Министерство науки и высшего образования Российской Федерации Федеральное государственное бюджетное образовательное учреждение высшего образования   
«Петрозаводский государственный университет»   
Физико-технический институт   
Кафедра твердого тела

Курсовой проект по дисциплине “Технология программирования”.

Авторы работы:   
студенты группы 21418  
Н. С. Сорокин  
А. К. Кемова  
С. Д. Забирохин  
Ф. М. Сафронов  
А. А. Губарь   
«\_\_\_\_» \_\_\_\_\_\_\_\_\_ 2023 г.

Научный руководитель:  
Кандидат физико-математических наук  
А. В. Бульба  
«\_\_\_\_» \_\_\_\_\_\_\_\_\_ 2023 г.

Петрозаводск 2023

Содержание

[**1 Цель 3**](#_ed4tt0m3kcn9)

[2 **Программная реализация 4**](#_2jihod2o10vs)

[2.1 Среда разработки 4](#_4lirjypoik04)

[2.2 Язык 4](#_kp181f9s62vg)

[2.3 Перечисление всех созданных заголовочных файлов и единиц компиляции 4](#_3e1o0q7ob8b0)

[**3 Пошаговое описание процесса обработки 6**](#_tq7m3l3jjxt0)

[3.1 Краткое словесное описание сюжета. 6](#_rzkjw0jp3ylw)

[3.2 Описание имеющихся у заказчика материалов (данных). 6](#_36m5iteqou0e)

[3.3 Действующих субъектов (actors) и их функциональные обязанности. 6](#_rpmm7galhzl2)

[3.4 Список и диаграммы вариантов использования. 6](#_wy6krp7fvylo)

[3.5 Описание вариантов использования. 8](#_lxcmqkr982hm)

[3.6 Диаграммы действий. 10](#_3sqf6f8xwin3)

[3.7 Предварительный список существительных. 11](#_weuhf8ys63ih)

[3.8 Уточненный список существительных. 11](#_9p5ij06stwhv)

[3.9 Список атрибутов. 13](#_3g27j8jcdcpd)

[3.10 Список сообщений(методов). 16](#_htdxgd7fyj0t)

[3.11 Диаграмма классов. 20](#_5agwqu900bst)

[3.12 Листинг заголовочных (.H) файлов. 20](#_75qohlwwqtv1)

[3.13 Листинг исходных файлов (.cpp). 32](#_baespc8ofgz9)

[3.14 Руководство пользователя. 73](#_z2z2fhc1bdrh)

[**4 История проекта на GitHub 79**](#_fiti79xp8ch2)

[**5 Заключение 80**](#_obtdu0ys9hyc)

### 1 Цель

Необходимо на С++ (IDE Qt Creator (Qt 5.4.2)) с использованием библиотеки SFML (Simple and Fast Multimedia Library — простая и быстрая мультимедийная библиотека) написать простую 2D игру.

В нашем случае 2D игрой будет культовая игра 90-х – “Tank 1990” (“Battle City”).

### 2 Программная реализация

#### 2.1 Среда разработки

Qt Creator — кроссплатформенная свободная IDE для разработки на С, С++ и QML. Разработана Trolltech (Digia) для работы с фреймворком Qt. Включает в себя графический интерфейс отладчика и визуальные средства разработки интерфейса как с использованием QtWidgets, так и QML, поддерживает компиляторы: GCC, Clang, MinGW, MSVC, Linux ICC, GCCE, RVCT, WINSCW.

Библиотека SFML версии 2.5.1, отладчик MinGW v8.1.0.

#### 2.2 Язык

С++ - это язык, поддерживающий объектно-ориентированное программирование, которое основано на представлении программы в виде совокупности объектов, каждый из которых является экземпляром определенного класса, а классы образуют иерархию наследования.

#### 2.3 Перечисление всех созданных заголовочных файлов и единиц компиляции

А) Заголовочные файлы:

base.h – интерфейсная часть класса: Base;

brickwall.h – интерфейсная часть класса: BrickWall;

bullet.h – интерфейсная часть класса: Bullet;

enemy.h – интерфейсная часть класса: Enemy;

gamescreen.h – интерфейсная часть класса: GameScreen;

inputrecordscreen.h – интерфейсная часть класса: InputRecordScreen;

listofrecords.h – интерфейсная часть класса: ListOfRecords;

map.h – интерфейсная часть класса: Map;

mapselectionscreen.h – интерфейсная часть класса: MapSelectionScreen;

metalwall.h – интерфейсная часть класса: MetalWall;

movingobj.h – интерфейсная часть класса: MovingObj;

object.h – интерфейсная часть класса: Object;

outputrecordscreen.h – интерфейсная часть класса: OutputRecordScreen;

player.h – интерфейсная часть класса: Player;

record.h – интерфейсная часть класса: Record;

screens.h – интерфейсная часть класса: Screens;

stationaryobj.h – интерфейсная часть класса: StationaryObj;

userinterfacescreen.h – интерфейсная часть класса: UserInterfaceScreen;

Б) В .cpp описана реализация методов классов, которые лежат в отдельных заголовочных файлов:

base.cpp – реализация методов класса Base;

brickwall.cpp – реализация методов класса BrickWall;

bullet.cpp – реализация методов класса Bullet;

enemy.cpp – реализация методов класса Enemy;

gamescreen.cpp – реализация методов класса GameScreen;

gamescreencontrol.cpp – продолжает реализацию методов класса GameScreen;

inputrecordscreen.cpp – реализация методов класса InputRecordScreen;

listofrecords.cpp – реализация методов класса ListOfRecords;

map.cpp – реализация методов класса Map;

mapselectionscreen.cpp – реализация методов класса MapSelectionScreen;

metalwall.cpp – реализация методов класса MetalWall;

movingobj.cpp – реализация методов класса MovingObj;

object.cpp – реализация методов класса Object;

outputrecordscreen.cpp – реализация методов класса OutputRecordScreen;

player.cpp – реализация методов класса Player;

record.cpp – реализация методов класса Record;

screens.cpp – реализация методов класса Screens;

stationaryobj.cpp – реализация методов класса StationaryObj;

userinterfacescreen.cpp – реализация методов класса UserInterfaceScreen;

main.cpp – находится главная функция main, инициализируется класс UserInterfaceScreen и вызывается метод interact;

### 3 Пошаговое описание процесса обработки

#### 3.1 Краткое словесное описание сюжета.

Сюжет рассматриваемого примера прост, к нам обратился небольшой издатель игра и попросил воссоздать популярную игру 90-х – “Tank 1990” (“Battle City”). Полигон действий виден сверху. Игрок должен, управляя своим танком, уничтожить все вражеские танки на уровне, которые постепенно появляются вверху игрового поля. Враги пытаются уничтожить штаб игрока (внизу игрового поля в виде орла) и его танк. На каждом уровне нужно уничтожить восемь единиц бронетехники противника. Если противник (или игрок) сможет разрушить штаб или лишит игрока всех жизней — игра окончена.

Также заказчик попросил сделать таблицу рекордов и пользовательский интерфейс(меню). Программный код должен поддерживать расширяемость. Для первой версии игры минимальное доступное количество уровней должно составлять - 1.

#### 3.2 Описание имеющихся у заказчика материалов (данных).

У заказчика на руках были рисунки танков, препятствий, а также других игровых объектов.

Таблица №1. Исходные данные заказчика.

|  |  |  |  |
| --- | --- | --- | --- |
| Рис.1. Кирпичная стена | Рис.2. База | Рис.3. Пуля | Рис.4. Металлическая стена |
|  |  |  |  |
| Рис.5. Жизни игрока | Рис.6. Игрок | Рис.7. Карта | Рис.8. Вражеский танк |

#### 3.3 Действующих субъектов (actors) и их функциональные обязанности.

Так как игра подразумевает однопользовательский режим, то действующим лицом будет игрок. Таким образом, одному игроку доступен весь функционал игры.

#### 3.4 Список и диаграммы вариантов использования.

**Список вариантов использования:**

1. Начать игру;
2. Выбрать карту;
3. Просмотреть таблицу рекордов;
4. Добавить запись в таблицу рекордов;
5. Перезаписать счет;
6. Двигаться влево;
7. Двигаться вправо;
8. Двигаться вверх;
9. Двигаться вниз;
10. Стрелять;
11. Выйти из игры;
12. Играть;
13. Вернуться назад.

В ходе анализа требований были разработаны две UML-диаграммы вариантов использования . Рассмотрим варианты использования в игровом меню, представленные на рисунке 9.

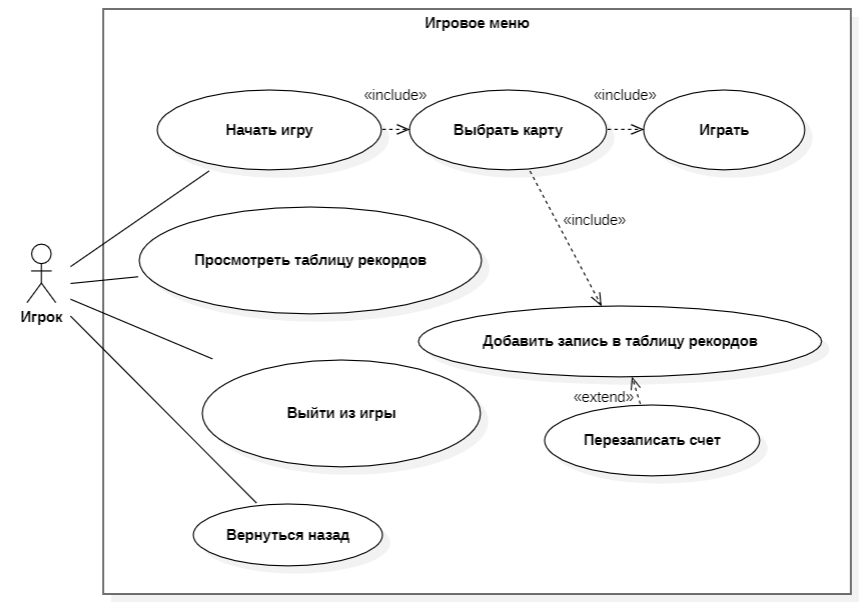


Рис.9. Диаграмма вариантов использования в игровом меню.

Далее рассмотрим варианты использования в игровой сцене, представленные на рисунке 10.

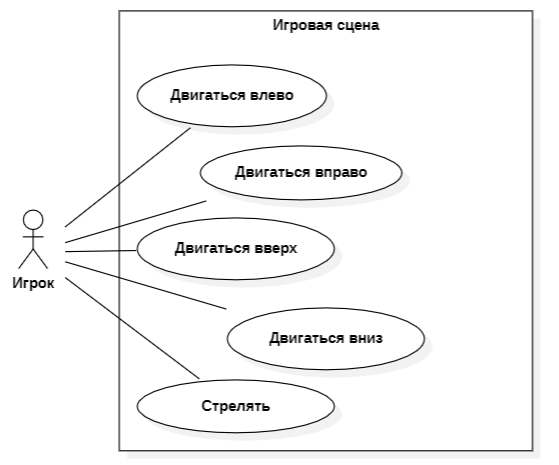


Рис.10. Диаграмма вариантов использования в игровой сцене.

#### 3.5 Описание вариантов использования.

Описание вариантов использования:

**“Начать игру”**

Открывает меню, где пользователь может выбрать карту(уровень). После выбора карты начинается сама игра. При победе или поражении предлагается записать свой счет в таблицу рекордов.

**“Выбрать карту”**

Игрок выбирает карту(уровень) путем нажатия кнопок влево/вправо, подтверждением карты служит кнопка “Enter”. На форме также присутствует кнопка “назад”.

**“Просмотреть таблицу рекордов”**

Позволяет пользователю просмотреть таблицу рекордов и увидеть игроков с наибольшим счетом. На форме также присутствует кнопка “назад”. Кнопка является активной всегда. Срабатывает при нажатии клавиши “Enter” на клавиатуре.

**“Добавить запись в таблицу рекордов”**

После победы или поражения пользователю предлагается записать свое имя в таблицу рекордов. Если такое имя уже есть, то предлагает перезаписать счет. Навигация по форме осуществляется с помощью стрелок на клавиатуре.

**“Перезаписать счет”**

В случае, если обнаружится, что игрок с таким никнеймом уже есть, предлагает переписать его старый счет на новый. Иначе счет не запишется в таблицу рекордов. Навигация по форме осуществляется с помощью стрелок на клавиатуре.

**“Двигаться влево”**

Игрок может перемещать свой танк влево при помощи кнопки “влево” на клавиатуре. Если перед игроком нет препятствий, то действие будет выполнено, в противном случае пропущено.

**“Двигаться вправо”**

Игрок может перемещать свой танк вправо при помощи кнопки “вправо” на клавиатуре. Если перед игроком нет препятствий, то действие будет выполнено, в противном случае пропущено.

**“Двигаться вверх”**

Игрок может перемещать свой танк вверх при помощи кнопки “вверх” на клавиатуре. Если перед игроком нет препятствий, то действие будет выполнено, в противном случае пропущено.

**“Двигаться вниз”**

Игрок может перемещать свой танк вниз при помощи кнопки “вниз” на клавиатуре. Если перед игроком нет препятствий, то действие будет выполнено, в противном случае пропущено.

**“Стрелять”**

Игрок может выпускать снаряд по направлению дула танка при помощи кнопки “Enter”. Снаряды должны выпускаться с задержкой.

**“Выйти из игры”**

Игрок, находясь в главном меню, может выйти из игры, выбрав нужный пункт при помощи стрелок на клавиатуре и нажав клавишу “Enter” на клавиатуре.

**“Играть”**

После выбора карты начинается игровая сцена и заканчивается при поражении или победе.

**“Вернуться назад”**

В любой момент, находясь в одном из пунктов меню, игрок может вернуться в главное меню при помощи кнопки “Назад”, кроме игровой сцены. Выбирается стрелками на клавиатуре. Подтверждается нажатием клавиши “Enter” на клавиатуре.

