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

Кудряшов И. С. \_\_\_\_\_\_

«\_\_».\_\_\_\_\_.2020г.

Проверил:

Ананьев В. В. \_\_\_\_\_\_

«\_\_».\_\_\_\_\_.2020г.

**Великий Новгород**

**2020**

1. ОГЛАВЛЕНИЕ

[ОГЛАВЛЕНИЕ 2](#_Toc42984743)

[ВВЕДЕНИЕ 3](#_Toc42984744)

[ГЛАВА 1. РАЗРАБОТКА ПРОЕКТА. 4](#_Toc42984745)

[1.1 Постановка задачи. 4](#_Toc42984746)

[1.2 Выбор инструментальных средств. 4](#_Toc42984747)

[1.3 Выбор модели. 4](#_Toc42984748)

[1.4 Выбор способа решения проблемы блокирования ввода/вывода. 4](#_Toc42984749)

[1.5 Протокол взаимодействия клиента и сервера. 6](#_Toc42984750)

[1.6 Выбор протокола транспортного уровня OSI-модели. 6](#_Toc42984751)

[1.7 Алгоритм решения задачи. 7](#_Toc42984752)

[1.8 Вывод по первой главе. 7](#_Toc42984753)

[ГЛАВА 2. РЕАЛИЗАЦИЯ ПРОЕКТА. 8](#_Toc42984754)

[2.1 Структура клиента и сервера. 8](#_Toc42984755)

[2.2 Программный интерфейс сервера. 8](#_Toc42984756)

[2.3 Описание программной реализации клиента 10](#_Toc42984757)

[2.4 Системные вызовы. 11](#_Toc42984758)

[2.5 Пример работы приложения. 12](#_Toc42984759)

[2.6 Результат работы. 12](#_Toc42984760)

[ЗАКЛЮЧЕНИЕ 13](#_Toc42984761)

[СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ. 13](#_Toc42984762)

[ПРИЛОЖЕНИЕ А 14](#_Toc42984763)

[ПРИЛОЖЕНИЕ Б 17](#_Toc42984764)

[Листинг В1 — Код из файла server.cpp 17](#_Toc42984765)

[Листинг В2 — Код из файла client.cpp 31](#_Toc42984766)

[Листинг В3 — Код из файла common\_types.hpp 41](#_Toc42984767)

[Листинг В4 — Файл map.txt 42](#_Toc42984768)

[Листинг В5 —Файл player\_collors.txt 42](#_Toc42984769)

1. ВВЕДЕНИЕ

Целью данной работы является создание многопользовательской игры, в которой задачей игроков является сбор максимального числа монет в лабиринте.

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

Основной функционал игры и правила игры следующие:

Число игроков в одной игре зависит от карты, в приложенной в примере карте равно 4. При подключении к игре игрок занимает одну из стартовых позиций, заданных в файле карты. С помощью клавиш ← ↑ → ↓ игроки могут передвигаться по полю. При попадании в ловушку игрок оказывается на точке своего появления в игре, а ловушка становится видима для всех игроков. В игре есть двери, их открытие/закрытие осуществляется на клавишу R. Цель игры собрать больше монет чем соперники. Местоположение монет на карте случайно. Так как часть монет может оказаться недосягаемой из-за ловушек для завершения игры необходимо собрать 4/5 монет. По завершению игры счёт выводится в командной строке клиентов, игроки помещаются на стартовые позиции, все ловушки снова становятся невидимыми, а двери закрытыми. Монеты раскидываются по карте в новом случайном порядке. Игру можно досрочно завершить командой restart в консоли сервера.

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 проектирование сетевой игры на несколько человек с механикой сбора монет в лабиринте. Игра должна работать под управлением операционной системы семейства Unix/Linux. Сетевая часть игры 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 Linux Mint 19.3, запущенный в виртуальной машине. Ядро версии 5.3.

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 результатов и перемещении игроков по полю 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. Отправка данных происходят в том же потоке, что и отрисовка кадра, поскольку этот поток обрабатывает события. Получение данных происходит в отдельном потоке, синхронизация происходит с использованием мьютексов.

Серверная часть моей программы идёт отдельно от клиентской, главный поток обрабатывает текстовые команды пользователя, так же он создаёт поток что загружает карту и принимает новые подключения от клиентов. На каждого клиента создаётся по 2 потока, один из которых принимает действия клиента и вносит их на карту, второй же отправляет актуальную версию карты игрокам. Синхронизация обеспечивается мьютексами.

1.5 Протокол взаимодействия клиента и сервера.

Типы данных описаны в заголовочном файле, общем для клиента и сервера. Перед отправкой данных отправляется структура, в которой указывается тип и размер отправляемых данных. Для данных малого размера поле размера в структуре используется для передачи данных и дополнительные пакеты не отправляются.

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 в строго определённом порядке.

1.7 Алгоритм решения задачи.

При подсоединении игрока сервер выбирает пустую стартовую позицию, если число игроков достигло лимита, то соединение закрывается и вслед за ним закрывается клиент превысивший лимит игроков.  
При совершении действий клиент отправляет сообщение серверу о том, что пытается сделать игрок. Сервер проверяет возможно ли совершить это действие и если да, то вносит изменение в карту. Поток сервера, отслеживающий изменения, обнаруживает, что время последней отправленной карты не совпадает с временем текущей копии и отправляет её клиенту, обновляя время последней отправленной карты на текущее.  
Когда общее число монет, собранных игроками, достигает 4/5 игра завершается, сервер отправляет клиентам топ по собранным монетам и приводит карту к изначальному состоянию но с другим местоположением монет.  
Игра завершается при завершении клиента.

1.8 Вывод по первой главе.

Таким образом, для реализации игры «Лабиринт» основной ОС выбран Linux Mint 19.3, языком программирования процедурной части 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.

Функционал сервера располагается в файле server.cpp  
Функционал клиента в файле client.cpp

Для хранения типов, общих для сервера и клиента, используется заголовочный файл common\_types.hpp

Цвета для игроков, которые сервер считывает при каждом запуске, находятся в файле player\_collors.txt

Карта, которую сервер считывает при запуске, находится в файле map.txt

2.2 Программный интерфейс сервера.

**Описание работы главного потока:**

Главный поток преобразует данный ему порт к целочисленному виду, затем запускает поток, который обрабатывает подключение новых игроков, сам ожидая команды из консоли.

**Описание работы потока ожидания новых клиентов:**

Считывает карту и цвета игроков, создаёт объекты для хранения данных о игре и для синхронизации потоков. После чего наполняет карту монетами и передаёт указатели на данные основному потоку для обеспечения возможности управления игрой из консоли. Далее запускается бесконечный цикл, в котором этот поток принимает новые соединения и при наличии свободных мест создаёт поток для обработки команд клиента и передаёт ему структуру с данными.

**Описание работы потока приёма команд игрока:**

После старта выбирает свободную ячейку для игрока и обновляет список игроков. Затем создаёт поток, который отслеживает изменения в карте и списке игроков и при необходимости отправляет их клиенту.  
Затем запускается процесс обработки команд от клиента.   
В начале проверяется правильные ли координаты своего местонахождения прислал клиент. Дальнейшее происходит в зависимости от типа действия.

Передвижение  
Если точка назначения за пределами игрового поля проверка немедленно прекращается.  
Если передвижение происходит в пустую ячейку или дверь, то происходит проверка не занята ли эта ячейка другим игроком и если она свободна, то игрок перемещается туда. Затем устанавливается новое время изменения для списка игроков чтобы новая версия была разослана клиентам.  
Если передвижение происходит в ловушку, то ловушка становится видимой, карта обновляется чтобы все клиенты получили версию карты с видимой ловушкой.   
Затем для скрытой ловушки, и для её видимой версии происходит перемещение игрока на точку появления в игре.  
Если передвижение происходит в точку с монетой то число собранных игроком монет увеличивается на 1, также увеличивается общее число собранных монет. Если общее число собранных монет превысило 4/5 от числа размещённых монет, то игроки помещаются на стартовые позиции, топ игроков сохраняется и посылается сигнал что его необходимо разослать клиентам. Затем карта очищается и заново заполняется монетами.  
Если число монет недостаточно для завершения игры то игрок переносится в новое местоположение, а место на карте где была монета заменяется пустотой.  
В случае действия, направленного на открытие/закрытие двери проверяется пуста ли эта клетка от игрока и затем есть ли там дверь. Если условия соблюдены, то дверь меняет своё состояние на противоположное и карта обновляется.

Цикл продолжается до тех пор, пока recv не вернёт 0 что означает разрыв соединения с клиентом.  
По завершению цикла закрывается поток отправляющий клиенту изменения, игрок помечается как неактивный, число подключенных игроков уменьшается, сокет закрывается, структура данных переданная для потоков удаляется.

**Описание работы потока отслеживания изменений:**

По умолчанию переменные, хранящие в себе время последней отправленной версии, принимаются равными 0 для того, чтобы клиент сразу получил карту и список игроков. Время последнего отправленного топа игроков принимается равным текущему для того, чтобы топ был отправлен только по завершению игры. Затем отправляется первое сообщение, содержащее номер игрока. Далее запускается цикл, в котором последовательно проверяется последняя ли версия карты, списка игроков и топа есть у клиента. Если время не совпадает, то сперва сохраняется время новой версии, затем отправляется подготовительное сообщение, содержащее тип и размер отправляемых данных, и в конце сами данные.   
Цикл завершается если при попытке отправить данные send возвращает 0 что означает потерю соединения. Для того чтобы посылка в закрытый сокет не вызывала сигнал системы используется параметр send MSG\_NOSIGNAL.

2.3 Описание программной реализации клиента

**Описание работы главного потока:**

После запуска порт и ip адрес отделяются друг от друга и с их помощью производится подключение к серверу. Также создаются переменные для хранения игрового поля и списка игроков. Далее запускается поток, получающий от сервера изменения карты, ему передаются необходимые данные. Затем главный поток ожидает пока не будут получены поле и список игроков от сервера. После их получения происходит загрузка ресурсов и инициализация SDL2. Далее пока не получен сигнал на завершение (в виде флага) или соединение не потеряно:  
Проверяется жив ли поток, получающий изменения от сервера. Если нет соединение считается потерянным. Затем проверяется наличие ивентов SDL2.  
При попытке закрыть окно происходит установка флага, означающего завершение игры.  
При изменении размера окна устанавливается флаг что необходимо заново отрисовать игровое поле.  
При нажатии клавиши передвижение на сервер отправляется текущее местоположение игрока и то куда он хочет перейти.  
При нажатии R для открытия двери проверяется наличие рядом с игроком дверей и для первой найденной двери отправляется событие на сервер.

