0x63, 0x95},

    {0x7e, 0x62, 0x0},

    {0x9a, 0x20, 0x51},

    {0xc1, 0xb0, 0x79},

    {0xaf, 0x0, 0x28},

    {0xf1, 0xbc, 0x8b},

    {0x9d, 0x22, 0x30},

    {0x60, 0x52, 0x1b},

    {0xff, 0xb0, 0xb7},

    {0x73, 0x4a, 0x0},

    {0xa4, 0x6e, 0x95},

    {0xff, 0x86, 0x64},

    {0x7a, 0x45, 0x21},

    {0xff, 0x86, 0x83},

    {0x92, 0x33, 0x15},

    {0xff, 0xa7, 0x79},

    {0x8d, 0x35, 0x3d},

    {0xb0, 0x73, 0x5f},

    {0xb2, 0x6b, 0x77}};

Листинг В15 — Код из файла client.cpp

#include <SDL2/SDL.h>

#include <SDL2/SDL\_ttf.h>

#include <stdio.h>

#include <string>

#include <cmath>

#include <sys/types.h>

#include <sys/socket.h>

#include <netinet/in.h>

#include <arpa/inet.h>

#include <errno.h>

#include <unistd.h>

#include <limits.h>

#include <stdlib.h>

#include "LTexture.h"

#include "colors.h"

#include "../shared.h"

const int SCREEN\_WIDTH = 640;

const int MENU\_HEIGHT = 100;

const int GAME\_HEIGHT = 640;

const int SCREEN\_HEIGHT = GAME\_HEIGHT + MENU\_HEIGHT;

const int FPS = 24;

bool init();

void close();

SDL\_Window \*gWindow = NULL;

SDL\_Renderer \*gRenderer = NULL;

TTF\_Font \*gBigFont, \*gMidFont, \*gSmallFont, \*gFoodFont;

LTexture \*gTextTexture;

bool init()

{

  bool success = true;

  if (SDL\_Init(SDL\_INIT\_VIDEO) < 0)

  {

    printf("SDL could not initialize! SDL Error: %s\n", SDL\_GetError());

    success = false;

  }

  else

  {

    if (!SDL\_SetHint(SDL\_HINT\_RENDER\_SCALE\_QUALITY, "1"))

    {

      printf("Warning: Linear texture filtering not enabled!");

    }

    gWindow = SDL\_CreateWindow("gnibbles", SDL\_WINDOWPOS\_UNDEFINED, SDL\_WINDOWPOS\_UNDEFINED, SCREEN\_WIDTH, SCREEN\_HEIGHT, SDL\_WINDOW\_SHOWN);

    if (gWindow == NULL)

    {

      printf("Window could not be created! SDL Error: %s\n", SDL\_GetError());

      success = false;

    }

    else

    {

      gRenderer = SDL\_CreateRenderer(gWindow, -1, SDL\_RENDERER\_ACCELERATED);

      if (gRenderer == NULL)

      {

        printf("Renderer could not be created! SDL Error: %s\n", SDL\_GetError());

        success = false;

      }

      else

      {

        SDL\_SetRenderDrawColor(gRenderer, 0xFF, 0xFF, 0xFF, 0xFF);

      }

    }

  }

  return success;

}

bool loadMedia()

{

  bool success = true;

  gSmallFont = TTF\_OpenFont("font.ttf", 24);

  gMidFont = TTF\_OpenFont("font.ttf", 36);

  gBigFont = TTF\_OpenFont("font.ttf", 60);

  if (gSmallFont == NULL || gMidFont == NULL || gBigFont == NULL)

  {

    printf("Failed to font! SDL\_ttf Error: %s\n", TTF\_GetError());

    success = false;

  }

  return success;

}

void close()

{

  SDL\_DestroyRenderer(gRenderer);

  SDL\_DestroyWindow(gWindow);

  gWindow = NULL;

  gRenderer = NULL;

  SDL\_Quit();

}

struct GameState

{

  char ind;

  char ready;

  char gamePhase;

  char prevGamePhase;

  char mapSize;

  char \*\*map;

  char scoresLen;

  ScoreboardRecord \*scores;

  char direction;

};

int main(int argc, char \*args[])

{

  if (!init() || TTF\_Init() || !loadMedia())

  {

    printf("Failed to initialize!\n");