#### 3.6 Диаграммы действий.

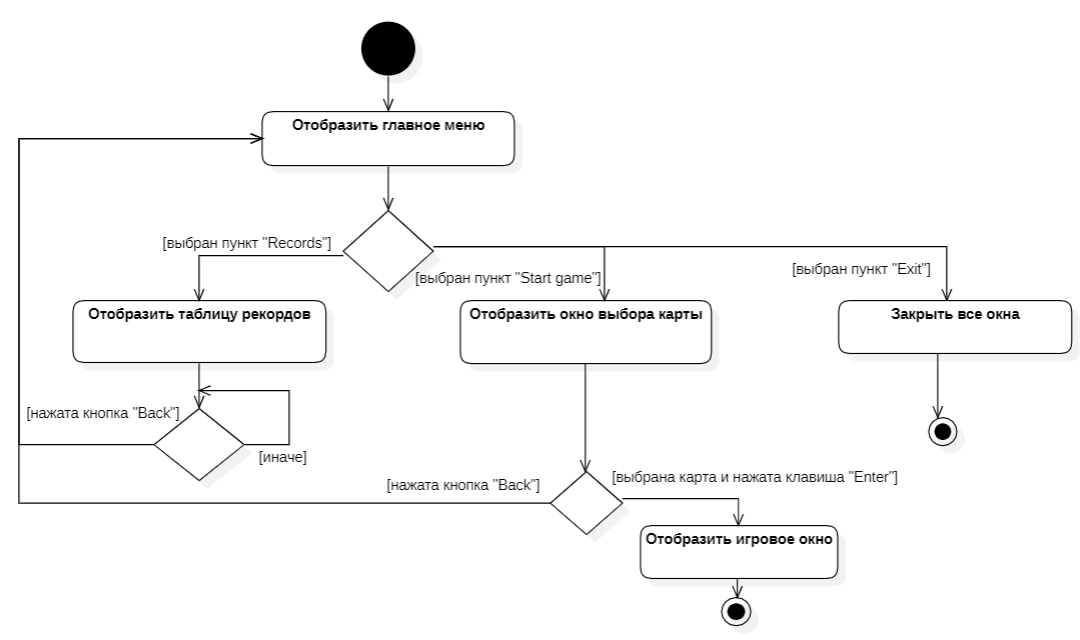


Рис.11. Диаграмма действий для меню.

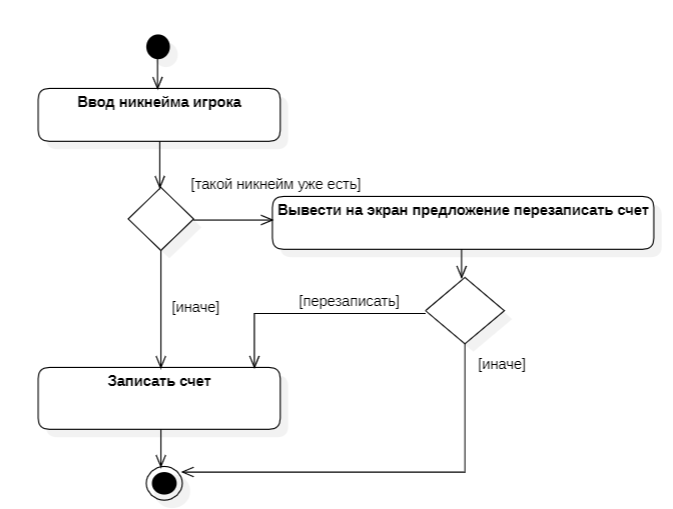


Рис.12. Диаграмма действий для записи счета.

#### 3.7 Предварительный список существительных.

Рассмотрим список всех существительных, взятых из описаний вариантов

использования:

1) Экран интерфейса пользователя;

2) Игрок;

3) Танк;

4) Пуля;

5) Таблица рекордов;

6) Экран вывода рекордов;

7) Экран выбора карты;

8) Противник;

9) Окно;

10) Карта;

11) Счет;

12) Препятствие;

13) Штаб;

14) Экран ввода рекорда;

15) Рекорд;

16) Жизни;

17) Кнопка;

18) Никнейм;

19) Игровое окно;

20) Кирпичная стена;

21) Металлическая стена;

22) Движущиеся объекты;

23) Статичные объекты;

24) Объекты.

#### 3.8 Уточненный список существительных.

1. Экран интерфейса пользователя;
2. Игрок;
3. Пуля;
4. Противник;
5. Список рекордов;
6. Экран вывода рекордов;
7. Экран;
8. Экран выбора карты;
9. Карта;
10. Экран ввода рекорда;
11. Рекорд;
12. Игровое окно;
13. Кирпичная стена;
14. Металлическая стена;
15. Движущиеся объекты;
16. Статичные объекты;
17. Объекты;
18. База.

Избыточность существительных добавлена неслучайно, а для того чтобы сделать возможность расширять игру по мере возможности.

Получившиеся классы:

Base - класс, в котором определяются данные о таком игровом объекте, как “база”.

BrickWall - класс, в котором определяются данные о таком игровом объекте, как “кирпичная стена”.

Bullet - класс, в котором определяются данные о таком игровом объекте, как “пуля”.

Enemy - класс, в котором определяются данные о таком игровом объекте, как “противник”.

GameScreen - класс, в котором происходит игра. Хранит в себе списки врагов, стен и других объектов. Также отвечает за управление и поведение объектов на игровой сцене.

InputRecordScreen - класс, который будет являться посредником между пользовательским интерфейсом и списком рекордов. В нем есть методы для взаимодействия с пользователем(обработка нажатия клавиш, запись рекорда в список).

ListOfRecords - класс, в котором хранится список рекордов. Присутствуют методы по добавлению нового рекорда, фильтрации рекордов, проверке на повторение и так далее .

Map - класс, который создает списки игровых объектов и расставляет в порядке расположения их на карте.

MapSelectionScreen - класс, который будет отображать окно выбора карты. Является промежуточным вариантом между игровым окном и главным меню.

MetalWall - класс, в котором определяются данные о таком игровом объекте, как “металлическая стена”.

MovingObj - абстрактный класс, в котором есть данные о здоровье и направлении объекта, а также есть все унаследованные атрибуты от класса Object.

Является родительским классом для всех подвижных объектов(противники, игрок, пули).

Object - абстрактный класс, в котором есть данные о положении, размере, текстуре объекта, а также виртуальные методы для взаимодействия с этими данными.

OutputRecordsScreen - класс, который будет являться посредником между пользовательским интерфейсом и списком рекордов. В нем есть методы для взаимодействия с пользователем(вывод списка рекордов в виде таблицы).

Player - класс, в котором определяются данные о таком игровом объекте, как “игрок”.

Record - класс, в котором хранятся данные о конкретном рекорде. Есть методы для взаимодействия с этими данными.

Screens - класс, в котором хранятся данные о ширине и длине кнопок, объект событий. Является родительским классом для всех экранов.

StationaryObj - абстрактный класс, в котором есть данные о разрушаемости и здоровье объекта, а также есть все унаследованные атрибуты от класса Object. Является родительским классом для всех неподвижных объектов(стены, база).

UserInterfaceScreen - класс, который будет взаимодействовать с пользователем. Именно он связывает остальные классы в единую целостность. Из методов есть только взаимодействие с пользователем.

#### 3.9 Список атрибутов.

1. Base:
   1. унаследованные от класса StationaryObj.
2. BrickWall:
   1. унаследованные от класса StationaryObj.
3. Bullet:
   1. унаследованные от класса MovingObj.
4. Enemy:
   1. Clock cooldown - время перезарядки;
   2. Clock cooldownAction - время перезарядки действия;
   3. int numberOfAction - номер действия;
   4. унаследованные от класса MovingObj.
5. GameScreen:
   1. унаследованные от класса Screens;
   2. RenderWindow\* win - указатель на окно;
   3. bool activeWin - активное окно;
   4. int numberMap - номер карты;
   5. int\* ptrScore - указатель на счет;
   6. RectangleShape gameBox - игровое поле;
   7. Text textPreScore,textScore - счет игрока;
   8. int countEnemy - счетчик противников;
   9. Text textPreCE,textCountEnemy - количество оставшихся врагов;
   10. Texture livesTexture - текстура жизней;
   11. Text textPreLives - текст к картинкам с жизнями;
   12. vector<RectangleShape\*> vecBoxLives - вектор, который отображает кол-во оставшихся жизней;
   13. vector<RectangleShape\*>::iterator iterRS - итератор для вектора RectangleShape\*;
   14. vector<vector<StationaryObj\*>> listOfWalls - список стен(упорядоченный);
   15. vector<StationaryObj\*>::iterator iterLOW - итератор;
   16. StationaryObj\* ptrWall - указатель на стену;
   17. Base\* ptrBase - указатель на базу;
   18. MovingObj\* ptrMovObject - указатель на движущийся объект;
   19. vector<Bullet\*> listOfBullets - список всех пуль;
   20. vector<Bullet\*>::iterator iterLOB - итератор;
   21. Vector2f spawnPoint - вектор точки появления противников;
   22. vector<Enemy\*> vecOfEnemy - вектор врагов;
   23. vector<Enemy\*>::iterator iterVOE - итератор;
   24. Player\* ptrPlayer - указатель на объект игрока;
6. InputRecordScreen:
   1. унаследованные от класса Screens;
   2. RenderWindow\* win - указатель на окно;
   3. bool activeWin - активное окно;
   4. int score - счет;
   5. Text text - текст титульника вкладки;
   6. Text text2 - текст к счету;
   7. Text textScore - счет;
   8. ListOfRecords\* listOR - список рекордов;
   9. RectangleShape RecTextBox - поле ввода;
   10. Text inputLine - текст ввода;
   11. Text textQuestion - текст вопроса;
   12. RectangleShape buttonOk - кнопка ок;
   13. RectangleShape buttonCancel - кнопка отмены;
   14. Text textBO, textBC - текст к кнопкам;
   15. bool repetition - повторение;
   16. int choice - выбор;
7. ListOfRecords:
   1. vector<Record\*> listOfRecords - вектор рекордсменов;
   2. vector<Record\*>::iterator iter - итератор для вектора;
8. Map:
   1. static const int HEIGHT\_GAME\_BOX\_BY\_64 = 16 - высота игрового поля;
   2. static const int WIDTH\_GAME\_BOX\_BY\_64 = 24 - ширина игрового поля;
   3. Vector2f vectPosition - вектор позиции;
   4. Vector2f startPlayerPosition - вектор начальной позиции игрока;
   5. Vector2f spawnPoint - вектор позиции появления врагов;
   6. Base\* basePtr - указатель на базу;
   7. string mapLayout[HEIGHT\_GAME\_BOX\_BY\_64] - массив строк карты.
9. MapSelectionScreen:
   1. унаследованные от класса Screens;
   2. RenderWindow\* win - указатель на окно;
   3. bool activeWin - активное окно;
   4. int choice - выбор;
   5. int score - счет;
   6. Text title - текст титульника вкладки;
   7. Texture mapTexture1 - текстура первой карты;
   8. RectangleShape Map1 - кнопка карты 1;
   9. RectangleShape Map2 - кнопка карты 2;
   10. RectangleShape Map3 - кнопка карты 3;
   11. Text textMap1, textMap2, textMap3 - текст к кнопкам карт;
   12. RectangleShape buttonBack - кнопка назад;
   13. Text textBB - текст к кнопке назад;
   14. GameScreen\* gameScreen - указатель типа GameScreen;
   15. InputRecordScreen\* inputRS;
10. MetalWall:
    1. унаследованные от класса StationaryObj.
11. MovingObj:
    1. int numDirection - номер направления;
    2. int health - количество здоровья;
    3. унаследованные от класса Object.
12. Object:
    1. Vector2f size - размер объекта;
    2. float x - позиция по x;
    3. float y - позиция по y;
    4. RectangleShape obj - форма для объекта;
    5. Image objImage - картинка объекта;
    6. Texture objTexture - текстура объекта.
13. OutputRecordsScreen:
    1. унаследованные от класса Screens;
    2. RenderWindow\* win - указатель на окно;
    3. Text title - текст титульника вкладки;
    4. ListOfRecords\* listOR - список рекордов;
    5. vector<Text\*> listRecordsNum - вектор указателей типа Text для вывода номеров;
    6. vector<Text\*> listRecordsNick - вектор указателей типа Text для вывода никнеймов;
    7. vector<Text\*> listRecordsScore - вектор указателей типа Text для вывода очков;
    8. vector<Text\*>::iterator iter - итератор;
    9. Text textNumber, textNickname, textScore - названия;
    10. RectangleShape table - прямоугольник для таблицы;
    11. RectangleShape buttonBack - кнопка назад;
    12. Text textBB - текст к кнопке.
14. Player:
    1. унаследованные от класса MovingObj;
    2. Clock cooldown - время перезарядки.
15. Record:
    1. string nickname - никнейм;
    2. int score - счет;
16. Screens:
    1. Vector2f centralPosition - вектор для центровки текста;
    2. Vector2f sizeButton - ширина и длинна кнопки;
    3. Event ev - объект событий;
    4. Font font - шрифт;
17. StationaryObj:
    1. bool destroyable - разрушаемость объекта;
    2. int health - здоровье объекта;
    3. унаследованные от класса Object.
18. UserInterfaceScreen:
    1. унаследованные от класса Screens;
    2. RenderWindow\* win - указатель на окно;
    3. OutputRecordsScreen\* ptrOutputRS - указатель на экран вывода рекордов;
    4. MapSelectionScreen\* ptrMapSelectionScreen - указатель типа MapSelectionScreen;
    5. Text label, label2, creatorsText, textBS, textBR, textBE - текстовые поля;
    6. RectangleShape buttonStart, buttonRecords, buttonExit - кнопки;
    7. int choice = 1 - выбор;

#### 3.10 Список сообщений(методов).

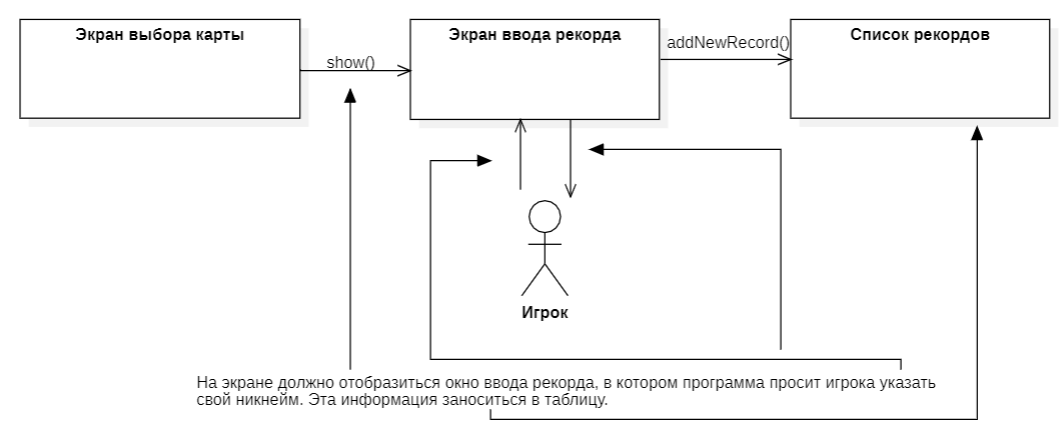


Рис.13. Глаголы варианта использования ”добавить запись в таблицу рекордов”.