Далее проверяется обновлялась ли карта. Если карта обновлялась или изменялся размер окна, то поле переотрисовывается.  
Получаются текущие размеры окна. По ним и размеру поля отрисовываются границы клеток. Затем отрисовываются элементы карты. Затем игроки по их местоположению.

**Описание работы потока получения изменений от сервера:**

В начале получается одно предварительное сообщение. Затем пока не разорванно соединение в зависимости от типа сообщения:

Для игрового поля проверяется верны ли текущие размеры поля и если нет оно пересоздаётся с присланными размерами. Затем построчно получается само поле.  
Для списка игроков проверяются его размеры, при необходимости приводятся к нужным и получается сам список.  
Для номера номер из предварительного сообщения сохраняется как номер игрока.  
Для счёта создаётся список нужной длины, в него сохраняется игровой счёт. Счёт выводится в консоль и затем список удаляется.

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. |
| recv() и send() | Используются для отправки и получения данных через сокеты. В отличии от read() и write() могут принимать дополнительные параметры. |
| 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. Имеют 4 аргумента: сокет из которого происходит чтение, источник либо приёмник данных, длина данных в байтах и флаги. |
| 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. |

2.5 Пример работы приложения.

В начале игроки подключаются и оказываются в стартовых позициях. Так как средства подтверждения готовности к игре не предусмотрено игроки собственными силами должны вести себя честно.

Затем игроки перемещаются по полю стараясь как можно быстрее собирать монеты. При попадании в ловушку они оказываются на стартовой позиции.  
По завершению игры выводится счёт, а игроки снова оказываются на стартовых позициях, местоположение монет так же меняется.

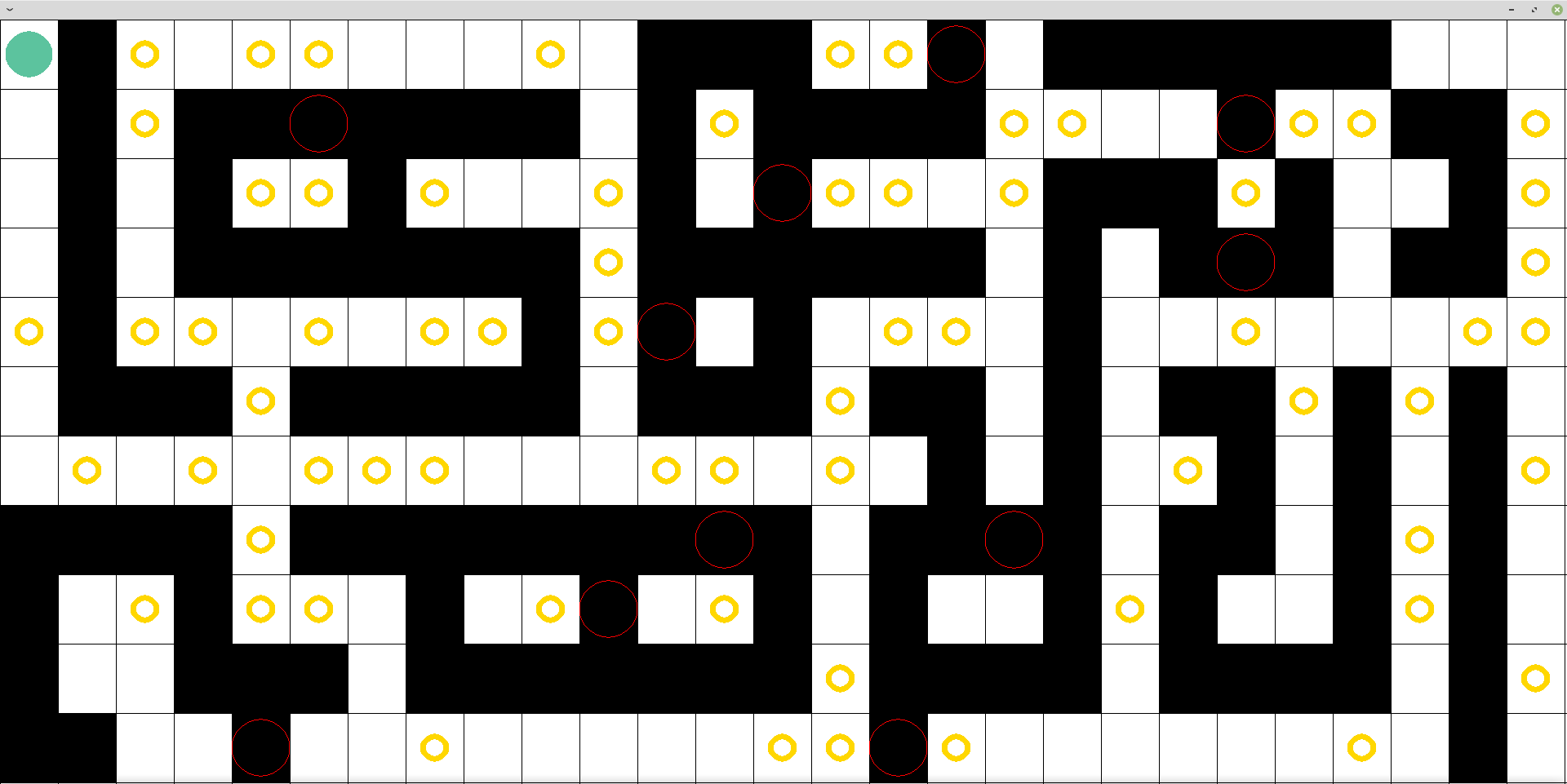
2.6 Результат работы.

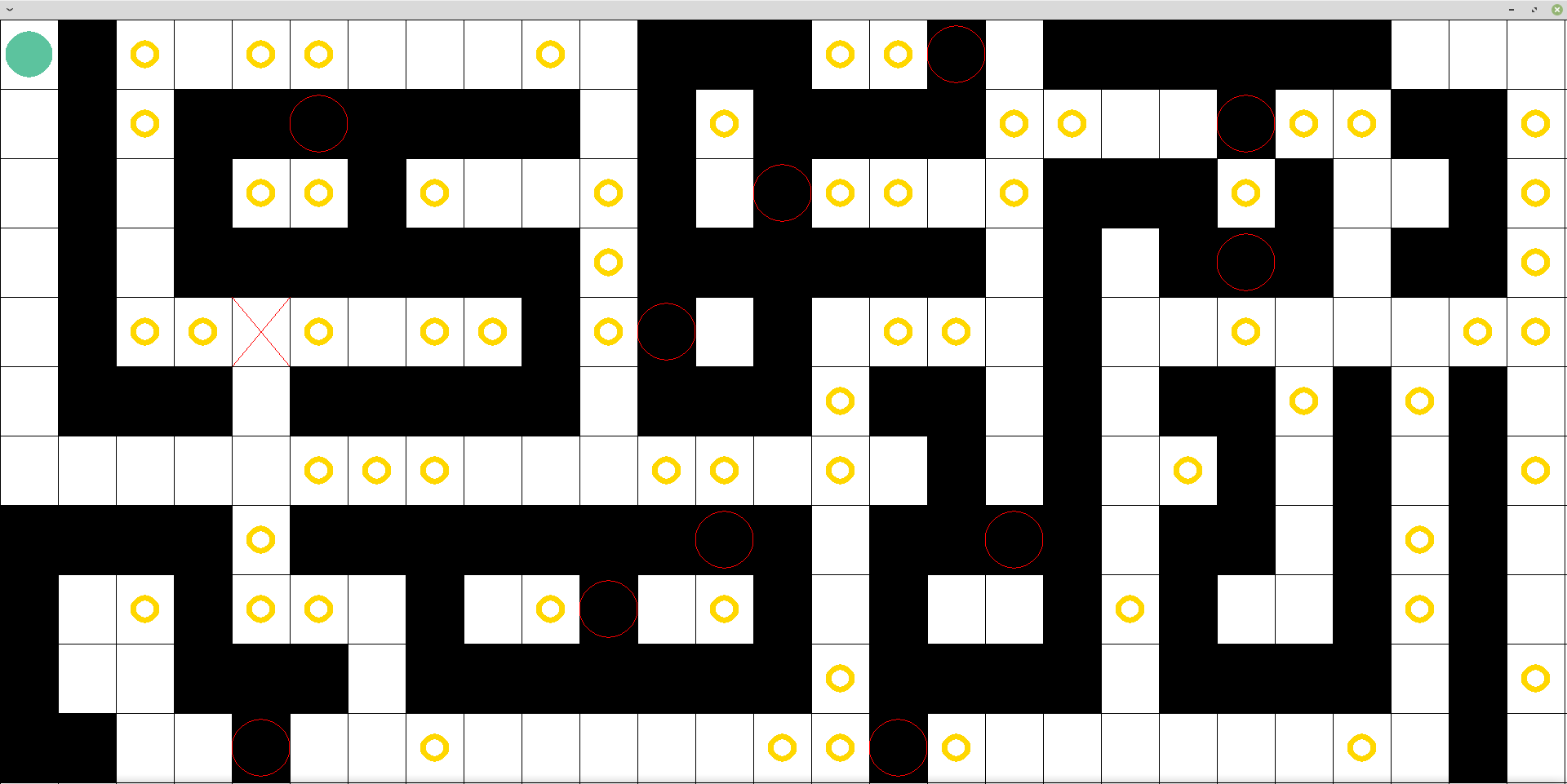
В ходе выполнения данной курсовой работы удалось разработать сетевую пошаговую игру “Лабиринт”, соответствующую поставленным к ней требованиям. Для обеспечения быстрой работы со множеством клиентов было использовано создание двух потоков для каждого игрока. Сервер и клиент были реализованные как отдельные приложения. Разработанный сервер удалось запустить с 4 игроками, но число игроков ограничено скорее наличием подходящей карты. 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.