  }

  else

  {

    gTextTexture = new LTexture();

    gTextTexture->init(gRenderer);

    struct sockaddr\_in servaddr;

    int sockfd = socket(PF\_INET, SOCK\_STREAM, 0);

    char \*sep = strchr(args[1], ':');

    bzero(&servaddr, sizeof(servaddr));

    servaddr.sin\_family = AF\_INET;

    servaddr.sin\_port = htons(atoi(sep + 1));

    sep[0] = 0;

    inet\_aton(args[1], &servaddr.sin\_addr);

    if (connect(sockfd, (struct sockaddr \*)&servaddr, sizeof(servaddr)) < 0)

    {

      printf("Can't connect to server\n");

      return 1;

    }

    GameState gameState;

    gameState.ready = false;

    gameState.gamePhase = WAITING\_FOR\_PLAYERS;

    gameState.prevGamePhase = -1;

    bool quit = false;

    SDL\_Event e;

    char textBuff[128];

    if (!read(sockfd, &gameState.ind, sizeof(gameState.ind)))

    {

      quit = true;

    }

    Uint32 startTime = 0;

    Uint32 endTime = 0;

    Uint32 delta = 0;

    short timePerFrame = 1000 / FPS;

    while (!quit)

    {

      if (!startTime)

      {

        startTime = SDL\_GetTicks();

      }

      else

      {

        delta = endTime - startTime;

      }

      while (SDL\_PollEvent(&e) != 0)

      {

        if (e.type == SDL\_QUIT)

        {

          quit = true;

        }

        else if (e.type == SDL\_KEYDOWN)

        {

          switch (e.key.keysym.sym)

          {

          case SDLK\_q:

            quit = true;

            break;

          case SDLK\_SPACE:

            gameState.ready = !gameState.ready;

            break;

          case SDLK\_UP:

            gameState.direction = 1;

            break;

          case SDLK\_RIGHT:

            gameState.direction = 2;

            break;

          case SDLK\_DOWN:

            gameState.direction = 3;

            break;

          case SDLK\_LEFT:

            gameState.direction = 4;

            break;

          default:

            break;

          }

        }

      }

      SDL\_SetRenderDrawColor(gRenderer, DARK);

      SDL\_RenderClear(gRenderer);

      if (gameState.gamePhase == WAITING\_FOR\_PLAYERS)

      {

        if (gameState.prevGamePhase == SCOREBOARD)

        {

          gameState.ready = false;

          delete[] gameState.scores;

        }

        WaitingState waitingState;

        if (!write(sockfd, &gameState.ready, sizeof(gameState.ready)))

        {

          break;

        }

        if (!read(sockfd, &waitingState, sizeof(waitingState)))

        {

          break;

        }

        gameState.gamePhase = waitingState.gamePhase;

        sprintf(textBuff, "%d of %d", waitingState.playersReady, waitingState.playersConnected);

        gTextTexture->loadFromRenderedText(textBuff, gBigFont, TWHITE);

        gTextTexture->render((SCREEN\_WIDTH - gTextTexture->getWidth()) / 2, (SCREEN\_HEIGHT - gTextTexture->getHeight() \* 2) / 2);

        sprintf(textBuff, "players are ready");

        gTextTexture->loadFromRenderedText(textBuff, gMidFont, TWHITE);

        gTextTexture->render((SCREEN\_WIDTH - gTextTexture->getWidth()) / 2, (SCREEN\_HEIGHT) / 2);

        int textOffset = 5;

        int textYPos = (SCREEN\_HEIGHT + gTextTexture->getHeight()) / 2 + gTextTexture->getHeight() + textOffset;

        if (gameState.ready)

        {

          sprintf(textBuff, "You are ready");

          gTextTexture->loadFromRenderedText(textBuff, gMidFont, TGREEN);

          gTextTexture->render((SCREEN\_WIDTH - gTextTexture->getWidth()) / 2, textYPos);

        }

        else

        {

          sprintf(textBuff, "You are not ready");

          gTextTexture->loadFromRenderedText(textBuff, gMidFont, TRED);

          gTextTexture->render((SCREEN\_WIDTH - gTextTexture->getWidth()) / 2, textYPos);