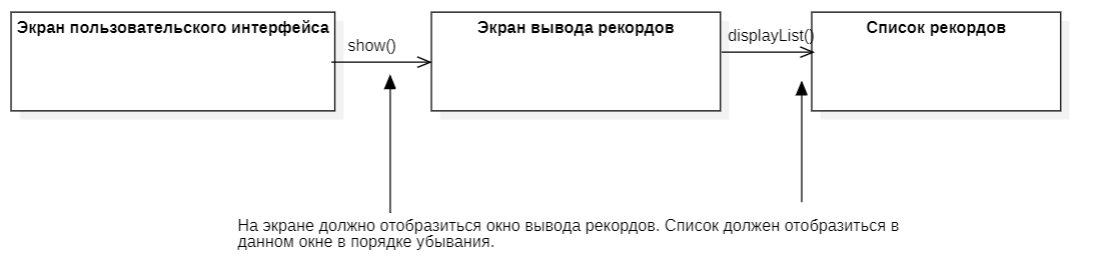


Рис.14. Глаголы варианта использования ”просмотреть таблицу рекордов”.

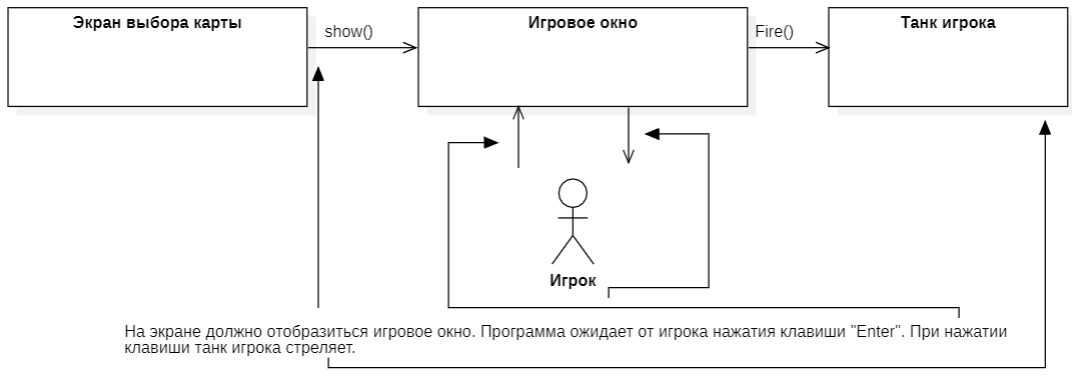


Рис.15. Глаголы варианта использования ”стрелять”.

1. Base:
   1. Base - конструктор;
   2. унаследованные от класса StationaryObj.
2. BrickWall:
   1. BrickWall - конструктор;
   2. changeSize - изменить размер объекта;
   3. унаследованные от класса StationaryObj.
3. Bullet:
   1. Bullet - конструктор;
   2. унаследованные от класса MovingObj.
4. Enemy:
   1. Enemy - конструктор;
   2. ChoiceOfEnemyAction - выбор действия у врага;
   3. CooldownRestart - перезапустить таймер;
   4. Fire - выстрелить;
   5. унаследованные от класса MovingObj.
5. GameScreen:
   1. унаследованные от класса Screens;
   2. GameScreen - конструктор;
   3. ~GameScreen - деструктор;
   4. show - показать окно;
   5. HandlingKeyPress - обработать клавиши;
   6. Update - обновить состояние объектов;
   7. getRowObject - получить строку объектов;
   8. isThereAWall - проверить на наличие стены;
   9. isThereAMovObject - проверить на наличие движущегося объекта;
   10. desctructorVectMovingObjects - очистить вектор движущихся объектов;
6. InputRecordScreen:
   1. унаследованные от класса Screens;
   2. InputRecordScreen - конструктор;
   3. ~InputRecordScreen - деструктор;
   4. show - показать окно;
   5. HandlingKeyPress - обработать клавиши;
7. ListOfRecords:
   1. ListOfRecords- конструктор;
   2. ~ListOfRecords - деструктор;
   3. addNewRecord - добавление нового рекорда;
   4. cheakRepetition - проверка на повторение имени;
   5. changeScore - изменить счет;
   6. getListRecord - получить список рекордов;
   7. sorting - отсортировать по убыванию;
   8. comparator - компаратор для сортировки;
8. Map:
   1. Map - конструктор;
   2. setElementsToVector - установить элементы в вектор;
   3. getStartPlayerPosition - получить стартовую позицию игрока;
   4. getSpawnPoint - получить позицию появления противников;
   5. getBasePtr - получить указатель базы;
9. MapSelectionScreen:
   1. унаследованные от класса Screens;
   2. MapSelectionScreen - конструктор;
   3. show - показать окно;
   4. HandlingKeyPress - обработать клавиши;
10. MetalWall:
    1. MetalWall - конструктор;
    2. унаследованные от класса StationaryObj.
11. MovingObj:
    1. MovingObj - конструктор;
    2. getNumDirection - получить номер направления;
    3. setNumDirection - установить номер направления;
    4. Move - переместить объект;
    5. changeHealth - изменить здоровье объекта;
    6. getHealth - получить здоровье объекта;
    7. унаследованные от класса Object.
12. Object:
    1. Object - конструктор;
    2. ~Object - деструктор;
    3. getX - получить координату по X;
    4. getY - получить координату по Y;
    5. getSize - получить размер;
    6. getRectangleShape - получить RectangleShape;
13. OutputRecordsScreen:
    1. OutputRecordsScreen - конструктор;
    2. ~OutputRecordsScreen - деструктор;
    3. show - показать окно;
    4. displayList - отобразить список;
    5. унаследованные от класса Screens;
14. Player:
    1. Player - конструктор;
    2. Fire - выстрелить;
    3. унаследованные от класса MovingObj;
15. Record:
    1. Record - конструктор;
    2. getNickname - получить никнейм;
    3. getScore - получить счет;
    4. changeScore - изменить счет;
16. Screens:
    1. Screens - конструктор;
    2. setCentralPos - установить центральное положение;
17. StationaryObj:
    1. StationaryObj - конструктор;
    2. changeHealth - изменить здоровье объекта;
    3. changeHealth - изменить здоровье объекта;
    4. getHealth - получить здоровье объекта;
    5. getTypeDestroy - получить тип разрушаемости;
    6. унаследованные от класса Object.
18. UserInterfaceScreen:
    1. UserInterfaceScreen - конструктор;
    2. ~UserInterfaceScreen -деструктор;
    3. interact - взаимодействовать(показать окно);
    4. changeColor - изменить цвет;
    5. унаследованные от класса Screens;

#### 3.11 Диаграмма классов.

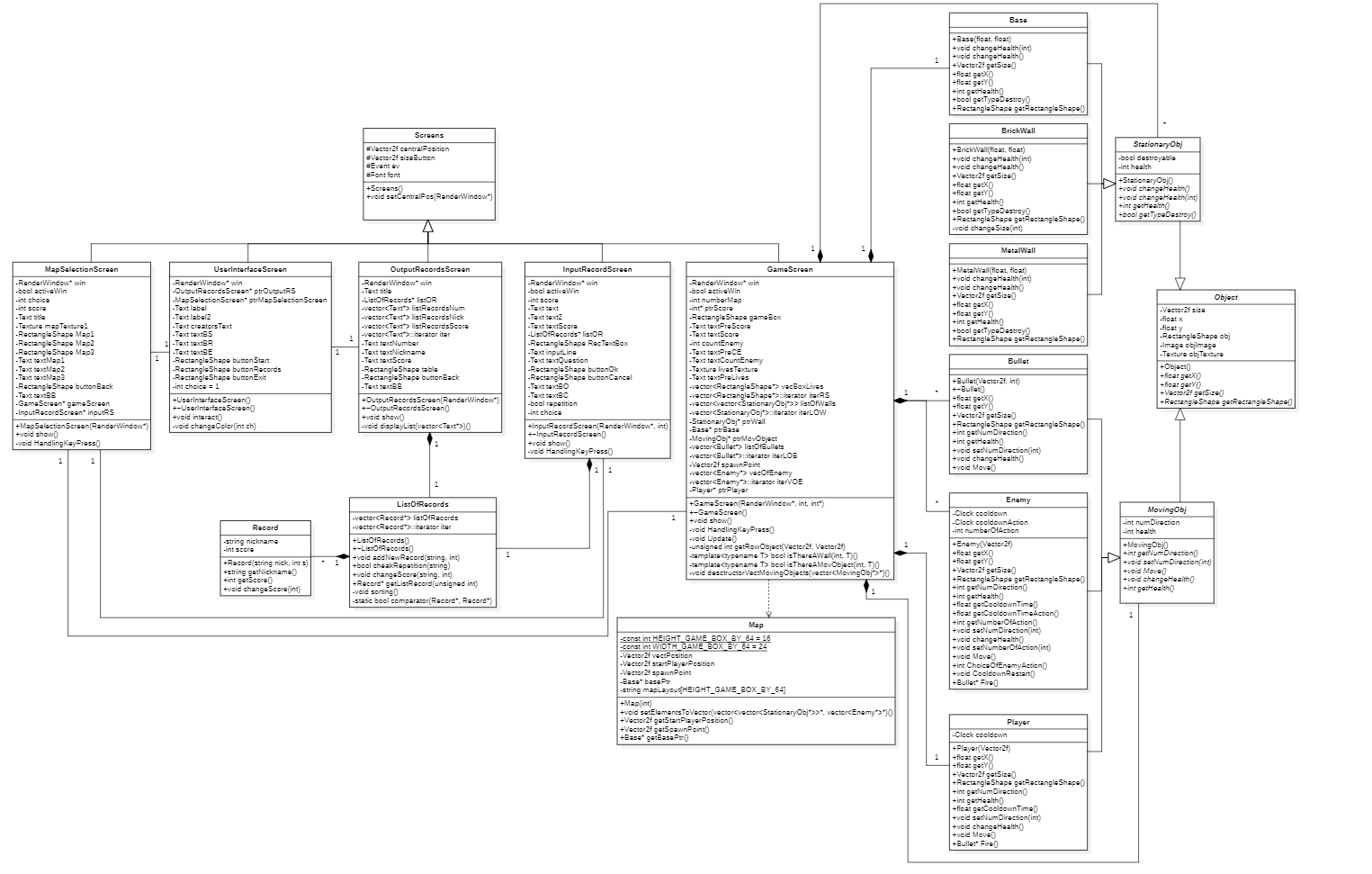


Рис.16. Диаграмма классов.

#### 3.12 Листинг заголовочных (.H) файлов.

**base.h:**

#ifndef BASE\_H

#define BASE\_H

#include "stationaryobj.h"

using namespace sf;

class Base: public StationaryObj

{

public:

Base(float, float);

void changeHealth(int) override;

void changeHealth() override;

Vector2f getSize() override;

float getX() override;

float getY() override;

int getHealth() override;

bool getTypeDestroy() override;

RectangleShape getRectangleShape() override;

};

#endif // BASE\_H

**brickwall.h:**

#ifndef BRICKWALL\_H

#define BRICKWALL\_H

#include "stationaryobj.h"

class BrickWall: public StationaryObj

{

public:

BrickWall(float, float);

void changeHealth(int) override;

void changeHealth() override;

Vector2f getSize() override;

float getX() override;

float getY() override;

int getHealth() override;

bool getTypeDestroy() override;

RectangleShape getRectangleShape() override;

private:

void changeSize(int);

};

#endif // BRICKWALL\_H

**bullet.h:**

#ifndef BULLET\_H

#define BULLET\_H

#include "movingobj.h"

class Bullet : public MovingObj

{

public:

Bullet(Vector2f,int);

~Bullet() override {}

float getX() override;

float getY() override;

Vector2f getSize() override;

RectangleShape getRectangleShape() override;

int getNumDirection() override;

int getHealth() override;

void setNumDirection(int) override {}

void changeHealth() override {}

void Move() override;

};

#endif // BULLET\_H

**enemy.h:**

#ifndef ENEMY\_H

#define ENEMY\_H

#include "movingobj.h"

#include "bullet.h"

#include <stdlib.h>

#include <windows.h>

using namespace std;

class Enemy : public MovingObj

{

private:

Clock cooldown;

Clock cooldownAction;

int numberOfAction;

public:

Enemy(Vector2f);

float getX() override;

float getY() override;

Vector2f getSize() override;

RectangleShape getRectangleShape() override;

int getNumDirection() override;

int getHealth() override;

float getCooldownTime();

float getCooldownTimeAction();

int getNumberOfAction();

void setNumDirection(int) override;

void changeHealth() override;

void setNumberOfAction(int);

void Move() override;

int ChoiceOfEnemyAction();

void CooldownRestart();

Bullet\* Fire();

};