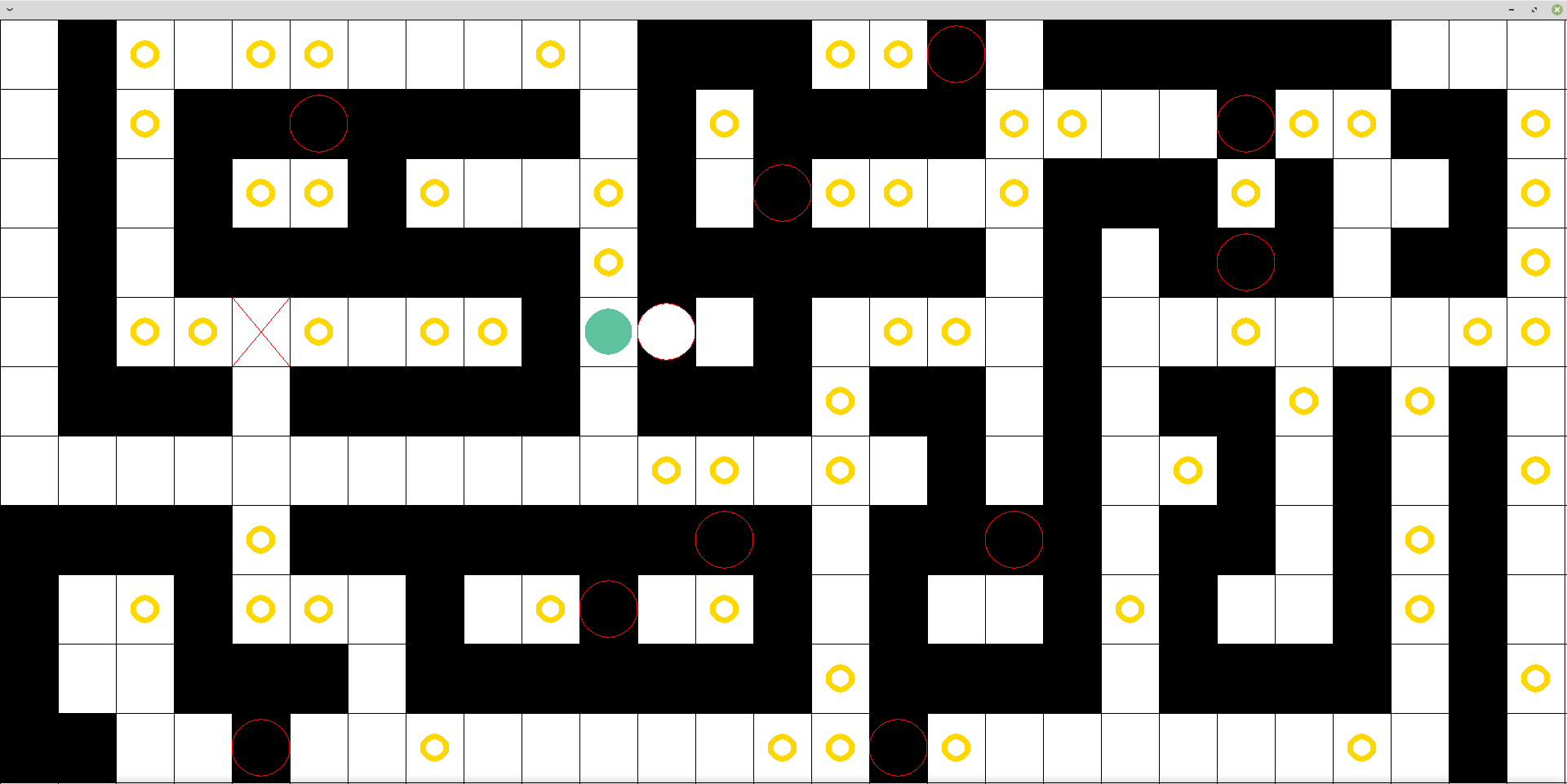
1. ЗАКЛЮЧЕНИЕ

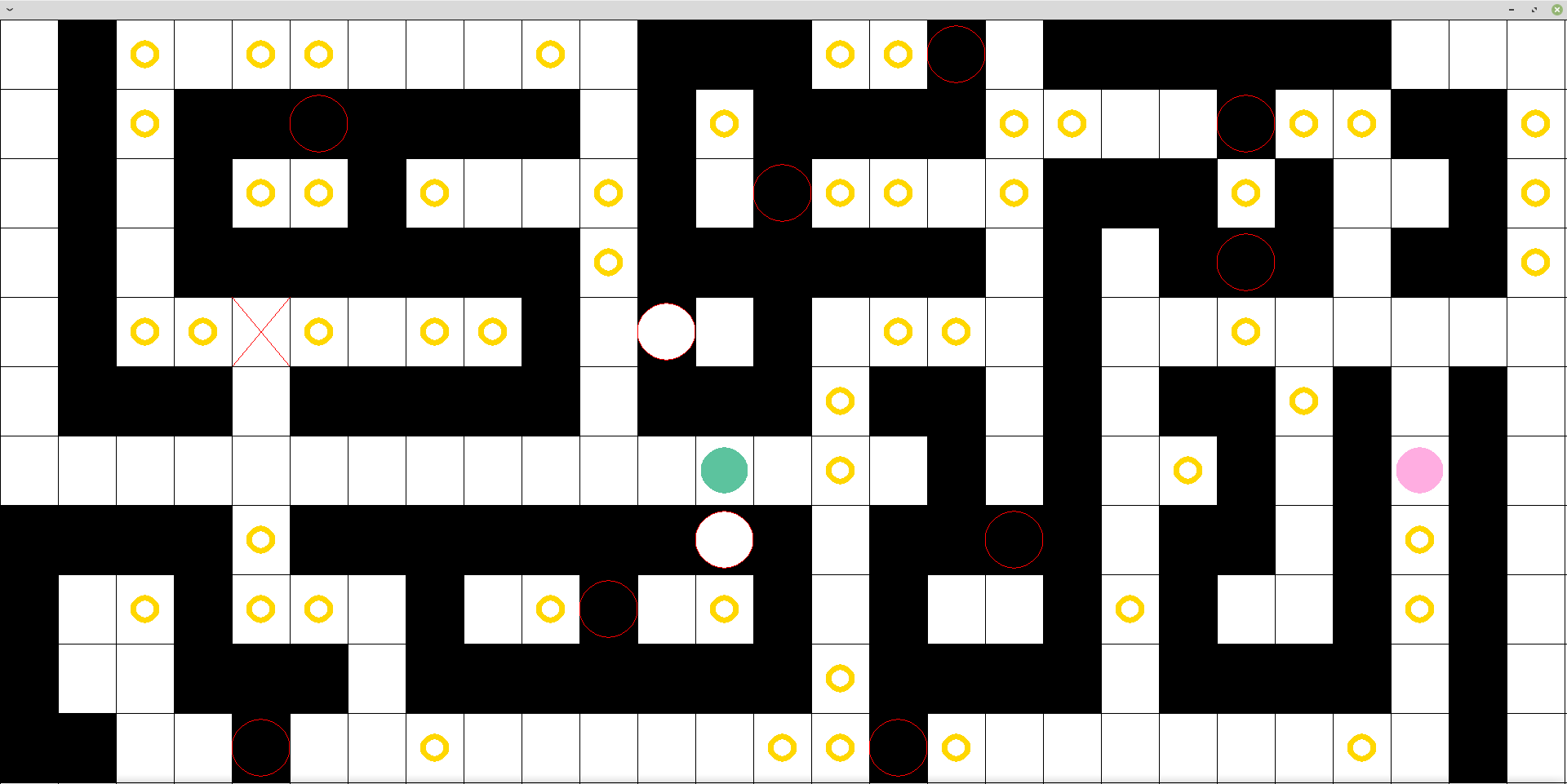
Таким образом, результат курсовой работы соответствует требованиям к реализации. Приложение работает на операционных системах семейства Linux. Для построения серверной и клиентской части используется язык C++ и компилятор g++. Сетевое взаимодействие было построено с помощью протокола TCP. Сервер может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

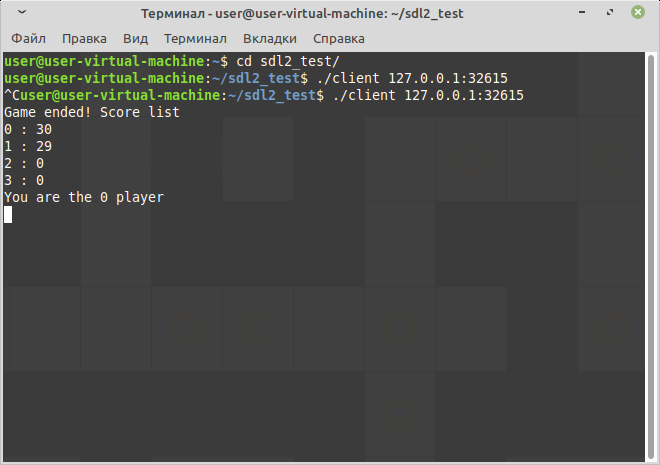
1. СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ.
2. Карпов В. Е., Коньков К. А. «Основы операционных систем. Курс лекций» - Учебное пособие, 2005 - 536с.
3. НОУ ИНТУИТ | Учебный курс | Основы операционных систем: [Электронный ресурс]. URL: http://www.intuit.ru.
4. Проект OpenNet: MAN (FreeBSD и Linux): [Электронный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