        }

        sprintf(textBuff, "Press space to change");

        gTextTexture->loadFromRenderedText(textBuff, gSmallFont, TWHITE);

        gTextTexture->render((SCREEN\_WIDTH - gTextTexture->getWidth()) / 2, textYPos + gTextTexture->getHeight() + textOffset);

        gameState.prevGamePhase = WAITING\_FOR\_PLAYERS;

      }

      else if (gameState.gamePhase == IN\_PROGRESS)

      {

        if (gameState.prevGamePhase == WAITING\_FOR\_PLAYERS)

        {

          if (!read(sockfd, &gameState.mapSize, sizeof(gameState.mapSize)))

          {

            break;

          }

          gameState.map = new char \*[gameState.mapSize];

          for (int i = 0; i < gameState.mapSize; i++)

          {

            gameState.map[i] = new char[gameState.mapSize];

          }

          gFoodFont = TTF\_OpenFont("font.ttf", SCREEN\_WIDTH / gameState.mapSize);

          gameState.direction = 0;

        }

        char mapStatus;

        if (!read(sockfd, &mapStatus, sizeof(mapStatus)))

        {

          break;

        }

        for (int i = 0; i < gameState.mapSize; i++)

        {

          if (mapStatus)

          {

            char rowStatus;

            if (!read(sockfd, &rowStatus, sizeof(rowStatus)))

            {

              quit = true;

              break;

            }

            if (rowStatus)

            {

              if (!read(sockfd, gameState.map[i], sizeof(\*gameState.map[i]) \* gameState.mapSize))

              {

                quit = true;

                break;

              }

            }

          }

          char currRow[gameState.mapSize];

          memcpy(currRow, gameState.map[i], sizeof(currRow));

          for (int j = 0; j < gameState.mapSize; j++)

          {

            if (currRow[j])

            {

              if (currRow[j] > 0)

              {

                SDL\_Rect fillRect = {SCREEN\_WIDTH / gameState.mapSize \* i, MENU\_HEIGHT + GAME\_HEIGHT / gameState.mapSize \* j, SCREEN\_WIDTH / gameState.mapSize, GAME\_HEIGHT / gameState.mapSize};

                SDL\_SetRenderDrawColor(gRenderer, MAP\_COLORS[currRow[j] - 1][0], MAP\_COLORS[currRow[j] - 1][1], MAP\_COLORS[currRow[j] - 1][2], 0xFF);

                SDL\_RenderFillRect(gRenderer, &fillRect);

              }

              else

              {

                sprintf(textBuff, "%d", -currRow[j]);

                gTextTexture->loadFromRenderedText(textBuff, gFoodFont, TGREEN);

                gTextTexture->render(SCREEN\_WIDTH / gameState.mapSize \* i + gTextTexture->getWidth() / 2, MENU\_HEIGHT + GAME\_HEIGHT / gameState.mapSize \* j);

              }

            }

          }

        }

        if (quit)

        {

          break;

        }

        InGameState inGameState;

        if (!read(sockfd, &inGameState, sizeof(inGameState)))

        {

          break;

        }

        gameState.gamePhase = inGameState.gamePhase;

        int OFFSET\_H = 10;

        int OFFSET\_W = SCREEN\_WIDTH / gameState.mapSize;

        {

          SDL\_Rect fillRect = {0, 0, SCREEN\_WIDTH, MENU\_HEIGHT};

          SDL\_SetRenderDrawColor(gRenderer, BLACK);

          SDL\_RenderFillRect(gRenderer, &fillRect);

        }

        sprintf(textBuff, "Color");

        gTextTexture->loadFromRenderedText(textBuff, gMidFont, TWHITE);

        int squereSize = MENU\_HEIGHT - gTextTexture->getHeight() - OFFSET\_H \* 3;

        gTextTexture->render(OFFSET\_W, OFFSET\_H);

        {

          SDL\_Rect fillRect = {OFFSET\_W + (gTextTexture->getWidth() - squereSize) / 2, MENU\_HEIGHT - squereSize - OFFSET\_H, squereSize, squereSize};

          SDL\_SetRenderDrawColor(gRenderer, MAP\_COLORS[gameState.ind + 1][0], MAP\_COLORS[gameState.ind + 1][1], MAP\_COLORS[gameState.ind + 1][2], 0xFF);

          SDL\_RenderFillRect(gRenderer, &fillRect);