#endif // ENEMY\_H

**gamescreen.h:**

#ifndef GAMESCREEN\_H

#define GAMESCREEN\_H

#include "screens.h"

#include <vector>

#include "map.h"

#include "object.h"

#include "player.h"

#include "bullet.h"

#include "base.h"

#include "enemy.h"

#include <stdlib.h>

#include <windows.h>

using namespace sf;

using namespace std;

class GameScreen: Screens

{

private:

RenderWindow\* win;

bool activeWin;

int numberMap;

int\* ptrScore;

RectangleShape gameBox;

Text textPreScore,textScore;

int countEnemy;

Text textPreCE,textCountEnemy;

Texture livesTexture;

Text textPreLives;

vector<RectangleShape\*> vecBoxLives;

vector<RectangleShape\*>::iterator iterRS;

vector<vector<StationaryObj\*>> listOfWalls;

vector<StationaryObj\*>::iterator iterLOW;

StationaryObj\* ptrWall;

Base\* ptrBase;

MovingObj\* ptrMovObject;

vector<Bullet\*> listOfBullets;

vector<Bullet\*>::iterator iterLOB;

Vector2f spawnPoint;

vector<Enemy\*> vecOfEnemy;

vector<Enemy\*>::iterator iterVOE;

Player\* ptrPlayer;

public:

GameScreen(RenderWindow\*, int, int\*);

~GameScreen();

void show();

private:

void HandlingKeyPress();

void Update();

unsigned int getRowObject(Vector2f, Vector2f );

template<typename T> bool isThereAWall(int, T);

template<typename T> bool isThereAMovObject(int, T);

void desctructorVectMovingObjects(vector<MovingObj\*>\*);

};

#endif // GAMESCREEN\_H

**inputrecordscreen.h:**

#ifndef INPUTRECORDSCREEN\_H

#define INPUTRECORDSCREEN\_H

#include "screens.h"

#include "listofrecords.h"

#include <string>

using namespace sf;

class InputRecordScreen: public Screens

{

private:

RenderWindow\* win;

bool activeWin;

int score;

Text text;

Text text2;

Text textScore;

ListOfRecords\* listOR;

RectangleShape RecTextBox;

Text inputLine;

Text textQuestion;

RectangleShape buttonOk;

RectangleShape buttonCancel;

Text textBO, textBC;

bool repetition;

int choice;

public:

InputRecordScreen(RenderWindow\*, int);

~InputRecordScreen();

void show();

private:

void HandlingKeyPress();

};

#endif // INPUTRECORDSCREEN\_H

**listofrecords.h:**

#ifndef LISTOFRECORDS\_H

#define LISTOFRECORDS\_H

#include <vector>

#include <record.h>

#include <algorithm>

#include <iostream>

#include <fstream>

class ListOfRecords

{

private:

vector<Record\*> listOfRecords;

vector<Record\*>::iterator iter;

public:

ListOfRecords();

~ListOfRecords();

void addNewRecord(string, int);

bool cheakRepetition(string);

void changeScore(string, int);

Record\* getListRecord(unsigned int);

private:

void sorting();

static bool comparator(Record\*, Record\*);

};

#endif // LISTOFRECORDS\_H

**map.h:**

#ifndef MAP\_H

#define MAP\_H

#include <string>

#include <fstream>

#include <iostream>

#include <vector>

#include "SFML/Graphics.hpp"

#include "brickwall.h"

#include "metalwall.h"

#include "enemy.h"

#include "base.h"

using namespace std;

using namespace sf;

class Map

{

private:

static const int HEIGHT\_GAME\_BOX\_BY\_64 = 16;

static const int WIDTH\_GAME\_BOX\_BY\_64 = 24;

Vector2f vectPosition;

Vector2f startPlayerPosition;

Vector2f spawnPoint;

Base\* basePtr;

string mapLayout[HEIGHT\_GAME\_BOX\_BY\_64];

public:

Map(int);

void setElementsToVector(vector<vector<StationaryObj\*>>\*, vector<Enemy\*>\*);

Vector2f getStartPlayerPosition();

Vector2f getSpawnPoint();

Base\* getBasePtr();

};

#endif // MAP\_H

**mapselectionscreen.h:**

#ifndef MAPSELECTIONSCREEN\_H

#define MAPSELECTIONSCREEN\_H

#include "screens.h"

#include "gamescreen.h"

#include "inputrecordscreen.h"

class MapSelectionScreen: public Screens

{

private:

RenderWindow\* win; // окно

bool activeWin;

int choice;

int score;

Text title;

Texture mapTexture1;

RectangleShape Map1;

RectangleShape Map2;

RectangleShape Map3;

Text textMap1, textMap2, textMap3;

RectangleShape buttonBack;

Text textBB;

GameScreen\* gameScreen;

InputRecordScreen\* inputRS;

public:

MapSelectionScreen(RenderWindow\*);

void show();

private:

void HandlingKeyPress();

};

#endif // MAPSELECTIONSCREEN\_H

**metalwall.h:**

#ifndef METALWALL\_H

#define METALWALL\_H

#include "stationaryobj.h"

class MetalWall: public StationaryObj

{

public:

MetalWall(float, float);

void changeHealth(int) override {}

void changeHealth() override {}

Vector2f getSize() override;

float getX() override;

float getY() override;

int getHealth() override;

bool getTypeDestroy() override;

RectangleShape getRectangleShape() override;

};

#endif // METALWALL\_H

**movingobj.h:**

#ifndef MOVINGOBJ\_H

#define MOVINGOBJ\_H

#include "object.h"

class MovingObj: public Object

{

protected:

int numDirection;

int health;

public:

MovingObj();

int virtual getNumDirection() = 0;

void virtual setNumDirection(int) = 0;

void virtual Move() = 0;

void virtual changeHealth() = 0;

int virtual getHealth() = 0;

};

#endif // MOVINGOBJ\_H

**object.h:**

#ifndef OBJECT\_H

#define OBJECT\_H

#include "SFML/Graphics.hpp"

using namespace sf;

class Object

{

protected:

Vector2f size;

float x;

float y;

RectangleShape obj;

Image objImage;

Texture objTexture;

public:

Object() {}

virtual ~Object() {}

float virtual getX() = 0;

float virtual getY() = 0;

Vector2f virtual getSize() = 0;

RectangleShape virtual getRectangleShape() = 0;

};

#endif // OBJECT\_H

**outputrecordscreen.h:**

#ifndef OUTPUTRECORDSSCREEN\_H

#define OUTPUTRECORDSSCREEN\_H

#include <screens.h>

#include <listofrecords.h>

#include <vector>

using namespace sf;

class OutputRecordsScreen: public Screens

{

private:

RenderWindow\* win;

Text title;

ListOfRecords\* listOR;

vector<Text\*> listRecordsNum;

vector<Text\*> listRecordsNick;

vector<Text\*> listRecordsScore;

vector<Text\*>::iterator iter;

Text textNumber, textNickname, textScore;

RectangleShape table;

RectangleShape buttonBack;

Text textBB;

public:

OutputRecordsScreen(RenderWindow\*);

~OutputRecordsScreen();

void show();

private:

void displayList(vector<Text\*>);

};

#endif // OUTPUTRECORDSSCREEN\_H

**player.h:**

#ifndef PLAYER\_H

#define PLAYER\_H

#include "movingobj.h"

#include "bullet.h"

#include "SFML/System/Clock.hpp"

class Player: public MovingObj

{

private:

Clock cooldown;

public:

Player(Vector2f);

float getX() override;

float getY() override;

Vector2f getSize() override;

RectangleShape getRectangleShape() override;

int getNumDirection() override;

int getHealth() override;

float getCooldownTime();

void setNumDirection(int) override;

void changeHealth() override;

void Move() override;

Bullet\* Fire();

};

#endif // PLAYER\_H

**record.h:**

#ifndef RECORD\_H

#define RECORD\_H

#include <iostream>

using namespace std;

class Record

{

private:

string nickname;

int score;

public:

Record(string nick, int s):nickname(nick), score(s) {}

string getNickname();

int getScore();

void changeScore(int);

};

#endif // RECORD\_H

**screens.h:**

#ifndef SCREENS\_H

#define SCREENS\_H

#include <SFML/Graphics.hpp>

using namespace sf;

class Screens

{

protected:

Vector2f centralPosition;

Vector2f sizeButton;

Event ev;

Font font;

public:

Screens();

void setCentralPos(RenderWindow\*);

};

#endif // SCREENS\_H

**stationaryobj.h:**

#ifndef STATIONARYOBJ\_H

#define STATIONARYOBJ\_H

#include "object.h"

class StationaryObj: public Object

{

protected:

bool destroyable;

int health;

public:

StationaryObj() {}

void virtual changeHealth() = 0;

void virtual changeHealth(int) = 0;

int virtual getHealth() = 0;

bool virtual getTypeDestroy() = 0;

};

#endif // STATIONARYOBJ\_H

**userinterfacescreen.h:**

#ifndef USERINTERFACESCREEN\_H

#define USERINTERFACESCREEN\_H

#include "outputrecordsscreen.h"

#include "mapselectionscreen.h"

#include "screens.h"

#include <iostream>

using namespace sf;

using namespace std;

class UserInterfaceScreen: public Screens

{

private:

RenderWindow\* win;

OutputRecordsScreen\* ptrOutputRS;

MapSelectionScreen\* ptrMapSelectionScreen;

Text label, label2, creatorsText, textBS, textBR, textBE;

RectangleShape buttonStart, buttonRecords, buttonExit;

int choice = 1;

public:

UserInterfaceScreen();

~UserInterfaceScreen() {}

void interact();

private:

void changeColor(int ch);

};

#endif // USERINTERFACESCREEN\_H

#### 3.13 Листинг исходных файлов (.cpp).

**base.cpp:**

#include "base.h"

Base::Base(float x, float y)

{

destroyable = true;

size = Vector2f(64,64);

health = 3;

objTexture.loadFromFile("img/base.png");

obj = RectangleShape(size);

obj.setTexture(&objTexture);

obj.setPosition(x, y);

}

void Base::changeHealth(){

health -= 1;

}

void Base::changeHealth(int i){

health -= 1;

}

float Base::getX(){

return obj.getPosition().x;

}

float Base::getY(){

return obj.getPosition().y;

}

Vector2f Base::getSize(){

return size;

}

int Base::getHealth(){

return health;

}

bool Base::getTypeDestroy(){

return destroyable;

}

RectangleShape Base::getRectangleShape(){

return obj;

}

**brickwall.cpp:**

#include "brickwall.h"

BrickWall::BrickWall(float X, float Y)

{

this->x = X;

this->y = Y;

destroyable = true;

size = Vector2f(64,64);

health = 4;

objImage.loadFromFile("img/brickWall.png");

objTexture.loadFromImage(objImage);

obj = RectangleShape(size);

obj.setTexture(&objTexture);

obj.setPosition(this->x, this->y);

}

int BrickWall::getHealth(){

return health;

}

bool BrickWall::getTypeDestroy(){

return destroyable;

}

void BrickWall::changeSize(int numDiirection){

int numberOfFile;

switch (numDiirection) {

case 1: // сверху

size.y -= 16;

numberOfFile = size.y;

objImage.loadFromFile("img/brickWall"+ std::to\_string(numberOfFile) +"H.png");

objTexture.loadFromImage(objImage);

obj.setTexture(&objTexture, true);

obj.setSize(size);

break;

case 2: // снизу

size.y -= 16;

numberOfFile = size.y;

objImage.loadFromFile("img/brickWall"+ std::to\_string(numberOfFile) +"H.png");

objTexture.loadFromImage(objImage);

obj.setTexture(&objTexture, true);

obj.setSize(size);

obj.setPosition(obj.getPosition().x, obj.getPosition().y+16);

break;

case 3: // слева

size.x -= 16;

numberOfFile = size.x;

objImage.loadFromFile("img/brickWall"+ std::to\_string(numberOfFile) +"W.png");

objTexture.loadFromImage(objImage);

obj.setTexture(&objTexture, true);

obj.setSize(size);

break;

case 4: // справа

size.x -= 16;

numberOfFile = size.x;

objImage.loadFromFile("img/brickWall"+ std::to\_string(numberOfFile) +"W.png");

objTexture.loadFromImage(objImage);

obj.setTexture(&objTexture, true);

obj.setSize(size);

obj.setPosition(obj.getPosition().x+16, obj.getPosition().y);

break;

}

}

void BrickWall::changeHealth(int numDiirection){

health -= 1;

if(health != 0)

changeSize(numDiirection);

}

void BrickWall::changeHealth(){

}

float BrickWall::getX(){

return obj.getPosition().x;

}

float BrickWall::getY(){

return obj.getPosition().y;

}

Vector2f BrickWall::getSize(){

return size;

}

RectangleShape BrickWall::getRectangleShape(){

return obj;

}

**bullet.cpp:**

#include "bullet.h"

Bullet::Bullet(Vector2f startPosition, int numDir)

{

this->x = startPosition.x;

this->y = startPosition.y;

numDirection = numDir;

health = 1;

switch (numDirection) {

case 1:

size = Vector2f(12,16);

objImage.loadFromFile("img/bulletUp.png");

break;

case 2:

size = Vector2f(12,16);

objImage.loadFromFile("img/bulletDown.png");

break;

case 3:

size = Vector2f(16,12);

objImage.loadFromFile("img/bulletLeft.png");

break;

case 4:

size = Vector2f(16,12);

objImage.loadFromFile("img/bulletRight.png");

break;

}

objImage.createMaskFromColor(Color(0,0,1));

objTexture.loadFromImage(objImage);

obj = RectangleShape(size);

obj.setTexture(&objTexture);

obj.setPosition(this->x, this->y);

}

float Bullet::getX(){

return obj.getPosition().x;

}

float Bullet::getY(){

return obj.getPosition().y;

}

Vector2f Bullet::getSize(){

return size;

}

int Bullet::getHealth(){

return health;

}

RectangleShape Bullet::getRectangleShape(){

return obj;

}

int Bullet::getNumDirection(){

return numDirection;