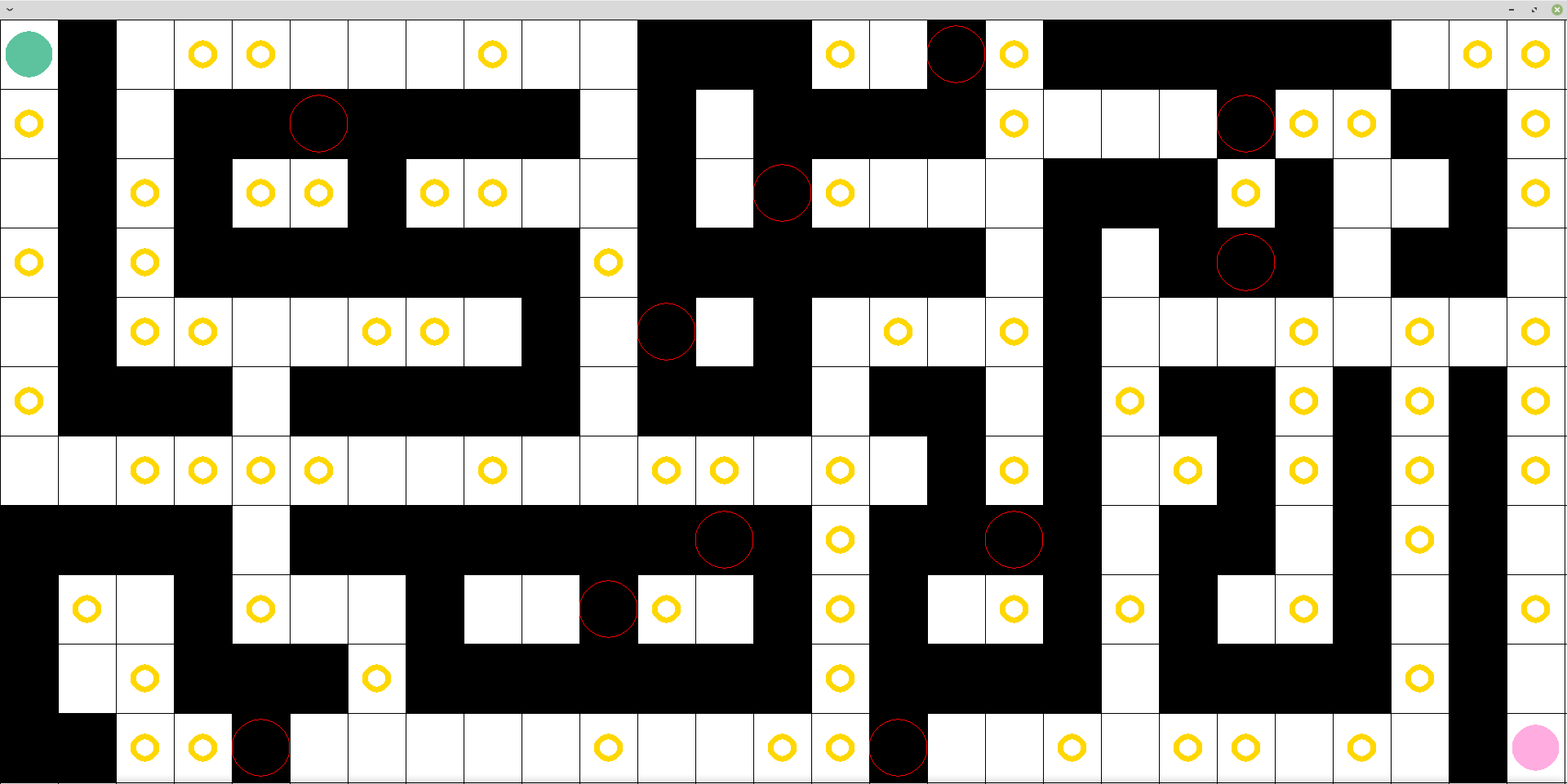












1. ПРИЛОЖЕНИЕ Б

Листинг В1 — Код из файла server.cpp

#include <sys/socket.h>

#include <sys/time.h>

#include <netinet/in.h>

#include <strings.h>

#include <pthread.h>

#include <unistd.h>

#include <string.h>

#include <iostream>

#include <string>

#include <vector>

#include <fstream>

#include "common\_types.hpp"

struct thread\_data

{

    pthread\_mutex\_t \*time\_map\_mutex;

    struct timeval \*update\_map\_time;

    pthread\_mutex\_t \*time\_player\_mutex;

    struct timeval \*update\_player\_time;

    pthread\_mutex\_t \*map\_mutex;

    std::vector<std::vector<field\_cells\_type>> \*map\_s;

    pthread\_mutex\_t \*player\_mutex;

    std::vector<player> \*player\_list;

    pthread\_mutex\_t \*player\_count\_mutex;

    size\_t \*player\_count\_connected;

    std::vector<std::pair<size\_t, size\_t>> \*start\_points;

    pthread\_mutex\_t \*players\_top\_mutex;

    std::vector<top\_unit\_count> \*players\_top, \*game\_result;

    pthread\_mutex\_t \*top\_time\_mutex;

    struct timeval \*players\_top\_time;

    top\_unit\_count \*coin\_count\_collected;

    top\_unit\_count \*coint\_count\_max;

    int sockfd;

    size\_t my\_id;

};

void \*client\_sender(void \*data)

{

    thread\_data &prop = \*((thread\_data \*)data); //присваивание в ссылку для удобства

    //структура с временем последней отправленной клиенту версии

    struct timeval last\_time\_map, last\_time\_player, last\_top\_time;

    last\_time\_map.tv\_sec = last\_time\_map.tv\_usec = 0; //равна 0 для того чтобы по старту в первый раз произошла отправка

    last\_time\_player = last\_time\_map;

    pthread\_mutex\_lock(prop.top\_time\_mutex);

    last\_top\_time = \*prop.players\_top\_time;

    pthread\_mutex\_unlock(prop.top\_time\_mutex);

    prepare\_message\_data\_send prep\_m;  //подготовительное сообщение с типом посылаемых данных

    prep\_m.type = my\_number\_from\_list; //отправка номера этого клиента

    prep\_m.size = prop.my\_id;

    int n = send(prop.sockfd, (prepare\_message\_data\_send \*)&prep\_m, sizeof(prepare\_message\_data\_send), MSG\_NOSIGNAL);

    while (n)

    {

        pthread\_mutex\_lock(prop.time\_map\_mutex);                           //проверка обновления времени

        if (memcmp(prop.update\_map\_time, &last\_time\_map, sizeof(timeval))) //если карта обновилась

        {

            last\_time\_map = \*prop.update\_map\_time; //обновление времени

            pthread\_mutex\_unlock(prop.time\_map\_mutex);

            prep\_m.type = field\_type;

            pthread\_mutex\_lock(prop.map\_mutex);

            prep\_m.size = prop.map\_s->size();

            prep\_m.second\_size = prop.map\_s->at(0).size();

            n = send(prop.sockfd, (prepare\_message\_data\_send \*)&prep\_m, sizeof(prepare\_message\_data\_send), MSG\_NOSIGNAL);

            for (size\_t i = 0; i < prop.map\_s->size(); i++) //отправка поля построчно

                n = send(prop.sockfd, (field\_cells\_type \*)prop.map\_s->at(i).data(), prop.map\_s->at(i).size() \* sizeof(field\_cells\_type), MSG\_NOSIGNAL);

            pthread\_mutex\_unlock(prop.map\_mutex);

        }

        else

            pthread\_mutex\_unlock(prop.time\_map\_mutex);

        pthread\_mutex\_lock(prop.time\_player\_mutex);

        if (memcmp(prop.update\_player\_time, &last\_time\_player, sizeof(timeval))) //если список игроков обновился

        {

            last\_time\_player = \*prop.update\_player\_time;

            pthread\_mutex\_unlock(prop.time\_player\_mutex);

            prep\_m.type = player\_list;

            pthread\_mutex\_lock(prop.player\_mutex);

            prep\_m.size = prop.player\_list->size();

            n = send(prop.sockfd, (prepare\_message\_data\_send \*)&prep\_m, sizeof(prepare\_message\_data\_send), MSG\_NOSIGNAL);

            n = send(prop.sockfd, (player \*)prop.player\_list->data(), prop.player\_list->size() \* sizeof(player), MSG\_NOSIGNAL);

            pthread\_mutex\_unlock(prop.player\_mutex);

        }

        else

            pthread\_mutex\_unlock(prop.time\_player\_mutex);

        pthread\_mutex\_lock(prop.top\_time\_mutex);

        if (memcmp(prop.players\_top\_time, &last\_top\_time, sizeof(timeval)))

        {

            last\_top\_time = \*prop.players\_top\_time;

            pthread\_mutex\_unlock(prop.top\_time\_mutex);

            prep\_m.type = score;

            prep\_m.size = prop.game\_result->size();

            n = send(prop.sockfd, (prepare\_message\_data\_send \*)&prep\_m, sizeof(prepare\_message\_data\_send), MSG\_NOSIGNAL);

            n = send(prop.sockfd, (top\_unit\_count \*)prop.game\_result->data(), prop.game\_result->size() \* sizeof(top\_unit\_count), MSG\_NOSIGNAL);

        }

        else

            pthread\_mutex\_unlock(prop.top\_time\_mutex);

        usleep(1);

    }

    return (void \*)(0);

}

void map\_renew(std::vector<std::vector<field\_cells\_type>> &map\_s)

{

    for (auto &i : map\_s)

        for (auto &j : i)

            switch (j)

            {

            case coin:

                j = empty;

                break;

            case door\_open:

                j = door\_lock;

                break;

            case trap\_on:

                j = trap;

                break;

            default:

                break;

            }

}

top\_unit\_count coin\_spawn(std::vector<std::vector<field\_cells\_type>> &map\_s, std::vector<std::pair<size\_t, size\_t>> &start\_points)

{

    top\_unit\_count count\_of\_coins = 0;

    srand(time(NULL));

    for (size\_t i = 0; i < map\_s.size(); i++)

        for (size\_t j = 0; j < map\_s[i].size(); j++)

            if (map\_s[i][j] == empty && rand() % 2)

            {

                bool is\_free = true;

                for (auto &k : start\_points)

                    if (j == k.second && i == k.first)

                    {

                        is\_free = false;

                        break;

                    }

                if (is\_free)

                {

                    map\_s[i][j] = coin;

                    count\_of\_coins++;

                }

            }

    return count\_of\_coins;

}

bool is\_field\_free\_player(size\_t x, size\_t y, size\_t my\_id, std::vector<player> &player\_list) //свободно ли поле от другого игрока

{

    bool is\_free = true;

    for (size\_t k = 0; k < player\_list.size() && is\_free; k++)                                                   //проверка не занята ли клетка одним из игроков

        is\_free = (k == my\_id) || !player\_list[k].is\_alive || !(player\_list[k].x == x && player\_list[k].y == y); //занята самим игроком или проверяемый не активен или координаты не совпадают

    return is\_free;