        }

        int scoreTextPos = (SCREEN\_WIDTH - gTextTexture->getWidth()) / 2;

        sprintf(textBuff, "SCORE");

        gTextTexture->loadFromRenderedText(textBuff, gMidFont, TWHITE);

        gTextTexture->render(scoreTextPos, OFFSET\_H);

        int pTextHeight = gTextTexture->getHeight();

        int pTextWidth = gTextTexture->getWidth();

        sprintf(textBuff, "%d", inGameState.score);

        gTextTexture->loadFromRenderedText(textBuff, gMidFont, TWHITE);

        gTextTexture->render(scoreTextPos + (pTextWidth - gTextTexture->getWidth()) / 2, pTextHeight + (MENU\_HEIGHT - pTextHeight - gTextTexture->getHeight()) / 2);

        sprintf(textBuff, "TURN");

        gTextTexture->loadFromRenderedText(textBuff, gMidFont, TWHITE);

        gTextTexture->render(SCREEN\_WIDTH - OFFSET\_W - gTextTexture->getWidth(), OFFSET\_H);

        {

          SDL\_Rect fillRect = {SCREEN\_WIDTH - (OFFSET\_W + (gTextTexture->getWidth() + squereSize) / 2), MENU\_HEIGHT - squereSize - OFFSET\_H, squereSize, squereSize};

          SDL\_SetRenderDrawColor(gRenderer, MAP\_COLORS[inGameState.activeWorm + 1][0], MAP\_COLORS[inGameState.activeWorm + 1][1], MAP\_COLORS[inGameState.activeWorm + 1][2], 0xFF);

          SDL\_RenderFillRect(gRenderer, &fillRect);

        }

        if (!write(sockfd, &gameState.direction, sizeof(gameState.direction)))

        {

          break;

        }

        gameState.direction = 0;

        gameState.prevGamePhase = IN\_PROGRESS;

      }

      else if (gameState.gamePhase == SCOREBOARD)

      {

        ScoreboardState scoreboardState;

        if (!read(sockfd, &scoreboardState, sizeof(ScoreboardState)))

        {

          break;

        }

        gameState.gamePhase = scoreboardState.gamePhase;

        if (gameState.prevGamePhase == IN\_PROGRESS)

        {

          for (int i = 0; i < gameState.mapSize; i++)

          {

            delete[] gameState.map[i];

          }

          delete[] gameState.map;

          gameState.scoresLen = scoreboardState.players;

          gameState.scores = new ScoreboardRecord[gameState.scoresLen];

          for (int i = 0; i < gameState.scoresLen; i++)

          {

            if (!read(sockfd, &gameState.scores[i], sizeof(gameState.scores[i])))

            {

              quit = true;

              break;

            }

          }

          if (quit)

          {

            break;

          }

        }

        int TOP\_OFFSET = 90;

        int OFFSET = 10;

        sprintf(textBuff, "SCOREBOARD");

        gTextTexture->loadFromRenderedText(textBuff, gBigFont, TWHITE);

        gTextTexture->render((SCREEN\_WIDTH - gTextTexture->getWidth()) / 2, TOP\_OFFSET);

        int recordsStart = TOP\_OFFSET + gTextTexture->getHeight() + OFFSET;

        for (int i = 0; i < gameState.scoresLen; i++)

        {

          ScoreboardRecord record = gameState.scores[i];

          sprintf(textBuff, "%d", record.score);

          SDL\_Color color = {MAP\_COLORS[record.ind + 1][0], MAP\_COLORS[record.ind + 1][1], MAP\_COLORS[record.ind + 1][2]};

          gTextTexture->loadFromRenderedText(textBuff, gBigFont, color);

          gTextTexture->render((SCREEN\_WIDTH - gTextTexture->getWidth()) / 2, recordsStart + i \* gTextTexture->getHeight());

        }

        char sync = 1;

        if (!write(sockfd, &sync, sizeof(sync)))

        {

          break;

        }

        gameState.prevGamePhase = SCOREBOARD;

      }

      else

      {

        break;

      }

      SDL\_RenderPresent(gRenderer);

      if (delta < timePerFrame)

      {

        SDL\_Delay(timePerFrame - delta);

      }

      startTime = endTime;

      endTime = SDL\_GetTicks();

    }

  }

  close();

  return 0;

}