}

void Bullet::Move(){

if(numDirection == 1)

obj.move(0,-2);//вперед

else if(numDirection == 2)

obj.move(0,2);//вниз

else if(numDirection == 3)

obj.move(-2,0);//влево

else if(numDirection == 4)

obj.move(2,0);//вправо

}

**enemy.cpp:**

#include "enemy.h"

using namespace std;

Enemy::Enemy(Vector2f startPosition)

{

this->x = startPosition.x;

this->y = startPosition.y;

numDirection = 1;

size = Vector2f(64,64);

health = 3;

objImage.loadFromFile("img/tankEnemyUp.png");

objImage.createMaskFromColor(Color(0,0,1));

objTexture.loadFromImage(objImage);

obj = RectangleShape(size);

obj.setTexture(&objTexture);

obj.setPosition(this->x, this->y);

cooldown.restart();

cooldownAction.restart();

}

int Enemy::getHealth(){

return health;

}

float Enemy::getCooldownTime(){

return cooldown.getElapsedTime().asSeconds();

}

float Enemy::getCooldownTimeAction(){

return cooldownAction.getElapsedTime().asSeconds();

}

void Enemy::changeHealth(){

health--;

}

float Enemy::getX(){

return obj.getPosition().x;

}

float Enemy::getY(){

return obj.getPosition().y;

}

Vector2f Enemy::getSize(){

return size;

}

RectangleShape Enemy::getRectangleShape(){

return obj;

}

int Enemy::getNumDirection(){

return numDirection;

}

void Enemy::setNumDirection(int numDir){

numDirection = numDir;

if(numDirection == 1){

objImage.loadFromFile("img/tankEnemyUp.png");

objImage.createMaskFromColor(Color(0,0,1));

objTexture.loadFromImage(objImage);

} else if(numDirection == 2){

objImage.loadFromFile("img/tankEnemyDown.png");

objImage.createMaskFromColor(Color(0,0,1));

objTexture.loadFromImage(objImage);

} else if(numDirection == 3){

objImage.loadFromFile("img/tankEnemyLeft.png");

objImage.createMaskFromColor(Color(0,0,1));

objTexture.loadFromImage(objImage);

} else if(numDirection == 4){

objImage.loadFromFile("img/tankEnemyRight.png");

objImage.createMaskFromColor(Color(0,0,1));

objTexture.loadFromImage(objImage);

}

}

void Enemy::Move(){

if(numDirection == 1)

obj.move(0,-1);//вперед

else if(numDirection == 2)

obj.move(0,1);//вниз

else if(numDirection == 3)

obj.move(-1,0);//влево

else if(numDirection == 4)

obj.move(1,0);//вправо

}

Bullet\* Enemy::Fire(){

cooldown.restart();

if(numDirection == 1)

return new Bullet(Vector2f(obj.getPosition().x + (size.x / 2), obj.getPosition().y - 1), numDirection);

else if(numDirection == 2)

return new Bullet(Vector2f(obj.getPosition().x + (size.x / 2), obj.getPosition().y + size.y + 1), numDirection);

else if(numDirection == 3)

return new Bullet(Vector2f(obj.getPosition().x - 1, obj.getPosition().y + (size.y / 2)), numDirection);

else if(numDirection == 4)

return new Bullet(Vector2f(obj.getPosition().x + size.x + 1, obj.getPosition().y + (size.y / 2)), numDirection);

return NULL;

}

int Enemy::getNumberOfAction(){

return numberOfAction;

}

void Enemy::setNumberOfAction(int num){

numberOfAction = num;

}

void Enemy::CooldownRestart(){

cooldownAction.restart();

}

int Enemy::ChoiceOfEnemyAction(){

SYSTEMTIME st;

int randNum = numberOfAction;

if(getCooldownTimeAction() > 1.0f){

GetLocalTime(&st);

srand(st.wMilliseconds);

randNum = rand() % 5;

setNumberOfAction(randNum);

CooldownRestart();

}

return randNum;

}

**gamescreen.cpp:**

#include "gamescreen.h"

using namespace sf;

using namespace std;

GameScreen::GameScreen(RenderWindow\* window, int choiceMap, int\* score): win(window), numberMap(choiceMap), ptrScore(score){

activeWin = true;

countEnemy = 8;

setCentralPos(win);

font.loadFromFile("Font/Pixeled.ttf");

livesTexture.loadFromFile("img/heart.png");

gameBox = RectangleShape(Vector2f(1472,1024));

gameBox.setOutlineThickness(3);

gameBox.setOutlineColor(Color::Red);

gameBox.setFillColor(Color::Black);

gameBox.setPosition(3, 3);

textPreScore = Text("SCORE:", font);

textPreScore.setCharacterSize(30);

textPreScore.setPosition((win->getSize().x + gameBox.getGlobalBounds().width + 6 - textPreScore.getGlobalBounds().width)/2, 100);

textScore = Text("0", font);

textScore.setCharacterSize(30);

textScore.setFillColor(Color::Green);

textScore.setPosition((win->getSize().x + gameBox.getGlobalBounds().width + 6 - textScore.getGlobalBounds().width)/2,

130+ textPreScore.getGlobalBounds().height);

textPreCE = Text("REMAINING ENEMIES:", font);

textPreCE.setCharacterSize(22);

textPreCE.setPosition((win->getSize().x + gameBox.getGlobalBounds().width + 6 - textPreCE.getGlobalBounds().width)/2,

textScore.getPosition().y + 100);

textCountEnemy = Text(String(to\_string(countEnemy)), font);

textCountEnemy.setCharacterSize(30);

textCountEnemy.setFillColor(Color::Red);

textCountEnemy.setPosition((win->getSize().x + gameBox.getGlobalBounds().width + 6 - textCountEnemy.getGlobalBounds().width)/2,

textPreCE.getPosition().y + 50);

textPreLives = Text("REMAINING LIVES:", font);

textPreLives.setCharacterSize(22);

textPreLives.setPosition((win->getSize().x + gameBox.getGlobalBounds().width + 6 - textPreLives.getGlobalBounds().width)/2,

textCountEnemy.getPosition().y + 100);

vecBoxLives.push\_back(new RectangleShape(Vector2f(64,64)));

vecBoxLives[0]->setTexture(&livesTexture);

vecBoxLives[0]->setPosition((win->getSize().x + gameBox.getGlobalBounds().width + 6 - vecBoxLives[0]->getGlobalBounds().width)/2 - 80,

textPreLives.getPosition().y + 50);

vecBoxLives.push\_back(new RectangleShape(Vector2f(64,64)));

vecBoxLives[1]->setTexture(&livesTexture);

vecBoxLives[1]->setPosition((win->getSize().x + gameBox.getGlobalBounds().width + 6 - vecBoxLives[1]->getGlobalBounds().width)/2,

textPreLives.getPosition().y + 50);

vecBoxLives.push\_back(new RectangleShape(Vector2f(64,64)));

vecBoxLives[2]->setTexture(&livesTexture);

vecBoxLives[2]->setPosition((win->getSize().x + gameBox.getGlobalBounds().width + 6 - vecBoxLives[2]->getGlobalBounds().width)/2 + 80,

textPreLives.getPosition().y + 50);

Map\* mapLoad = new Map(choiceMap);

mapLoad->setElementsToVector(&listOfWalls, &vecOfEnemy);

ptrPlayer = new Player(mapLoad->getStartPlayerPosition());

ptrPlayer->setNumDirection(1);

ptrBase = mapLoad->getBasePtr();

spawnPoint = mapLoad->getSpawnPoint();

delete mapLoad;

}

GameScreen::~GameScreen(){

while (!vecBoxLives.empty())

{

iterRS = vecBoxLives.begin();

delete \*iterRS;

vecBoxLives.erase(iterRS);

}

while(!listOfWalls.empty()){

while (!listOfWalls[0].empty())

{

iterLOW = listOfWalls[0].begin();

delete \*iterLOW;

listOfWalls[0].erase(iterLOW);

}

listOfWalls.erase(listOfWalls.begin());

}

while (!vecOfEnemy.empty())

{

iterVOE = vecOfEnemy.begin();

delete \*iterVOE;

vecOfEnemy.erase(iterVOE);

}

while (!listOfBullets.empty())

{

iterLOB = listOfBullets.begin();

delete \*iterLOB;

listOfBullets.erase(iterLOB);

}

delete ptrPlayer;

}

void GameScreen::show(){

while (activeWin) {

while (win->pollEvent(ev)) {

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

win->close();

else if(ev.type == Event::KeyPressed){

HandlingKeyPress();

}

else if(ev.type == Event::KeyReleased){

if(ev.key.code == Keyboard::Enter && ptrPlayer->getCooldownTime() > 1)

listOfBullets.push\_back(ptrPlayer->Fire());

}

}

win->clear();

win->draw(gameBox);

win->draw(textPreScore);

win->draw(textScore);

win->draw(textPreCE);

win->draw(textCountEnemy);

win->draw(textPreLives);

Update();

win->display();

}

}

**gamescreencontrol.cpp:**

#include "gamescreen.h"

using namespace sf;

using namespace std;

unsigned int GameScreen::getRowObject(Vector2f objectSize, Vector2f objectPos)

{

for(unsigned int i = 0; i < listOfWalls.size(); i++){

if((i\*64+3) <= (objectPos.y + objectSize.y / 2) && (objectPos.y + objectSize.y / 2) <= ((i+1)\*64+3))

return i;

}

return 0;

}

void GameScreen::desctructorVectMovingObjects(vector<MovingObj\*>\* vecMovingObjects){

vector<MovingObj\*>::iterator iterVect;

iterVect = (\*vecMovingObjects).begin();

while (iterVect != (\*vecMovingObjects).end())

{

(\*vecMovingObjects).erase(iterVect);

}

}

template<typename T> bool GameScreen::isThereAWall(int numDirection, T checkedObject){

Vector2f objectPos = Vector2f(checkedObject->getX(), checkedObject->getY());

Vector2f objectSize = checkedObject->getSize();

Vector2f checkedObjectPos;

unsigned int rowObject;

float func;

rowObject = getRowObject(objectSize, objectPos);

if(numDirection == 1 || numDirection == 2){

for(unsigned short i = 0; i < 2; i++){

iterLOW = listOfWalls[rowObject].begin();

while (iterLOW != listOfWalls[rowObject].end())

{

checkedObjectPos.x = (\*\*iterLOW).getX();

checkedObjectPos.y = (\*\*iterLOW).getY();

if((checkedObjectPos.x <= objectPos.x && (checkedObjectPos.x + (\*\*iterLOW).getSize().x) > objectPos.x) ||

(checkedObjectPos.x < (objectPos.x + objectSize.x) && (checkedObjectPos.x + (\*\*iterLOW).getSize().x) >= (objectPos.x + objectSize.x)))

{

if(numDirection == 1){

func = objectPos.y - (checkedObjectPos.y + (\*\*iterLOW).getSize().y);

if(func <= 0 && func > -(\*\*iterLOW).getSize().x){

ptrWall = \*iterLOW;

return true;

}

} else {

func = checkedObjectPos.y - (objectPos.y + objectSize.y);

if( func <= 0){

ptrWall = \*iterLOW;

return true;

}

}

}

iterLOW++;

}

numDirection == 1 ? rowObject-- : rowObject++;

}

} else if(numDirection == 3 || numDirection == 4){

for(unsigned short i = rowObject-1; i < rowObject +2; i++){

iterLOW = listOfWalls[i].begin();

while (iterLOW != listOfWalls[i].end())

{

checkedObjectPos.x = (\*\*iterLOW).getX();

checkedObjectPos.y = (\*\*iterLOW).getY();

if((checkedObjectPos.y <= objectPos.y && (checkedObjectPos.y + (\*\*iterLOW).getSize().y) > objectPos.y) ||

(checkedObjectPos.y < (objectPos.y + objectSize.y) && (checkedObjectPos.y + (\*\*iterLOW).getSize().y) >= (objectPos.y + objectSize.y)))

{

if(numDirection == 3)

func = objectPos.x - (checkedObjectPos.x + (\*\*iterLOW).getSize().x);

else

func = checkedObjectPos.x - (objectPos.x + objectSize.x);

if(func <= 0 && func > -(\*\*iterLOW).getSize().x) {

ptrWall = \*iterLOW;

return true;

}

}

iterLOW++;

}

}

}

return false;

}

template<typename T> bool GameScreen::isThereAMovObject(int numDirection, T checkedObject){

Vector2f objectPos = Vector2f(checkedObject->getX(), checkedObject->getY());

Vector2f objectSize = checkedObject->getSize();

Vector2f checkedObjectPos; Vector2f checkedobjectSize;

vector<MovingObj\*> buffObject;

vector<MovingObj\*>::iterator iterBO;

float func;

buffObject.push\_back(ptrPlayer);

for(vector<Enemy\*>::iterator iter = vecOfEnemy.begin(); iter != vecOfEnemy.end(); iter++)

buffObject.push\_back(\*iter);

for(vector<Bullet\*>::iterator iterB = listOfBullets.begin(); iterB != listOfBullets.end(); iterB++)

buffObject.push\_back(\*iterB);

iterBO = buffObject.begin();

while (iterBO != buffObject.end())

{

if((\*iterBO) == checkedObject){

buffObject.erase(iterBO);

break;

}

iterBO++;

}

if(numDirection == 1 || numDirection == 2){

iterBO = buffObject.begin();

while (iterBO != buffObject.end())

{

checkedObjectPos = Vector2f((\*\*iterBO).getX(),(\*\*iterBO).getY());

checkedobjectSize = (\*\*iterBO).getSize();

if((checkedObjectPos.x <= objectPos.x && (checkedObjectPos.x + checkedobjectSize.x) > objectPos.x) ||

(checkedObjectPos.x < (objectPos.x + objectSize.x) && (checkedObjectPos.x + checkedobjectSize.x) >= (objectPos.x + objectSize.x))){

if(numDirection == 1){

func = objectPos.y - (checkedObjectPos.y + checkedobjectSize.y);

if(func <= 0 && func > -checkedobjectSize.y){

ptrMovObject = \*iterBO;

return true;