}

void \*client\_reciver(void \*data)

{

    thread\_data &prop = \*((thread\_data \*)data); //присваивание в ссылку для удобства

    //поиск свободной ячейки для игрока

    pthread\_mutex\_lock(prop.player\_mutex);

    for (size\_t k = 0; k < prop.player\_list->size(); k++)

        if (!prop.player\_list->at(k).is\_alive)

        {

            prop.my\_id = prop.player\_list->at(k).id;

            prop.player\_list->at(prop.my\_id).x = prop.start\_points->at(prop.my\_id).second;

            prop.player\_list->at(prop.my\_id).y = prop.start\_points->at(prop.my\_id).first;

            prop.player\_list->at(prop.my\_id).is\_alive = true;

            break;

        }

    pthread\_mutex\_unlock(prop.player\_mutex);

    pthread\_mutex\_lock(prop.time\_player\_mutex); //время меняется на каждое изменение

    gettimeofday(prop.update\_player\_time, NULL);

    pthread\_mutex\_unlock(prop.time\_player\_mutex);

    pthread\_t sender\_start; //запуск потока отслеживающего изменения поля и отсылающего клиенту

    pthread\_create(&sender\_start, NULL, client\_sender, (void \*)&prop);

    action\_send input\_act;                                                        //переменная для получения действий игрока

    int n = recv(prop.sockfd, (action\_send \*)&input\_act, sizeof(action\_send), 0); //получение первого действи

    while (n)                                                                     //если n==0 соединение разорвано

    {

        pthread\_mutex\_lock(prop.player\_mutex);

        if (input\_act.from\_x == prop.player\_list->at(prop.my\_id).x && input\_act.from\_y == prop.player\_list->at(prop.my\_id).y) //правильно ли текущее местоположение игрока что знает клиент

        {

            pthread\_mutex\_unlock(prop.player\_mutex);

            switch (input\_act.action) //для возможности добавления новых действий

            {

            case move:

                if (!(input\_act.to\_x < prop.map\_s->at(0).size() && input\_act.to\_y < prop.map\_s->size())) //не выходит ли точка назначения за пределы карты

                    break;

                pthread\_mutex\_lock(prop.map\_mutex);

                switch (prop.map\_s->at(input\_act.to\_y)[input\_act.to\_x]) //в зависимости от того в какую точку идёт игрок

                {

                case empty:

                case door\_open:

                    pthread\_mutex\_lock(prop.player\_mutex);

                    if (is\_field\_free\_player(input\_act.to\_x, input\_act.to\_y, prop.my\_id, \*prop.player\_list)) //если поле свободно от другого игрока

                    {

                        prop.player\_list->at(prop.my\_id).x = input\_act.to\_x;

                        prop.player\_list->at(prop.my\_id).y = input\_act.to\_y;

                        pthread\_mutex\_lock(prop.time\_player\_mutex);

                        gettimeofday(prop.update\_player\_time, NULL);

                        pthread\_mutex\_unlock(prop.time\_player\_mutex);

                    }

                    pthread\_mutex\_unlock(prop.player\_mutex);

                    break;

                case trap:

                    prop.map\_s->at(input\_act.to\_y)[input\_act.to\_x] = trap\_on;

                    pthread\_mutex\_lock(prop.time\_map\_mutex);

                    gettimeofday(prop.update\_map\_time, NULL);

                    pthread\_mutex\_unlock(prop.time\_map\_mutex);

                case trap\_on:

                    pthread\_mutex\_lock(prop.player\_mutex);

                    prop.player\_list->at(prop.my\_id).x = prop.start\_points->at(prop.my\_id).second;

                    prop.player\_list->at(prop.my\_id).y = prop.start\_points->at(prop.my\_id).first;

                    pthread\_mutex\_unlock(prop.player\_mutex);

                    pthread\_mutex\_lock(prop.time\_player\_mutex);

                    gettimeofday(prop.update\_player\_time, NULL);

                    pthread\_mutex\_unlock(prop.time\_player\_mutex);

                    break;

                case coin:

                    pthread\_mutex\_lock(prop.players\_top\_mutex);

                    prop.players\_top->at(prop.my\_id)++;                               // добавление очков игроку

                    (\*prop.coin\_count\_collected)++;                                   //сколько монет всего собрано

                    if (\*prop.coin\_count\_collected > (\*prop.coint\_count\_max \* 4) / 5) //собрано ли 4/5 монет(на случай если часть в недоступной зоне карты)

                    {

                        pthread\_mutex\_lock(prop.player\_mutex);

                        for (size\_t i = 0; i < prop.start\_points->size(); i++) //отправка игроков на стартовые позиции

                            if (prop.player\_list->at(i).is\_alive)

                            {

                                prop.player\_list->at(i).x = prop.start\_points->at(i).second;

                                prop.player\_list->at(i).y = prop.start\_points->at(i).first;

                            }

                        pthread\_mutex\_unlock(prop.player\_mutex);

                        \*prop.game\_result = \*prop.players\_top;                                         //сохранение результата ишры

                        \*prop.players\_top = std::vector<top\_unit\_count>(prop.start\_points->size(), 0); //очистка топа

                        pthread\_mutex\_lock(prop.top\_time\_mutex);

                        gettimeofday(prop.players\_top\_time, NULL); //обновление времени для инициализации процедуры отправки топа

                        pthread\_mutex\_unlock(prop.top\_time\_mutex);

                        map\_renew(\*prop.map\_s); //очистка карты

                        \*prop.coin\_count\_collected = 0;

                        \*prop.coint\_count\_max = coin\_spawn(\*prop.map\_s, \*prop.start\_points); //создание новых монет

                    }

                    else

                    {

                        pthread\_mutex\_lock(prop.player\_mutex);

                        prop.player\_list->at(prop.my\_id).x = input\_act.to\_x;

                        prop.player\_list->at(prop.my\_id).y = input\_act.to\_y;

                        pthread\_mutex\_unlock(prop.player\_mutex);

                        prop.map\_s->at(input\_act.to\_y)[input\_act.to\_x] = empty; //очистка ячейки от монеты

                    }

                    pthread\_mutex\_unlock(prop.players\_top\_mutex);

                    pthread\_mutex\_lock(prop.time\_player\_mutex);

                    gettimeofday(prop.update\_player\_time, NULL);

                    pthread\_mutex\_unlock(prop.time\_player\_mutex);

                    pthread\_mutex\_lock(prop.time\_map\_mutex);

                    gettimeofday(prop.update\_map\_time, NULL);

                    pthread\_mutex\_unlock(prop.time\_map\_mutex);

                    break;

                default:

                    break;

                }

                pthread\_mutex\_unlock(prop.map\_mutex);

                break;

            case doorAction:

                pthread\_mutex\_lock(prop.player\_mutex);

                pthread\_mutex\_lock(prop.map\_mutex);

                if (is\_field\_free\_player(input\_act.to\_x, input\_act.to\_y, prop.my\_id, \*prop.player\_list) && (prop.map\_s->at(input\_act.to\_y)[input\_act.to\_x] == door\_open || prop.map\_s->at(input\_act.to\_y)[input\_act.to\_x] == door\_lock))

                {

                    prop.map\_s->at(input\_act.to\_y)[input\_act.to\_x] = prop.map\_s->at(input\_act.to\_y)[input\_act.to\_x] == door\_open ? door\_lock : door\_open;

                    pthread\_mutex\_lock(prop.time\_map\_mutex);

                    gettimeofday(prop.update\_map\_time, NULL);

                    pthread\_mutex\_unlock(prop.time\_map\_mutex);

                }

                pthread\_mutex\_unlock(prop.player\_mutex);

                pthread\_mutex\_unlock(prop.map\_mutex);

                break;

            default:

                break;

            }

        }

        else

            pthread\_mutex\_unlock(prop.player\_mutex);

        n = recv(prop.sockfd, (action\_send \*)&input\_act, sizeof(action\_send), 0);

    }

    pthread\_mutex\_lock(prop.time\_map\_mutex); //закрыть все мьютексы чтобы не закрыть поток следящий за изменением в тот момент когда он имеет заблокированный мьютекс

    pthread\_mutex\_lock(prop.time\_player\_mutex);

    pthread\_mutex\_lock(prop.map\_mutex);

    pthread\_mutex\_lock(prop.player\_mutex);

    pthread\_cancel(sender\_start);                      //перед удалением данных для потока завершить поток отслеживания изменений

    prop.player\_list->at(prop.my\_id).is\_alive = false; //ячейка игрока более не занята

    gettimeofday(prop.update\_player\_time, NULL);

    pthread\_mutex\_unlock(prop.time\_map\_mutex);

    pthread\_mutex\_unlock(prop.time\_player\_mutex);

    pthread\_mutex\_unlock(prop.map\_mutex);

    pthread\_mutex\_unlock(prop.player\_mutex);

    pthread\_mutex\_lock(prop.player\_count\_mutex);

    (\*prop.player\_count\_connected)--;

    pthread\_mutex\_unlock(prop.player\_count\_mutex);

    close(prop.sockfd);

    delete &prop; //удаление структуры с данными для потоков

    return (void \*)(0);

}

void \*main\_client\_thread(void \*data)

{

    // открытие файла карты

    std::ifstream in("map.txt");

    //размерности карты

    size\_t lines, colonums;

    in >> lines >> colonums;

    if (!in.is\_open()) //выход если открытие файла не успешно

    {

        std::cout << "Map file not found\n";

        exit(2);

    }

    //создание поля для игры

    std::vector<std::vector<field\_cells\_type>> map\_s(lines, std::vector<field\_cells\_type>(colonums));

    {

        size\_t temp; //переменная для чтения ячеек, отдельный блок для удаления после выхода

        for (size\_t i = 0; i < map\_s.size(); i++)

            for (size\_t j = 0; j < map\_s[i].size(); j++)

            {

                in >> temp; //чтение в темп так как чтение напрямую происходит в char виде

                map\_s[i][j] = temp;