}

} else {

func = checkedObjectPos.y - (objectPos.y + objectSize.y);

if( func == 0 ){

ptrMovObject = \*iterBO;

return true;

}

}

}

iterBO++;

}

} else if(numDirection == 3 || numDirection == 4){

iterBO = buffObject.begin();

while (iterBO != buffObject.end())

{

checkedObjectPos = Vector2f((\*\*iterBO).getX(),(\*\*iterBO).getY());

checkedobjectSize = (\*\*iterBO).getSize();

if((checkedObjectPos.y <= objectPos.y && (checkedObjectPos.y + checkedobjectSize.y) > objectPos.y) ||

(checkedObjectPos.y < (objectPos.y + objectSize.y) && (checkedObjectPos.y + checkedobjectSize.y) >= (objectPos.y + objectSize.y)))

{

if(numDirection == 3)

func = objectPos.x - (checkedObjectPos.x + checkedobjectSize.x);

else

func = checkedObjectPos.x - (objectPos.x + objectSize.x);

if(func <= 0 && func > -checkedobjectSize.x) {

ptrMovObject = \*iterBO;

return true;

}

}

iterBO++;

}

}

return false;

}

void GameScreen::Update(){

iterRS = vecBoxLives.begin();

while (iterRS != vecBoxLives.end())

{

if(ptrPlayer->getHealth() < vecBoxLives.size() || ptrBase->getHealth() < vecBoxLives.size()){

delete (\*iterRS);

vecBoxLives.erase(iterRS);

} else

win->draw(\*\*iterRS);

iterRS++;

}

for(unsigned int i = 0; i < listOfWalls.size(); i++){

iterLOW = listOfWalls[i].begin();

while (iterLOW != listOfWalls[i].end())

{

if((\*\*iterLOW).getHealth() != 0)

win->draw((\*\*iterLOW).getRectangleShape());

else{

if(\*iterLOW != ptrBase)

delete \*iterLOW;

listOfWalls[i].erase(iterLOW);

}

iterLOW++;

}

}

win->draw((\*ptrPlayer).getRectangleShape());

iterVOE = vecOfEnemy.begin();

while (iterVOE != vecOfEnemy.end())

{

switch ((\*iterVOE)->ChoiceOfEnemyAction()) {

case 0:

(\*iterVOE)->setNumDirection(1);

if(!isThereAWall(1, (\*iterVOE)) && !isThereAMovObject(1, (\*iterVOE)))

(\*iterVOE)->Move();

else

(\*iterVOE)->setNumberOfAction(1);

break;

case 1:

(\*iterVOE)->setNumDirection(2);

if(!isThereAWall(2,(\*iterVOE)) && !isThereAMovObject(2, (\*iterVOE)))

(\*iterVOE)->Move();

else

(\*iterVOE)->setNumberOfAction(0);

break;

case 2:

(\*iterVOE)->setNumDirection(3);

if(!isThereAWall(3,(\*iterVOE)) && !isThereAMovObject(3, (\*iterVOE)))

(\*iterVOE)->Move();

else

(\*iterVOE)->setNumberOfAction(3);

break;

case 3:

(\*iterVOE)->setNumDirection(4);

if(!isThereAWall(4,(\*iterVOE)) && !isThereAMovObject(4, (\*iterVOE)))

(\*iterVOE)->Move();

else

(\*iterVOE)->setNumberOfAction(2);

break;

case 4:

if((\*iterVOE)->getCooldownTime() > 1)

listOfBullets.push\_back((\*iterVOE)->Fire());

break;

}

if((\*\*iterVOE).getHealth() != 0)

win->draw((\*iterVOE)->getRectangleShape());

else{

delete \*iterVOE;

vecOfEnemy.erase(iterVOE);

\*ptrScore += 50;

textCountEnemy.setString(String(to\_string(--countEnemy)));

textScore.setString(String(to\_string(\*ptrScore)));

textScore.setPosition((win->getSize().x + gameBox.getGlobalBounds().width + 6 - textScore.getGlobalBounds().width)/2,

130+ textPreScore.getGlobalBounds().height);

continue;

}

iterVOE++;

}

if(vecOfEnemy.size() < 2 && countEnemy >= 2){

vecOfEnemy.push\_back(new Enemy(spawnPoint));

iterVOE = vecOfEnemy.begin();

iterVOE++;

if(isThereAMovObject(4, \*iterVOE) || isThereAMovObject(3, \*iterVOE) || isThereAMovObject(2, \*iterVOE)|| isThereAMovObject(1, \*iterVOE)){

delete \*iterVOE;

vecOfEnemy.erase(iterVOE);

}

}

iterLOB = listOfBullets.begin();

while (iterLOB != listOfBullets.end())

{

if(isThereAWall((\*iterLOB)->getNumDirection(), (\*iterLOB))){

ptrWall->changeHealth((\*iterLOB)->getNumDirection());

delete \*iterLOB;

listOfBullets.erase(iterLOB);

continue;

} else if(isThereAMovObject((\*iterLOB)->getNumDirection(), (\*iterLOB))){

ptrMovObject->changeHealth();

delete \*iterLOB;

listOfBullets.erase(iterLOB);

continue;

} else {

(\*iterLOB)->Move();

win->draw((\*iterLOB)->getRectangleShape());

}

iterLOB++;

}

if(ptrPlayer->getHealth() == 0 || ptrBase->getHealth() == 0 || countEnemy == 0)

activeWin = false;

}

void GameScreen::HandlingKeyPress(){

switch (ev.key.code) {

case Keyboard::Up:

ptrPlayer->setNumDirection(1);

if(!isThereAWall(1, ptrPlayer) && !isThereAMovObject(1, ptrPlayer))

ptrPlayer->Move();

break;

case Keyboard::Down:

ptrPlayer->setNumDirection(2);

if(!isThereAWall(2, ptrPlayer) && !isThereAMovObject(2, ptrPlayer))

ptrPlayer->Move();

break;

case Keyboard::Left:

ptrPlayer->setNumDirection(3);

if(!isThereAWall(3, ptrPlayer) && !isThereAMovObject(3, ptrPlayer))

ptrPlayer->Move();

break;

case Keyboard::Right:

ptrPlayer->setNumDirection(4);

if(!isThereAWall(4, ptrPlayer) && !isThereAMovObject(4, ptrPlayer))

ptrPlayer->Move();

break;

default:

break;

}

}

**inputrecordscreen.cpp:**

#include "inputrecordscreen.h"

using namespace sf;

InputRecordScreen::InputRecordScreen(RenderWindow\* window, int scr): win(window), score(scr)

{

listOR = new ListOfRecords();

repetition = false;

activeWin = true;

choice = 1;

// Шрифт

setCentralPos(win);

font.loadFromFile("Font/Pixeled.ttf");

// Title

text2 = Text("YOUR SCORE:",font);

text2.setCharacterSize(40);

text2.setPosition(centralPosition.x - text2.getGlobalBounds().width/2, centralPosition.y/4 - text2.getGlobalBounds().height/2);

textScore = Text(std::to\_string(score), font);

textScore.setFillColor(Color::Green);

textScore.setCharacterSize(80);

textScore.setPosition(centralPosition.x - textScore.getGlobalBounds().width/2,

centralPosition.y/4 - textScore.getGlobalBounds().height/2 + 100);

text = Text("ENTER NICKNAME", font);

text.setCharacterSize(40);

text.setPosition(centralPosition.x - text.getGlobalBounds().width/2, centralPosition.y - text.getGlobalBounds().height/2 - 80);

// Текст бар

RecTextBox = RectangleShape(Vector2f(550, 70));

RecTextBox.setOutlineThickness(3);

RecTextBox.setOutlineColor(Color::Red);

RecTextBox.setPosition(centralPosition.x - RecTextBox.getGlobalBounds().width/2+2, centralPosition.y - RecTextBox.getGlobalBounds().height/2+2);

inputLine = Text("", font);

inputLine.setCharacterSize(20);

inputLine.setFillColor(Color::Black);

inputLine.setPosition(RecTextBox.getPosition().x + RecTextBox.getGlobalBounds().width/2 - inputLine.getGlobalBounds().width/2,

RecTextBox.getPosition().y + RecTextBox.getGlobalBounds().height/2 - inputLine.getGlobalBounds().height/2);

// кнопки управления

textQuestion = Text("THIS NAME ALREADY EXISTS, SAVE IT?", font);

textQuestion.setCharacterSize(40);

textQuestion.setPosition(centralPosition.x - textQuestion.getGlobalBounds().width/2,

centralPosition.y - textQuestion.getGlobalBounds().height/2 + 200);

buttonOk = RectangleShape(sizeButton);

buttonOk.setOutlineThickness(3);

buttonOk.setOutlineColor(Color::Yellow);

buttonOk.setPosition(centralPosition.x - buttonOk.getGlobalBounds().width/2+252,

centralPosition.y - buttonOk.getGlobalBounds().height/2+402);

buttonCancel = RectangleShape(sizeButton);

buttonCancel.setOutlineThickness(3);

buttonCancel.setOutlineColor(Color::Yellow);

buttonCancel.setPosition(centralPosition.x - buttonCancel.getGlobalBounds().width/2-252,

centralPosition.y - buttonCancel.getGlobalBounds().height/2+402);

textBO = Text("OK", font);

textBO.setCharacterSize(25);

textBO.setFillColor(Color::Black);

textBO.setPosition(buttonOk.getPosition().x + buttonOk.getGlobalBounds().width/2 - textBO.getGlobalBounds().width/2,

buttonOk.getPosition().y + buttonOk.getGlobalBounds().height/2 - textBO.getGlobalBounds().height/2);

textBC = Text("Cancel", font);

textBC.setCharacterSize(25);

textBC.setFillColor(Color::Black);

textBC.setPosition(buttonCancel.getPosition().x + buttonCancel.getGlobalBounds().width/2 - textBC.getGlobalBounds().width/2,

buttonCancel.getPosition().y + buttonCancel.getGlobalBounds().height/2 - textBC.getGlobalBounds().height/2);

}

InputRecordScreen::~InputRecordScreen()

{

delete listOR;

}

void InputRecordScreen::HandlingKeyPress(){

if(!repetition){

if(ev.key.code == Keyboard::Enter){

repetition = listOR->cheakRepetition(inputLine.getString());

if(!repetition){

listOR->addNewRecord(inputLine.getString(),score);

activeWin = false;

}

}else if(97 <= ev.key.code+97 && ev.key.code+97 <= 122 && inputLine.getString().getSize() <= 15){

inputLine.setString(inputLine.getString() + (char)(ev.key.code+97));

inputLine.setPosition(RecTextBox.getPosition().x + RecTextBox.getGlobalBounds().width/2 - inputLine.getGlobalBounds().width/2,

RecTextBox.getPosition().y + RecTextBox.getGlobalBounds().height/2 - inputLine.getGlobalBounds().height/2);

}else if(ev.key.code == Keyboard::BackSpace){

inputLine.setString(inputLine.getString().substring(0,inputLine.getString().getSize()-1));

inputLine.setPosition(RecTextBox.getPosition().x + RecTextBox.getGlobalBounds().width/2 - inputLine.getGlobalBounds().width/2,

RecTextBox.getPosition().y + RecTextBox.getGlobalBounds().height/2 - inputLine.getGlobalBounds().height/2);

}

}else{

if(ev.key.code == Keyboard::Enter){

if(choice == 1)

repetition = false;

else{

listOR->changeScore(inputLine.getString(),score);

activeWin = false;

}

} else if(ev.key.code == Keyboard::Left){

if(choice == 1)

choice = 2;

else

choice -= 1;

} else if(ev.key.code == Keyboard::Right){

if(choice == 2)

choice = 1;

else

choice += 1;

}

}

}

void InputRecordScreen::show(){

while (activeWin) {

while (win->pollEvent(ev)) {

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

win->close();

else if(ev.type == Event::KeyReleased){

HandlingKeyPress();

if(choice == 1){

buttonOk.setOutlineColor(Color::Yellow);

buttonCancel.setOutlineColor(Color::Red);

} else{

buttonOk.setOutlineColor(Color::Red);

buttonCancel.setOutlineColor(Color::Yellow);

}

}

}

win->clear();

win->draw(text2);

win->draw(textScore);

win->draw(text);

win->draw(RecTextBox);

win->draw(inputLine);

if(repetition){

win->draw(textQuestion);

win->draw(buttonOk);

win->draw(buttonCancel);

win->draw(textBO);

win->draw(textBC);

}

win->display();

}

}

**listofrecords.cpp:**

#include "listofrecords.h"

using namespace std;

ListOfRecords::ListOfRecords()

{

string name;

int score;

ifstream in("Data/record.txt"); // окрываем файл для чтения

if (in.is\_open())

{

while (!in.eof())

{

in >> name;

in >> score;

listOfRecords.push\_back(new Record(name, score));

}

}

in.close(); // закрываем файл

}

ListOfRecords::~ListOfRecords(){

ofstream out; // поток для записи

out.open("Data/record.txt"); // окрываем файл для записи

if (out.is\_open())

{

iter = listOfRecords.begin();

while (iter != listOfRecords.end())

{

if((\*listOfRecords.rbegin()) == \*iter)

out << (\*\*iter).getNickname() << endl << (\*\*iter).getScore();

else

out << (\*\*iter).getNickname() << endl << (\*\*iter).getScore() << endl;

iter++;

}

}

out.close();

while (!listOfRecords.empty())

{

iter = listOfRecords.begin();

delete \*iter;

listOfRecords.erase(iter);

}

}

Record\* ListOfRecords::getListRecord(unsigned int num){

if(listOfRecords.size() >= num)

return listOfRecords[num-1];

else {

return new Record("0", 0);

}

}

bool ListOfRecords::comparator(Record\* firstElement, Record \*secondElement){

return firstElement->getScore() > secondElement->getScore();

}

void ListOfRecords::sorting(){

sort(listOfRecords.begin(), listOfRecords.end(), comparator);

}

void ListOfRecords::addNewRecord(string nick, int score){

listOfRecords.push\_back(new Record(nick, score));

sorting();

}

void ListOfRecords::changeScore(string nickname, int newScore){

iter = listOfRecords.begin();

while (iter != listOfRecords.end())

{

if((\*\*iter).getNickname() == nickname){

(\*\*iter).changeScore(newScore);

return;

}

iter++;

}

}

bool ListOfRecords::cheakRepetition(string cheak){

iter = listOfRecords.begin();

while (iter != listOfRecords.end())

{

if((\*\*iter).getNickname() == cheak)

return true;

iter++;

}

return false;

}

**map.cpp:**

#include "map.h"

using namespace std;

Map::Map(int numMap)

{

int i = 0;

char buff[24];

if(numMap == 1){

ifstream in("Data/maps/map1.txt"); // окрываем файл для чтения

if (in.is\_open())

{

while (!in.eof() && i < HEIGHT\_GAME\_BOX\_BY\_64)

{

in.getline(buff, 24);

mapLayout[i] = string(buff);

i++;

}

}

in.close(); // закрываем файл

}

}

void Map::setElementsToVector(vector<vector<StationaryObj\*>>\* listOfWalls, vector<Enemy\*>\* vecOfEnemy){

float X,Y = 3;

for(int i=0; i < HEIGHT\_GAME\_BOX\_BY\_64; i++){

X=3;

vector<StationaryObj\*> rowObjects;

for(int j = 0; j < WIDTH\_GAME\_BOX\_BY\_64; j++){

if(mapLayout[i][j] == '1')

rowObjects.push\_back(new BrickWall(X,Y));

else if(mapLayout[i][j] == '0')

rowObjects.push\_back(new MetalWall(X,Y));

else if(mapLayout[i][j] == '5')

startPlayerPosition = Vector2f(X,Y);

else if(mapLayout[i][j] == '4'){

vecOfEnemy->push\_back(new Enemy(Vector2f(X, Y)));

spawnPoint = Vector2f(X, Y);

}

else if(mapLayout[i][j] == '#'){

basePtr = new Base(X,Y);

rowObjects.push\_back(basePtr);

}

X += 64;

}

listOfWalls->push\_back(rowObjects);

Y += 64;

}

}

Vector2f Map::getSpawnPoint(){

return spawnPoint;

}

Vector2f Map::getStartPlayerPosition(){

return startPlayerPosition;

}

Base\* Map::getBasePtr(){

return basePtr;

}

**mapselectionscreen.cpp:**

#include "mapselectionscreen.h"

#include <iostream>

MapSelectionScreen::MapSelectionScreen(RenderWindow\* window): win(window)

{

activeWin = true;

choice = 1;

score = 0;

// Шрифт

setCentralPos(win);

font.loadFromFile("Font/Pixeled.ttf");

title = Text("MAP SELECTION",font);

title.setCharacterSize(40);

title.setPosition(centralPosition.x - title.getGlobalBounds().width/2, centralPosition.y/4 - title.getGlobalBounds().height/2);

// Карта 1

mapTexture1.loadFromFile("img/map1.png");

Map1 = RectangleShape(Vector2f(512, 512));

Map1.setOutlineThickness(3);

Map1.setOutlineColor(Color::Red);

Map1.setTexture(&mapTexture1);

Map1.setPosition(82, centralPosition.y - Map1.getGlobalBounds().height/2+2);

textMap1 = Text("Map 1", font);

textMap1.setCharacterSize(40);

textMap1.setFillColor(Color::Black);

textMap1.setPosition(Map1.getPosition().x + Map1.getGlobalBounds().width/2 - textMap1.getGlobalBounds().width/2,

Map1.getPosition().y + Map1.getGlobalBounds().height/2 - textMap1.getGlobalBounds().height/2);

// Карта 2(некст аптдейт)

Map2 = RectangleShape(Vector2f(512, 512));

Map2.setOutlineThickness(3);

Map2.setOutlineColor(Color::Yellow);

Map2.setPosition(centralPosition.x - Map2.getGlobalBounds().width/2+2, centralPosition.y - Map2.getGlobalBounds().height/2+2);

textMap2 = Text("Map 2(NEXT UPDATE)", font);

textMap2.setCharacterSize(30);

textMap2.setFillColor(Color::Black);

textMap2.setPosition(Map2.getPosition().x + Map2.getGlobalBounds().width/2 - textMap2.getGlobalBounds().width/2,

Map2.getPosition().y + Map2.getGlobalBounds().height/2 - textMap2.getGlobalBounds().height/2);

// Карта 3(некст апдейт)

Map3 = RectangleShape(Vector2f(512, 512));

Map3.setOutlineThickness(3);

Map3.setOutlineColor(Color::Yellow);

Map3.setPosition((2\*centralPosition.x) - 82 - Map3.getGlobalBounds().width, centralPosition.y - Map3.getGlobalBounds().height/2+2);

textMap3 = Text("Map 3(NEXT UPDATE)", font);

textMap3.setCharacterSize(30);

textMap3.setFillColor(Color::Black);

textMap3.setPosition(Map3.getPosition().x + Map3.getGlobalBounds().width/2 - textMap3.getGlobalBounds().width/2,

Map3.getPosition().y + Map3.getGlobalBounds().height/2 - textMap3.getGlobalBounds().height/2);

// Кнопка назад

buttonBack = RectangleShape(sizeButton);

buttonBack.setOutlineThickness(3);

buttonBack.setOutlineColor(Color::Yellow);

buttonBack.setPosition(centralPosition.x - buttonBack.getGlobalBounds().width/2+3,

centralPosition.y - buttonBack.getGlobalBounds().height/2+402);

textBB = Text("BACK", font);

textBB.setCharacterSize(25);

textBB.setFillColor(Color::Black);

textBB.setPosition(buttonBack.getPosition().x + buttonBack.getGlobalBounds().width/2 - textBB.getGlobalBounds().width/2,

buttonBack.getPosition().y + buttonBack.getGlobalBounds().height/2 - textBB.getGlobalBounds().height/2);

}

void MapSelectionScreen::HandlingKeyPress(){

if(ev.key.code == Keyboard::Enter){

switch (choice) {

case 1:

gameScreen = new GameScreen(win, choice, &score);

gameScreen->show();

delete gameScreen;

inputRS = new InputRecordScreen(win, score);

inputRS->show();

delete inputRS;

break;

case 2:

//map2

break;

case 3:

//map3

break;

case 0:

activeWin = false;

break;

}

}else if(ev.key.code == Keyboard::Left){

if(choice == 1)

choice = 3;

else

choice -= 1;

} else if(ev.key.code == Keyboard::Right){

if(choice == 3)

choice = 1;

else

choice += 1;

} else if(ev.key.code == Keyboard::Up){

if(choice >= 1)

choice = 0;

else

choice = 1;

} else if(ev.key.code == Keyboard::Down){

if(choice == 0)

choice = 1;

else

choice = 0;

}

}

void MapSelectionScreen::show(){

while (activeWin) {

while (win->pollEvent(ev)) {

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

win->close();

else if(ev.type == Event::KeyReleased){

HandlingKeyPress();

switch (choice) {

case 1:

Map1.setOutlineColor(Color::Red);

Map2.setOutlineColor(Color::Yellow);

Map3.setOutlineColor(Color::Yellow);

buttonBack.setOutlineColor(Color::Yellow);

break;

case 2:

Map1.setOutlineColor(Color::Yellow);

Map2.setOutlineColor(Color::Red);

Map3.setOutlineColor(Color::Yellow);

buttonBack.setOutlineColor(Color::Yellow);

break;

case 3:

Map1.setOutlineColor(Color::Yellow);

Map2.setOutlineColor(Color::Yellow);

Map3.setOutlineColor(Color::Red);

buttonBack.setOutlineColor(Color::Yellow);

break;

case 0:

Map1.setOutlineColor(Color::Yellow);

Map2.setOutlineColor(Color::Yellow);

Map3.setOutlineColor(Color::Yellow);

buttonBack.setOutlineColor(Color::Red);

break;

}

}

}

win->clear();

win->draw(title);

win->draw(Map1);

win->draw(Map2);

win->draw(Map3);

win->draw(textMap1);

win->draw(textMap2);

win->draw(textMap3);

win->draw(buttonBack);

win->draw(textBB);

win->display();

}

}

**metalwall.cpp:**

#include "metalwall.h"

MetalWall::MetalWall(float X, float Y)

{

this->x = X;

this->y = Y;

destroyable = false;

size = Vector2f(64,64);

health = 3;

objImage.loadFromFile("img/metalWall.png");

objTexture.loadFromImage(objImage);

obj = RectangleShape(size);

obj.setTexture(&objTexture);

obj.setPosition(this->x, this->y);

}

int MetalWall::getHealth(){

return health;

}

bool MetalWall::getTypeDestroy(){

return destroyable;

}

float MetalWall::getX(){

return obj.getPosition().x;

}

float MetalWall::getY(){

return obj.getPosition().y;

}

Vector2f MetalWall::getSize(){

return size;

}

RectangleShape MetalWall::getRectangleShape(){

return obj;

}

**movingobj.cpp:**

#include "movingobj.h"

MovingObj::MovingObj()

{

}

**outputrecordscreen.cpp:**

#include "outputrecordsscreen.h"

using namespace sf;

OutputRecordsScreen::OutputRecordsScreen(RenderWindow\* window): win(window){

listOR = new ListOfRecords();

Text\* textBuffNick;

Text\* textBuffScore;

Text\* textBuffNumber;

// Шрифт

setCentralPos(win);

font.loadFromFile("Font/Pixeled.ttf");

// Текст кнопок

textBB = Text("Back", font);

textBB.setCharacterSize(40);

textBB.setFillColor(Color::Black);

buttonBack.setSize(sizeButton);

buttonBack.setOutlineThickness(3);

buttonBack.setOutlineColor(Color::Red);

buttonBack.setPosition(centralPosition.x - buttonBack.getGlobalBounds().width/2 + 3,

centralPosition.y - buttonBack.getGlobalBounds().height/2 + 450);

textBB.setPosition(buttonBack.getPosition().x - textBB.getGlobalBounds().width/2 + buttonBack.getGlobalBounds().width/2 + 3,

buttonBack.getPosition().y - textBB.getGlobalBounds().height/2 + buttonBack.getGlobalBounds().height/2 + 3);

table.setSize(Vector2f(800,700));

table.setOutlineThickness(3);

table.setOutlineColor(Color::Blue);

table.setPosition(centralPosition.x - table.getGlobalBounds().width/2+2, centralPosition.y - table.getGlobalBounds().height/2+2);

title = Text("RECORDS TABLE", font);

title.setCharacterSize(40);

title.setPosition(centralPosition.x - title.getGlobalBounds().width/2,

table.getPosition().y - title.getGlobalBounds().height - 80);

textNumber = Text("Num", font);

textNumber.setCharacterSize(25);

textNumber.setFillColor(Color::Black);

textNumber.setPosition(table.getPosition().x + 48,

table.getPosition().y - textNumber.getGlobalBounds().height/2 + table.getGlobalBounds().height/11 + 2);

textNickname = Text("Nickname", font);

textNickname.setCharacterSize(25);

textNickname.setFillColor(Color::Black);

textNickname.setPosition(table.getPosition().x - textNickname.getGlobalBounds().width/2+table.getGlobalBounds().width/2,

table.getPosition().y - textNickname.getGlobalBounds().height/2 + table.getGlobalBounds().height/11 + 2);

textScore = Text("Score", font);

textScore.setCharacterSize(25);

textScore.setFillColor(Color::Black);

textScore.setPosition(table.getPosition().x - textScore.getGlobalBounds().width+table.getGlobalBounds().width - 48,

table.getPosition().y - textScore.getGlobalBounds().height/2 + table.getGlobalBounds().height/11 + 2);

for(unsigned int i = 1; i < 11; i++){

textBuffNumber = new Text(to\_string(i),font);

textBuffNumber->setCharacterSize(20);

textBuffNumber->setFillColor(Color::Black);

textBuffNumber->setPosition(table.getPosition().x + 48 + textNumber.getGlobalBounds().width/2,

table.getPosition().y - textBuffNumber->getGlobalBounds().height/2 + (i+1)\*table.getGlobalBounds().height/12);

listRecordsNum.push\_back(textBuffNumber);

textBuffNick = new Text(listOR->getListRecord(i)->getNickname(),font);

textBuffNick->setCharacterSize(20);

textBuffNick->setFillColor(Color::Black);

textBuffNick->setPosition(textNickname.getPosition().x + textNickname.getGlobalBounds().width/2 - textBuffNick->getGlobalBounds().width/2,

table.getPosition().y - textBuffNick->getGlobalBounds().height/2 + (i+1)\*table.getGlobalBounds().height/12);

listRecordsNick.push\_back(textBuffNick);

textBuffScore = new Text(to\_string(listOR->getListRecord(i)->getScore()),font);

textBuffScore->setCharacterSize(20);

textBuffScore->setFillColor(Color::Black);

textBuffScore->setPosition(textScore.getPosition().x + textScore.getGlobalBounds().width/2 - textBuffScore->getGlobalBounds().width/2,

table.getPosition().y - textBuffScore->getGlobalBounds().height/2 + (i+1)\*table.getGlobalBounds().height/12);

listRecordsScore.push\_back(textBuffScore);

}

delete listOR;

}

OutputRecordsScreen::~OutputRecordsScreen(){

while (!listRecordsNum.empty())

{

iter = listRecordsNum.begin();

delete \*iter;

listRecordsNum.erase(iter);

}

while (!listRecordsNick.empty())

{

iter = listRecordsNick.begin();

delete \*iter;

listRecordsNick.erase(iter);

}

while (!listRecordsScore.empty())

{

iter = listRecordsScore.begin();

delete \*iter;

listRecordsScore.erase(iter);

}

}

void OutputRecordsScreen::displayList(vector<Text \*> vec){

iter = vec.begin();

while (iter != vec.end())

{

win->draw(\*\*iter);

iter++;

}

}

void OutputRecordsScreen::show(){

while (win->isOpen()) {

while (win->pollEvent(ev)) {

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

win->close();

else if(ev.type == Event::KeyReleased){

if(ev.key.code == Keyboard::Enter)

return;