            }

    }

    size\_t start\_points\_count;

    in >> start\_points\_count;

    std::vector<std::pair<size\_t, size\_t>> start\_points(start\_points\_count);

    for (size\_t i = 0; i < start\_points\_count; i++) // считывание стартовых точек

        in >> start\_points[i].first >> start\_points[i].second;

    in.close();                    //закрытие файла

    in.open("player\_collors.txt"); //файл с цветами игроков

    if (!in.is\_open())             //выход если открытие файла не успешно

    {

        std::cout << "Collor file not found\n";

        exit(3);

    }

    size\_t player\_collors\_limit\_count;

    in >> player\_collors\_limit\_count;

    std::vector<player> player\_list(player\_collors\_limit\_count); //список игроков

    for (size\_t i = 0; i < player\_collors\_limit\_count; i++)      //чтение цветов и заполнение полей по умолчанию

    {

        size\_t r, g, b;

        in >> r >> g >> b;

        player\_list[i].r = r;

        player\_list[i].g = g;

        player\_list[i].b = b;

        player\_list[i].x = player\_list[i].y = 0;

        player\_list[i].id = i;

        player\_list[i].is\_alive = false; //соединён ли игрок

    }

    in.close();

    size\_t player\_limit\_count = std::min(player\_collors\_limit\_count, start\_points\_count);

    //топ игроков по счёту

    std::vector<top\_unit\_count> players\_top(player\_limit\_count, 0), game\_result;

    pthread\_mutex\_t players\_top\_mutex;

    pthread\_mutex\_init(&players\_top\_mutex, NULL);

    struct timeval players\_top\_time;

    gettimeofday(&players\_top\_time, NULL);

    pthread\_mutex\_t top\_time\_mutex;

    pthread\_mutex\_init(&top\_time\_mutex, NULL);

    // время последнего обновления карты для детекции изменений

    struct timeval update\_map\_time;

    gettimeofday(&update\_map\_time, NULL);

    pthread\_mutex\_t time\_map\_mutex;

    pthread\_mutex\_init(&time\_map\_mutex, NULL);

    // время последнего изменения в списке игроков

    struct timeval update\_player\_time;

    gettimeofday(&update\_player\_time, NULL);

    pthread\_mutex\_t time\_player\_mutex;

    pthread\_mutex\_init(&time\_player\_mutex, NULL);

    //мьютекс для одновременного доступа к карте только 1 потока

    pthread\_mutex\_t map\_mutex;

    pthread\_mutex\_init(&map\_mutex, NULL);

    //мьютекс для одновременного доступа к списку игроков 1 потока

    pthread\_mutex\_t player\_mutex;

    pthread\_mutex\_init(&player\_mutex, NULL);

    int sockfd;

    struct sockaddr\_in servaddr;

    bzero(&servaddr, sizeof(servaddr));

    servaddr.sin\_family = AF\_INET;

    servaddr.sin\_port = htons(((thread\_data \*)data)->sockfd);

    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)

        {

            close(sockfd);

            exit(1);

        }

    }

    socklen\_t servlen = sizeof(servaddr);

    listen(sockfd, 100);

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

    std::cout << "Listening on port: " << ntohs(servaddr.sin\_port) << std::endl;

    //число подключённых игроков

    size\_t player\_count\_connected = 0;

    //мьютекс для числа подключённых игроков

    pthread\_mutex\_t player\_count\_mutex;

    pthread\_mutex\_init(&player\_count\_mutex, NULL);

    top\_unit\_count coin\_count\_collected = 0;

    top\_unit\_count coint\_count\_max = coin\_spawn(map\_s, start\_points);

    ((thread\_data \*)data)->time\_map\_mutex = &time\_map\_mutex;

    ((thread\_data \*)data)->update\_map\_time = &update\_map\_time;

    ((thread\_data \*)data)->time\_player\_mutex = &time\_player\_mutex;

    ((thread\_data \*)data)->update\_player\_time = &update\_player\_time;

    ((thread\_data \*)data)->map\_mutex = &map\_mutex;

    ((thread\_data \*)data)->map\_s = &map\_s;

    ((thread\_data \*)data)->player\_mutex = &player\_mutex;

    ((thread\_data \*)data)->player\_list = &player\_list;

    ((thread\_data \*)data)->player\_count\_mutex = &player\_count\_mutex;

    ((thread\_data \*)data)->player\_count\_connected = &player\_count\_connected;

    ((thread\_data \*)data)->sockfd = sockfd;

    ((thread\_data \*)data)->start\_points = &start\_points;

    ((thread\_data \*)data)->players\_top\_mutex = &players\_top\_mutex;

    ((thread\_data \*)data)->players\_top = &players\_top;

    ((thread\_data \*)data)->top\_time\_mutex = &top\_time\_mutex;

    ((thread\_data \*)data)->players\_top\_time = &players\_top\_time;

    ((thread\_data \*)data)->coin\_count\_collected = &coin\_count\_collected;

    ((thread\_data \*)data)->coint\_count\_max = &coint\_count\_max;

    ((thread\_data \*)data)->game\_result = &game\_result;

    while (true)

    {

        struct sockaddr\_in cliaddr;

        socklen\_t clilen = sizeof(cliaddr);

        //ожидание нового клиента

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

        pthread\_mutex\_lock(&player\_count\_mutex);

        if (player\_count\_connected == player\_limit\_count)

        {

            pthread\_mutex\_unlock(&player\_count\_mutex);

            close(newsockfd);

            continue;

        }

        else

        {

            player\_count\_connected++;

            pthread\_mutex\_unlock(&player\_count\_mutex);

        }

        //сохранение данных необходимых для работы ротока, удаляются потоком

        thread\_data \*prop = new thread\_data;

        prop->time\_map\_mutex = &time\_map\_mutex;

        prop->update\_map\_time = &update\_map\_time;

        prop->time\_player\_mutex = &time\_player\_mutex;

        prop->update\_player\_time = &update\_player\_time;

        prop->map\_mutex = &map\_mutex;

        prop->map\_s = &map\_s;

        prop->player\_mutex = &player\_mutex;

        prop->player\_list = &player\_list;

        prop->player\_count\_mutex = &player\_count\_mutex;

        prop->player\_count\_connected = &player\_count\_connected;

        prop->sockfd = newsockfd;

        prop->start\_points = &start\_points;

        prop->players\_top\_mutex = &players\_top\_mutex;

        prop->players\_top = &players\_top;

        prop->top\_time\_mutex = &top\_time\_mutex;

        prop->players\_top\_time = &players\_top\_time;

        prop->coin\_count\_collected = &coin\_count\_collected;

        prop->coint\_count\_max = &coint\_count\_max;

        prop->game\_result = &game\_result;

        // запуск потока для клиента, нет необходимости хранить, завершит себя сам по потере соединения

        pthread\_t client\_start;

        pthread\_create(&client\_start, NULL, client\_reciver, (void \*)prop);

    }

    return (void \*)(0);

}

int main(int argc, char \*argv[])

{

    // нужное ли число аргументов при запуске

    if (argc != 2)

    {

        std::cout << "No port given\n";

        return -1;

    }

    // порт из строки в число

    int port = atoi(argv[1]);

    // 0 если строка не число

    if (port == 0)

    {

        perror("Not correct port");

        return -1;

    }

    //поток для приёма новых соединений

    pthread\_t main\_thread;

    //создание потока и передача порта

    thread\_data data;

    data.sockfd = port;

    pthread\_create(&main\_thread, NULL, main\_client\_thread, (void \*)&data);

    std::cout << "Ready to recive commands\n";

    bool done = false;

    std::string command;

    while (!done)

    {

        std::getline(std::cin, command); //чтение строки с командой

        if (command == "exit")

        {

            pthread\_cancel(main\_thread); //остановка потока

            close(data.sockfd);          //закрытие сокета

            exit(0);                     //выход

        }

        else if (command == "restart")

        {

            pthread\_mutex\_lock(data.map\_mutex);

            pthread\_mutex\_lock(data.players\_top\_mutex);

            pthread\_mutex\_lock(data.player\_mutex);

            for (size\_t i = 0; i < data.start\_points->size(); i++) //отправка игроков на стартовые позиции

                if (data.player\_list->at(i).is\_alive)

                {

                    data.player\_list->at(i).x = data.start\_points->at(i).second;

                    data.player\_list->at(i).y = data.start\_points->at(i).first;

                }

            pthread\_mutex\_unlock(data.player\_mutex);

            \*data.game\_result = \*data.players\_top; //сохранение результата ишры

            \*data.players\_top = std::vector<top\_unit\_count>(data.start\_points->size(), 0);

            pthread\_mutex\_lock(data.top\_time\_mutex);

            gettimeofday(data.players\_top\_time, NULL); //обновление времени для инициализации процедуры отправки топа

            pthread\_mutex\_unlock(data.top\_time\_mutex);

            map\_renew(\*data.map\_s); //очистка карты

            \*data.coin\_count\_collected = 0;

            \*data.coint\_count\_max = coin\_spawn(\*data.map\_s, \*data.start\_points);

            pthread\_mutex\_unlock(data.map\_mutex);

            pthread\_mutex\_unlock(data.players\_top\_mutex);

            pthread\_mutex\_lock(data.time\_player\_mutex);

            gettimeofday(data.update\_player\_time, NULL);

            pthread\_mutex\_unlock(data.time\_player\_mutex);

            pthread\_mutex\_lock(data.time\_map\_mutex);

            gettimeofday(data.update\_map\_time, NULL);

            pthread\_mutex\_unlock(data.time\_map\_mutex);

        }

        else if (command == "help")

        {

            std::cout << "Avalible commands:\n";

            std::cout << "exit - closes app\n";

            std::cout << "help - shows avalible commands\n";

            std::cout << "restart - show score and restart game\n";

        }

        else

        {

            std::cout << "Unknown command, please use help to get list of avalible commands\n";

        }

    }

    return 0;

}

Листинг В2 — Код из файла client.cpp

#include <netinet/in.h>

#include <arpa/inet.h>

#include <pthread.h>

#include <sys/time.h>

#include <unistd.h>

#include <signal.h>

#include "SDL2/SDL.h"

#include "SDL2/SDL2\_gfxPrimitives.h"

#include <vector>

#include <algorithm>

#include <iostream>

#include "common\_types.hpp"

void boards(SDL\_Renderer \*renderer, const int w, const int h, const int lines, const int colonums) //отрисовка границ клеток

{

    SDL\_SetRenderDrawColor(renderer, 0, 0, 0, SDL\_ALPHA\_OPAQUE);

    int y\_part = h / lines; //высота одной ячейки

    for (int i = 0; i <= lines; i++)

        SDL\_RenderDrawLine(renderer, 0, y\_part \* i, w, y\_part \* i);

    int x\_part = w / colonums; //ширина одной ячейки

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

        SDL\_RenderDrawLine(renderer, x\_part \* i, 0, x\_part \* i, h);

}

void map\_s(SDL\_Renderer \*renderer, const int w, const int h, std::vector<std::vector<field\_cells\_type>> &mat) //создание объектов поля, должно быть доработано до нескольких игроков разных цветов, текущие отображения мргут быть заменены на текстуры из файла

{

    int h\_side\_len = h / mat.size();    //высота одной ячейки

    int w\_side\_len = w / mat[0].size(); //ширина одной ячейки

    for (size\_t i = 0; i < mat.size(); i++)

        for (size\_t j = 0; j < mat[i].size(); j++)

            switch (mat[i][j])

            {

            case wall: //отрисовка чёрного прямоугольника для стены

                boxRGBA(renderer, w\_side\_len \* (int)j, h\_side\_len \* (int)i, w\_side\_len \* (int)(j + 1), h\_side\_len \* (int)(i + 1), 0, 0, 0, SDL\_ALPHA\_OPAQUE);

                break;

            case door\_lock:

                boxRGBA(renderer, w\_side\_len \* (int)j, h\_side\_len \* (int)i, w\_side\_len \* (int)(j + 1), h\_side\_len \* (int)(i + 1), 0, 0, 0, SDL\_ALPHA\_OPAQUE);

                circleRGBA(renderer, w\_side\_len \* (j + 0.5), h\_side\_len \* (i + 0.5), std::min(w\_side\_len, h\_side\_len) / 2, 255, 0, 0, SDL\_ALPHA\_OPAQUE);

                break;

            case door\_open:

                boxRGBA(renderer, w\_side\_len \* (int)j, h\_side\_len \* (int)i, w\_side\_len \* (int)(j + 1), h\_side\_len \* (int)(i + 1), 0, 0, 0, SDL\_ALPHA\_OPAQUE);

                filledCircleRGBA(renderer, w\_side\_len \* (j + 0.5), h\_side\_len \* (i + 0.5), std::min(w\_side\_len, h\_side\_len) / 2, 255, 255, 255, SDL\_ALPHA\_OPAQUE);

                circleRGBA(renderer, w\_side\_len \* (j + 0.5), h\_side\_len \* (i + 0.5), std::min(w\_side\_len, h\_side\_len) / 2, 255, 0, 0, SDL\_ALPHA\_OPAQUE);

                break;

            case trap\_on:

                lineRGBA(renderer, w\_side\_len \* (int)j, h\_side\_len \* (int)i, w\_side\_len \* (int)(j + 1), h\_side\_len \* (int)(i + 1), 255, 0, 0, SDL\_ALPHA\_OPAQUE);

                lineRGBA(renderer, w\_side\_len \* (int)j, h\_side\_len \* (int)(i + 1), w\_side\_len \* (int)(j + 1), h\_side\_len \* (int)i, 255, 0, 0, SDL\_ALPHA\_OPAQUE);

                break;

            case coin:

                filledCircleRGBA(renderer, w\_side\_len \* (j + 0.5), h\_side\_len \* (i + 0.5), std::min(w\_side\_len, h\_side\_len) / 4, 255, 215, 0, SDL\_ALPHA\_OPAQUE);

                filledCircleRGBA(renderer, w\_side\_len \* (j + 0.5), h\_side\_len \* (i + 0.5), std::min(w\_side\_len, h\_side\_len) / 7, 255, 255, 255, SDL\_ALPHA\_OPAQUE);

                break;

            default:

                break;

            }

}

void players\_print(SDL\_Renderer \*renderer, const int w, const int h, const size\_t x\_size, const size\_t y\_size, std::vector<player> &player\_list)

{

    int h\_side\_len = h / x\_size; //высота одной ячейки

    int w\_side\_len = w / y\_size; //ширина одной ячейки

    for (size\_t i = 0; i < player\_list.size(); i++)

        if (player\_list[i].is\_alive)

        {

            filledCircleRGBA(renderer, w\_side\_len \* (player\_list[i].x + 0.5), h\_side\_len \* (player\_list[i].y + 0.5), 0.8 \* std::min(w\_side\_len, h\_side\_len) / 2, player\_list[i].r, player\_list[i].g, player\_list[i].b, SDL\_ALPHA\_OPAQUE);

        }

}

struct thread\_data //структура для передачи информации в поток не объявленна в общем заголовочном файле так как может быть разная для сервера и клиента

{

    pthread\_mutex\_t \*time\_mutex;

    struct timeval \*update\_time;

    pthread\_mutex\_t \*map\_mutex;

    std::vector<std::vector<field\_cells\_type>> \*map\_s;

    pthread\_mutex\_t \*player\_mutex;

    std::vector<player> \*player\_list;

    int sockfd;

    size\_t my\_id;

};

void \*reciver(void \*data)

{

    thread\_data &prop = \*((thread\_data \*)data); //приведение указателя к ссылке

    prepare\_message\_data\_send input\_format; //для получения сообщения ингициализирующего передачу

    int n = recv(prop.sockfd, (prepare\_message\_data\_send \*)&input\_format, sizeof(prepare\_message\_data\_send), 0);

    while (n) //пока соединение не разорванно

    {

        switch (input\_format.type) //в зависимости от типа передаваемых данных

        {

        case field\_type:

            pthread\_mutex\_lock(prop.map\_mutex);

            if (prop.map\_s->size() != input\_format.size || prop.map\_s->at(0).size() != input\_format.second\_size)

                \*prop.map\_s = std::vector<std::vector<field\_cells\_type>>(input\_format.size, std::vector<field\_cells\_type>(input\_format.second\_size));

            for (size\_t i = 0; i < prop.map\_s->size(); i++) //получение поля построчно

                n = recv(prop.sockfd, (field\_cells\_type \*)prop.map\_s->at(i).data(), prop.map\_s->at(i).size() \* sizeof(field\_cells\_type), 0);

            pthread\_mutex\_unlock(prop.map\_mutex);

            pthread\_mutex\_lock(prop.time\_mutex);

            gettimeofday(prop.update\_time, NULL); //обновление метки времени

            pthread\_mutex\_unlock(prop.time\_mutex);

            break;

        case player\_list:

            pthread\_mutex\_lock(prop.player\_mutex);

            if (prop.player\_list->size() != input\_format.size)

                \*prop.player\_list = std::vector<player>(input\_format.size);

            n = recv(prop.sockfd, (player \*)prop.player\_list->data(), prop.player\_list->size() \* sizeof(player), 0);

            pthread\_mutex\_unlock(prop.player\_mutex);

            pthread\_mutex\_lock(prop.time\_mutex);

            gettimeofday(prop.update\_time, NULL); //обновление метки времени

            pthread\_mutex\_unlock(prop.time\_mutex);

            break;

        case my\_number\_from\_list:

            prop.my\_id = input\_format.size;

            break;

        case score:

        {

            std::vector<top\_unit\_count> game\_score(input\_format.size);

            n = recv(prop.sockfd, (top\_unit\_count \*)game\_score.data(), game\_score.size() \* sizeof(top\_unit\_count), 0);

            std::cout << "Game ended! Score list" << std::endl;

            for (size\_t i = 0; i < game\_score.size(); i++)

                std::cout << i << " : " << game\_score[i] << std::endl;

            std::cout << "You are the " << prop.my\_id << " player" << std::endl;

        }

        break;

        default:

            break;

        }

        n = recv(prop.sockfd, (prepare\_message\_data\_send \*)&input\_format, sizeof(prepare\_message\_data\_send), 0); //получение нового сообщения от сервера

    }

    return (void \*)(0);

}

int main(int argc, char \*argv[])

{

    if (argc != 2) //дан ли аргумент адрес

    {

        perror("No address given");

        return -1;

    }

    int border = strstr(argv[1], ":") - argv[1]; //поиск : разделителя строки

    if (border + argv[1] == NULL) //strstr возвращает NULL если ничего не найдено

    {

        perror("Not correct format of address");

        return -1;

    }

    char \*ip = new char[border + 1];       //для хранения ip адреса вырезанного из строки

    strncpy(ip, argv[1], border);          //копирование ip адреса

    ip[border] = '\0';                     //окончание строки

    int port = atoi(argv[1] + border + 1); // преобразование порта к числовому виду

    if (port == 0) // 0 если строка не число

    {

        perror("Not correct port");

        return -1;

    }

    std::vector<player> player\_list;                //список игроков

    std::vector<std::vector<field\_cells\_type>> mat; //матрица для хранения поля будет пересоздана при получении размера

    struct timeval update\_time, last\_time;          //время обновления данных и время последних отрисованных данных

    gettimeofday(&update\_time, NULL);               //занесение текущего времени

    last\_time = update\_time;                        //для избегания отрисовки до получения версии сервера

    pthread\_mutex\_t time\_mutex;                     //мьютекс для времени

    pthread\_mutex\_init(&time\_mutex, NULL);

    pthread\_mutex\_t map\_mutex; //мьютекс для карты

    pthread\_mutex\_init(&map\_mutex, NULL);

    pthread\_mutex\_t player\_mutex; //мьютекс для списка игроков

    pthread\_mutex\_init(&player\_mutex, NULL);

    struct sockaddr\_in servaddr;

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

    servaddr.sin\_family = AF\_INET;

    servaddr.sin\_port = htons(port);

    inet\_aton(ip, &(servaddr.sin\_addr));

    delete[] ip; //использован больше не нужен

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

    {

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

        return 1;

    }

    int n = 1; // для хранения возврата соединения, если 0 соединение с сервером потеряно

    thread\_data share\_data; //данные для передачи в поток

    share\_data.sockfd = sockfd;

    share\_data.map\_s = &mat;

    share\_data.map\_mutex = &map\_mutex;

    share\_data.update\_time = &update\_time;

    share\_data.time\_mutex = &time\_mutex;

    share\_data.player\_mutex = &player\_mutex;

    share\_data.player\_list = &player\_list;

    pthread\_t reciver\_thread; //запуск потока

    pthread\_create(&reciver\_thread, NULL, reciver, (void \*)&share\_data);

    pthread\_mutex\_lock(&map\_mutex);

    pthread\_mutex\_lock(&player\_mutex);

    while (mat.empty() || player\_list.empty()) //не даёт начать отрисовку пока не получена карта и список игроков от сервера

    {

        pthread\_mutex\_unlock(&map\_mutex);

        pthread\_mutex\_unlock(&player\_mutex);

        usleep(1);

        if (pthread\_kill(reciver\_thread, 0) != 0) //проверка жив ли поток(поток слушающий данные узнаёт о потере соединения первый и закрывается)

        {

            std::cout << "connection lost" << std::endl;

            close(sockfd);

            exit(0);

        }

        pthread\_mutex\_lock(&map\_mutex);

        pthread\_mutex\_lock(&player\_mutex);

    }

    pthread\_mutex\_unlock(&map\_mutex);

    pthread\_mutex\_unlock(&player\_mutex);

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

    {

        SDL\_Window \*window = NULL;

        SDL\_Renderer \*renderer = NULL;

        if (SDL\_CreateWindowAndRenderer(640, 480, SDL\_WINDOW\_RESIZABLE, &window, &renderer) == 0) //создать окно размер которого разрешено изменять

        {

            SDL\_bool done = SDL\_FALSE; //хз зачем нужен такой тип в примере sdl он был)

            SDL\_bool need\_review = SDL\_FALSE;

            while (!done && n) //пока не завершено или соединение не потеряно

            {

                if (pthread\_kill(reciver\_thread, 0) != 0) //проверка жив ли поток(поток слушающий данные узнаёт о потере соединения первый и закрывается)

                {

                    n = 0;

                    std::cout << "connection lost" << std::endl;

                    continue;

                }

                SDL\_Event event;

                while (SDL\_PollEvent(&event))

                {

                    if (event.type == SDL\_QUIT)

                    {

                        done = SDL\_TRUE;

                    }

                    else if (event.type == SDL\_WINDOWEVENT)

                    {

                        switch (event.window.event)

                        {

                        case SDL\_WINDOWEVENT\_RESTORED:

                        case SDL\_WINDOWEVENT\_SIZE\_CHANGED: //для нормальной отрисовки при resize

                            need\_review = SDL\_TRUE;

                            break;

                        default:

                            break;

                        }

                    }

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

                    {

                        struct action\_send temp; //отправка действий на сервер

                        pthread\_mutex\_lock(&player\_mutex);

                        size\_t x = player\_list[share\_data.my\_id].x;

                        size\_t y = player\_list[share\_data.my\_id].y;

                        pthread\_mutex\_unlock(&player\_mutex);

                        temp.from\_x = x;

                        temp.from\_y = y;

                        switch (event.key.keysym.sym)

                        {

                        case SDLK\_UP:

                            temp.to\_x = x;

                            temp.to\_y = y - 1;

                            temp.action = move;

                            n = send(sockfd, (action\_send \*)&temp, sizeof(action\_send), MSG\_NOSIGNAL);

                            break;

                        case SDLK\_DOWN:

                            temp.to\_x = x;

                            temp.to\_y = y + 1;

                            temp.action = move;

                            n = send(sockfd, (action\_send \*)&temp, sizeof(action\_send), MSG\_NOSIGNAL);

                            break;

                        case SDLK\_LEFT:

                            temp.to\_x = x - 1;

                            temp.to\_y = y;

                            temp.action = move;

                            n = send(sockfd, (action\_send \*)&temp, sizeof(action\_send), MSG\_NOSIGNAL);

                            break;

                        case SDLK\_RIGHT:

                            temp.to\_x = x + 1;

                            temp.to\_y = y;

                            temp.action = move;

                            n = send(sockfd, (action\_send \*)&temp, sizeof(action\_send), MSG\_NOSIGNAL);

                            break;

                        case SDLK\_r:

                            temp.action = doorAction;

                            pthread\_mutex\_lock(&map\_mutex);

                            if (y - 1 < mat.size() && (mat[y - 1][x] == door\_open || mat[y - 1][x] == door\_lock))

                            {

                                temp.to\_x = x;

                                temp.to\_y = y - 1;

                                n = send(sockfd, (action\_send \*)&temp, sizeof(action\_send), MSG\_NOSIGNAL);

                            }

                            else if (y + 1 < mat.size() && (mat[y + 1][x] == door\_open || mat[y + 1][x] == door\_lock))

                            {

                                temp.to\_x = x;

                                temp.to\_y = y + 1;

                                n = send(sockfd, (action\_send \*)&temp, sizeof(action\_send), MSG\_NOSIGNAL);

                            }

                            else if (x - 1 < mat[y].size() && (mat[y][x - 1] == door\_open || mat[y][x - 1] == door\_lock))

                            {

                                temp.to\_x = x - 1;

                                temp.to\_y = y;

                                n = send(sockfd, (action\_send \*)&temp, sizeof(action\_send), MSG\_NOSIGNAL);

                            }

                            else if (x + 1 < mat[y].size() && (mat[y][x + 1] == door\_open || mat[y][x + 1] == door\_lock))

                            {

                                temp.to\_x = x + 1;

                                temp.to\_y = y;

                                n = send(sockfd, (action\_send \*)&temp, sizeof(action\_send), MSG\_NOSIGNAL);

                            }

                            pthread\_mutex\_unlock(&map\_mutex);

                            break;

                        default:

                            break;

                        }

                    }

                }

                pthread\_mutex\_lock(&time\_mutex);

                if (!memcmp(&update\_time, &last\_time, sizeof(timeval)) && !need\_review) //если карта не изменилась не отрисовывать снова, отрисовать снова если специальный флаг поднят

                {

                    pthread\_mutex\_unlock(&time\_mutex);

                    usleep(1);

                    continue;

                }

                pthread\_mutex\_unlock(&time\_mutex);

                last\_time = update\_time; //обновить до отрисованной версии

                need\_review = SDL\_FALSE; //опустить специальный флаг

                SDL\_SetRenderDrawColor(renderer, 255, 255, 255, SDL\_ALPHA\_OPAQUE);

                SDL\_RenderClear(renderer); //очистить поле белым цветом

                int w, h;

                SDL\_GetWindowSize(window, &w, &h); //на случай изменения размеров окна получить новый размер

                pthread\_mutex\_lock(&map\_mutex); //отрисовка элементов карты и границ между клетками

                boards(renderer, w, h, mat.size(), mat[0].size());

                map\_s(renderer, w, h, mat);

                pthread\_mutex\_unlock(&map\_mutex);

                pthread\_mutex\_lock(&player\_mutex);

                players\_print(renderer, w, h, mat.size(), mat[0].size(), player\_list);

                pthread\_mutex\_unlock(&player\_mutex);

                SDL\_RenderPresent(renderer);

            }

        }

        if (renderer)

        {

            SDL\_DestroyRenderer(renderer);

        }

        if (window)

        {

            SDL\_DestroyWindow(window);

        }

    }

    pthread\_mutex\_destroy(&time\_mutex);

    pthread\_mutex\_destroy(&map\_mutex);

    pthread\_mutex\_destroy(&player\_mutex);

    SDL\_Quit();

    close(sockfd);

    return 0;

}

Листинг В3 — Код из файла common\_types.hpp

//файл типов общих для сервера и клиента и как правило используемых для передачи

enum field

{

    empty = 0,

    wall = 1,

    door\_lock = 2,

    door\_open = 3,

    trap = 4,

    trap\_on = 5,

    coin = 6

};

enum actions

{

    move = 0,

    doorAction = 1

};

struct action\_send

{

    actions action;

    size\_t from\_x, from\_y, to\_x, to\_y;

};

enum data\_type\_send

{

    field\_type = 0,

    player\_list = 1,

    my\_number\_from\_list = 2,

    score = 3

};

struct prepare\_message\_data\_send

{

    data\_type\_send type;

    size\_t size, second\_size;

};

#define field\_cells\_type uint8\_t

struct player

{

    size\_t id;

    size\_t x, y;

    uint8\_t r, g, b;

    bool is\_alive;

};

#define top\_unit\_count size\_t

Листинг В4 — Файл map.txt

11 27

0 1 0 0 0 0 0 0 0 0 0 1 1 1 0 0 2 0 1 1 1 1 1 1 0 0 0

0 1 0 1 1 2 1 1 1 1 0 1 0 1 1 1 1 0 0 0 0 2 0 0 1 1 0

0 1 0 1 0 0 1 0 0 4 0 1 0 2 0 0 0 0 1 1 1 0 1 0 4 1 0

0 1 0 1 1 1 1 1 1 1 0 1 1 1 1 1 1 0 1 4 1 2 1 0 1 1 0

0 1 0 0 4 0 0 0 0 1 0 2 0 1 4 0 0 0 1 0 0 0 0 0 0 0 0

0 1 1 1 0 1 1 1 1 1 0 1 1 1 0 1 1 0 1 0 1 1 0 1 0 1 0

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 4 1 0 1 0 0 1 0 1 0 1 0

1 1 1 1 0 1 1 1 1 1 1 1 2 1 0 1 1 2 1 0 1 1 0 1 0 1 0

1 0 0 1 0 0 4 1 4 0 2 0 0 1 0 1 0 0 1 0 1 0 0 1 0 1 0

1 0 0 1 1 1 0 1 1 1 1 1 1 1 0 1 1 1 1 0 1 1 1 1 0 1 0

1 1 0 0 2 0 0 0 0 0 0 0 0 0 0 2 0 0 0 0 0 0 0 0 0 1 0

4

0 0

10 26

0 24

8 16

Листинг В5 —Файл player\_collors.txt

20

92 195 158

255 173 225

163 255 208

197 178 250

211 228 152

71 192 246

216 163 105

36 190 191

237 158 122

142 238 255

255 182 165

199 255 254

206 155 201

167 177 117

205 197 255

255 217 155

253 226 255

137 180 174

210 160 146

222 213 185