}

}

win->clear();

win->draw(title);

win->draw(table);

win->draw(textNumber);

win->draw(textNickname);

win->draw(textScore);

win->draw(buttonBack);

displayList(listRecordsNum);

displayList(listRecordsNick);

displayList(listRecordsScore);

win->draw(textBB);

win->display();

}

}

**player.cpp:**

#include "player.h"

Player::Player(Vector2f startPosition)

{

this->x = startPosition.x;

this->y = startPosition.y;

size = Vector2f(64,64);

health = 3;

objImage.loadFromFile("img/tankPlayerUp.png");

objImage.createMaskFromColor(Color(0,0,1));

objTexture.loadFromImage(objImage);

obj = RectangleShape(size);

obj.setTexture(&objTexture);

obj.setPosition(this->x, this->y);

cooldown.restart();

}

float Player::getX(){

return obj.getPosition().x;

}

float Player::getY(){

return obj.getPosition().y;

}

void Player::changeHealth(){

health--;

}

Vector2f Player::getSize(){

return size;

}

RectangleShape Player::getRectangleShape(){

return obj;

}

int Player::getNumDirection(){

return numDirection;

}

void Player::setNumDirection(int numDir){

numDirection = numDir;

if(numDirection == 1){

objImage.loadFromFile("img/tankPlayerUp.png");

objImage.createMaskFromColor(Color(0,0,1));

objTexture.loadFromImage(objImage);

} else if(numDirection == 2){

objImage.loadFromFile("img/tankPlayerDown.png");

objImage.createMaskFromColor(Color(0,0,1));

objTexture.loadFromImage(objImage);

} else if(numDirection == 3){

objImage.loadFromFile("img/tankPlayerLeft.png");

objImage.createMaskFromColor(Color(0,0,1));

objTexture.loadFromImage(objImage);

} else if(numDirection == 4){

objImage.loadFromFile("img/tankPlayerRight.png");

objImage.createMaskFromColor(Color(0,0,1));

objTexture.loadFromImage(objImage);

}

}

void Player::Move(){

if(numDirection == 1)

obj.move(0,-2);//вперед

else if(numDirection == 2)

obj.move(0,2);//вниз

else if(numDirection == 3)

obj.move(-2,0);//влево

else if(numDirection == 4)

obj.move(2,0);//вправо

}

int Player::getHealth(){

return health;

}

float Player::getCooldownTime(){

return cooldown.getElapsedTime().asSeconds();

}

Bullet\* Player::Fire(){

cooldown.restart();

if(numDirection == 1)

return new Bullet(Vector2f(obj.getPosition().x + (size.x / 2), obj.getPosition().y), numDirection);

else if(numDirection == 2)

return new Bullet(Vector2f(obj.getPosition().x + (size.x / 2), obj.getPosition().y + size.y), numDirection);

else if(numDirection == 3)

return new Bullet(Vector2f(obj.getPosition().x, obj.getPosition().y + (size.y / 2)), numDirection);

else if(numDirection == 4)

return new Bullet(Vector2f(obj.getPosition().x + size.x, obj.getPosition().y + (size.y / 2)), numDirection);

return NULL;

}

**record.cpp:**

#include "record.h"

string Record::getNickname(){

*return* nickname;

}

int Record::getScore(){

*return* score;

}

void Record::changeScore(int newScore){

score = newScore;

}

**screens.cpp:**

#include "screens.h"

Screens::Screens()

{

*//* *Шрифт*

font.loadFromFile("Font/Pixeled.ttf");

*//* *Размер* *кнопок*

sizeButton = Vector2f(430, 70);

}

void Screens::setCentralPos(RenderWindow\* win){

*//* *Центровка* *текста*

centralPosition = Vector2f(win->*getSize*().x/2 , win->*getSize*().y /2);

}

**userinterfacescreen.cpp:**

#include "userinterfacescreen.h"

#include <fstream>

using namespace sf;

using namespace std;

UserInterfaceScreen::UserInterfaceScreen(){

// Окно

win = new RenderWindow(VideoMode(400,400), "SFML", Style::Fullscreen); //указатель на окно

win->setFramerateLimit(50); //FPS = 60

win->setMouseCursorVisible(false); // Убран курсор

setCentralPos(win);

// Текст

label = Text("Crime", font);

label.setCharacterSize(100);

label2 = Text("City", font);

label2.setCharacterSize(100);

label.setPosition(centralPosition.x - label.getGlobalBounds().width /2,

centralPosition.y - label.getGlobalBounds().height/2 - 200);

label2.setPosition(centralPosition.x - label2.getGlobalBounds().width /2,

centralPosition.y - label2.getGlobalBounds().height/2 - 50);

creatorsText = Text("Creators:\nNikita Sorokin\nFedor Safronov\nAnna Kemova\nSemen Zabirohin\nAnton Gubar",font);

creatorsText.setCharacterSize(20);

creatorsText.setPosition(centralPosition.x \* 2 - label2.getGlobalBounds().width + 50, 10);

// Текст кнопок

textBS = Text("Start game", font);

textBS.setCharacterSize(40);

textBS.setFillColor(Color::Black);

textBR = Text("Records", font);

textBR.setCharacterSize(40);

textBR.setFillColor(Color::Black);

textBE = Text("Exit", font);

textBE.setFillColor(Color::Black);

textBE.setCharacterSize(40);

// Кнопка Старт

buttonStart.setSize(sizeButton);

buttonStart.setOutlineThickness(3);

buttonStart.setPosition(centralPosition.x - buttonStart.getGlobalBounds().width/2 + 3,

centralPosition.y - buttonStart.getGlobalBounds().height/2 + 150);

textBS.setPosition(buttonStart.getPosition().x - textBS.getGlobalBounds().width/2 + buttonStart.getGlobalBounds().width/2 + 3,

buttonStart.getPosition().y - textBS.getGlobalBounds().height/2 + buttonStart.getGlobalBounds().height/2 + 3);

// Кнопка Рекордов

buttonRecords.setSize(sizeButton);

buttonRecords.setOutlineThickness(3);

buttonRecords.setPosition(centralPosition.x - buttonRecords.getGlobalBounds().width/2 + 3,

centralPosition.y - buttonRecords.getGlobalBounds().height/2 + 250);

textBR.setPosition(buttonRecords.getPosition().x - textBR.getGlobalBounds().width/2 + buttonRecords.getGlobalBounds().width/2 + 3,

buttonRecords.getPosition().y - textBR.getGlobalBounds().height/2 + buttonRecords.getGlobalBounds().height/2 + 3);

// Кнопка Выхода

buttonExit.setSize(sizeButton);

buttonExit.setOutlineThickness(3);

buttonExit.setPosition(centralPosition.x - buttonExit.getGlobalBounds().width/2 + 3,

centralPosition.y - buttonExit.getGlobalBounds().height/2 + 350);

textBE.setPosition(buttonExit.getPosition().x - textBE.getGlobalBounds().width/2 + buttonExit.getGlobalBounds().width/2 + 3,

buttonExit.getPosition().y - textBE.getGlobalBounds().height/2 + buttonExit.getGlobalBounds().height/2 + 3);

// Изначально выбран старт игры

changeColor(1);

}

void UserInterfaceScreen::changeColor(int ch){

switch (ch) {

case 1:

buttonStart.setOutlineColor(Color::Red);

buttonRecords.setOutlineColor(Color::Yellow);

buttonExit.setOutlineColor(Color::Yellow);

break;

case 2:

buttonStart.setOutlineColor(Color::Yellow);

buttonRecords.setOutlineColor(Color::Red);

buttonExit.setOutlineColor(Color::Yellow);

break;

case 3:

buttonStart.setOutlineColor(Color::Yellow);

buttonRecords.setOutlineColor(Color::Yellow);

buttonExit.setOutlineColor(Color::Red);

break;

}

}

void UserInterfaceScreen::interact(){

while (win->isOpen()) {

while (win->pollEvent(ev)) {

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

win->close();

else if(ev.type == Event::KeyReleased){

if(ev.key.code == Keyboard::Down){

if(choice == 3)

choice = 1;

else

choice += 1;

}

else if(ev.key.code == Keyboard::Up){

if(choice == 1)

choice = 3;

else

choice -= 1;

}

else if(ev.key.code == Keyboard::Enter){

switch (choice) {

case 1:

ptrMapSelectionScreen = new MapSelectionScreen(win);

ptrMapSelectionScreen->show();

delete ptrMapSelectionScreen;

break;

case 2:

ptrOutputRS = new OutputRecordsScreen(win);

ptrOutputRS->show();

delete ptrOutputRS;

break;

case 3:

win->close();

break;

}

}

changeColor(choice);

}

}

win->clear();

win->draw(label);

win->draw(label2);

win->draw(creatorsText);

win->draw(buttonStart);

win->draw(buttonRecords);

win->draw(buttonExit);

win->draw(textBS);

win->draw(textBR);

win->draw(textBE);

win->display();

}

}

**main.cpp:**

#include <userinterfacescreen.h>

int main()

{

UserInterfaceScreen\* uis = new UserInterfaceScreen();

uis->interact();

delete uis;

return 0;

}

#### 3.14 Руководство пользователя.

Для запуска игры компилятор и отладчик должны соответствовать версии SMFL-2.5.1. Также необходимо отключить теневую сборку. Если появилось только окно консоли, то проверьте не заблокировал ли антивирусник exe файл.

При запуске игры пользователю открывается главное меню:

Навигация по форме осуществляется с помощью стрелок на клавиатуре. Для подтверждения выбора нажимается клавиша “Enter”.



Рис.17. Главное меню.

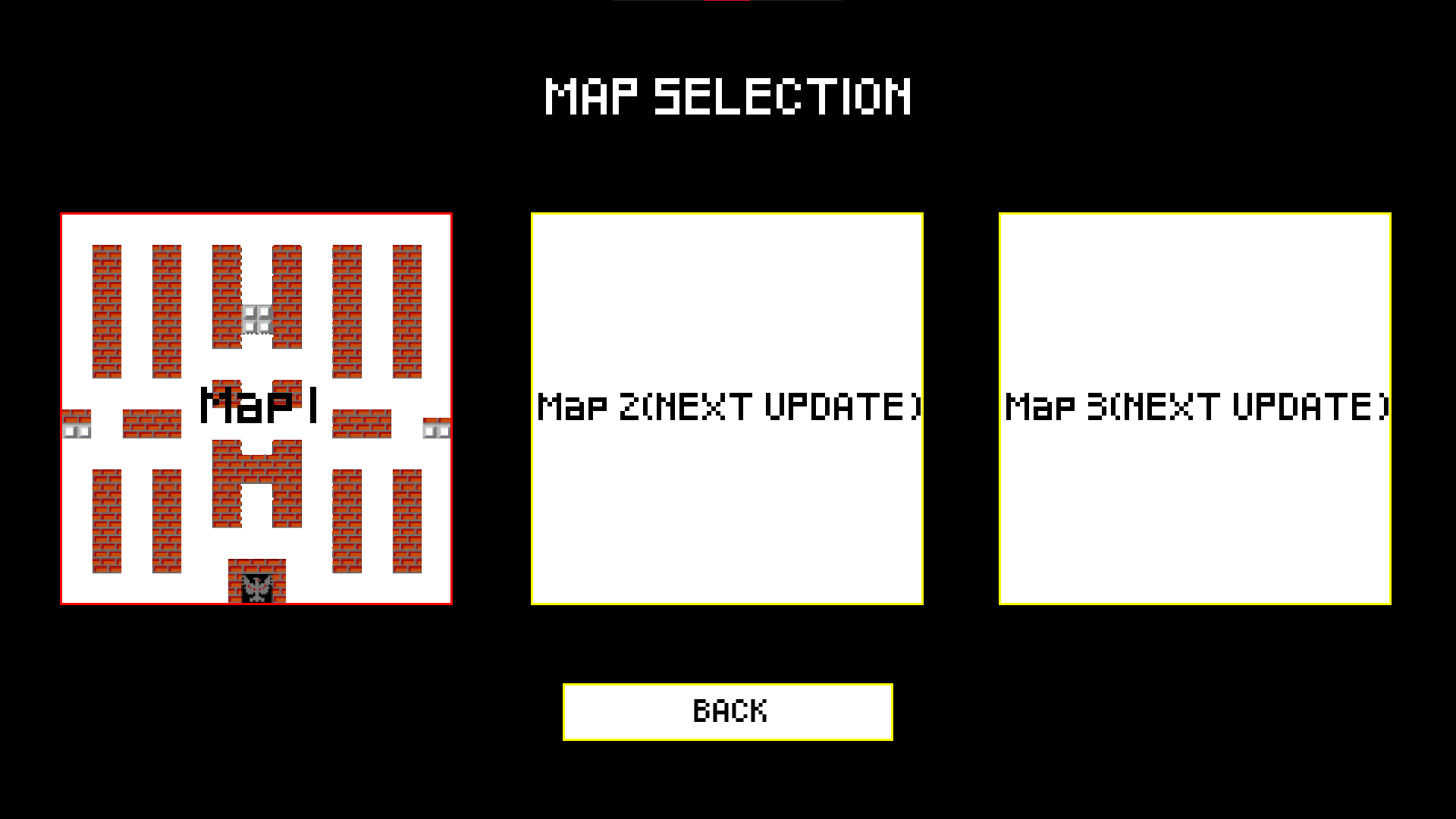
1. При выборе пункта “Start game”(нажать клавишу “Enter”) появится окно выбора карты.  
   

Рис.18. Окно выбора карты.

1. При выборе пункта “Records”(нажать на стрелку вниз и клавишу “Enter”) появится окно с таблице рекордов.



Рис.19. Окно таблицы рекордов.

1. При нажатии пункта “Exit”(нажать на стрелку вниз два раза и клавишу “Enter”) происходить выход из программы.

Рассмотрим более подробно пункт “Start game”:

Выбор карт осуществляется с помощью стрелок влево/вправо. Подтверждение выбора - клавиша “Enter”. Также можно вернутся обратно в главное меню с помощью стрелки вниз и клавиши “Enter”.

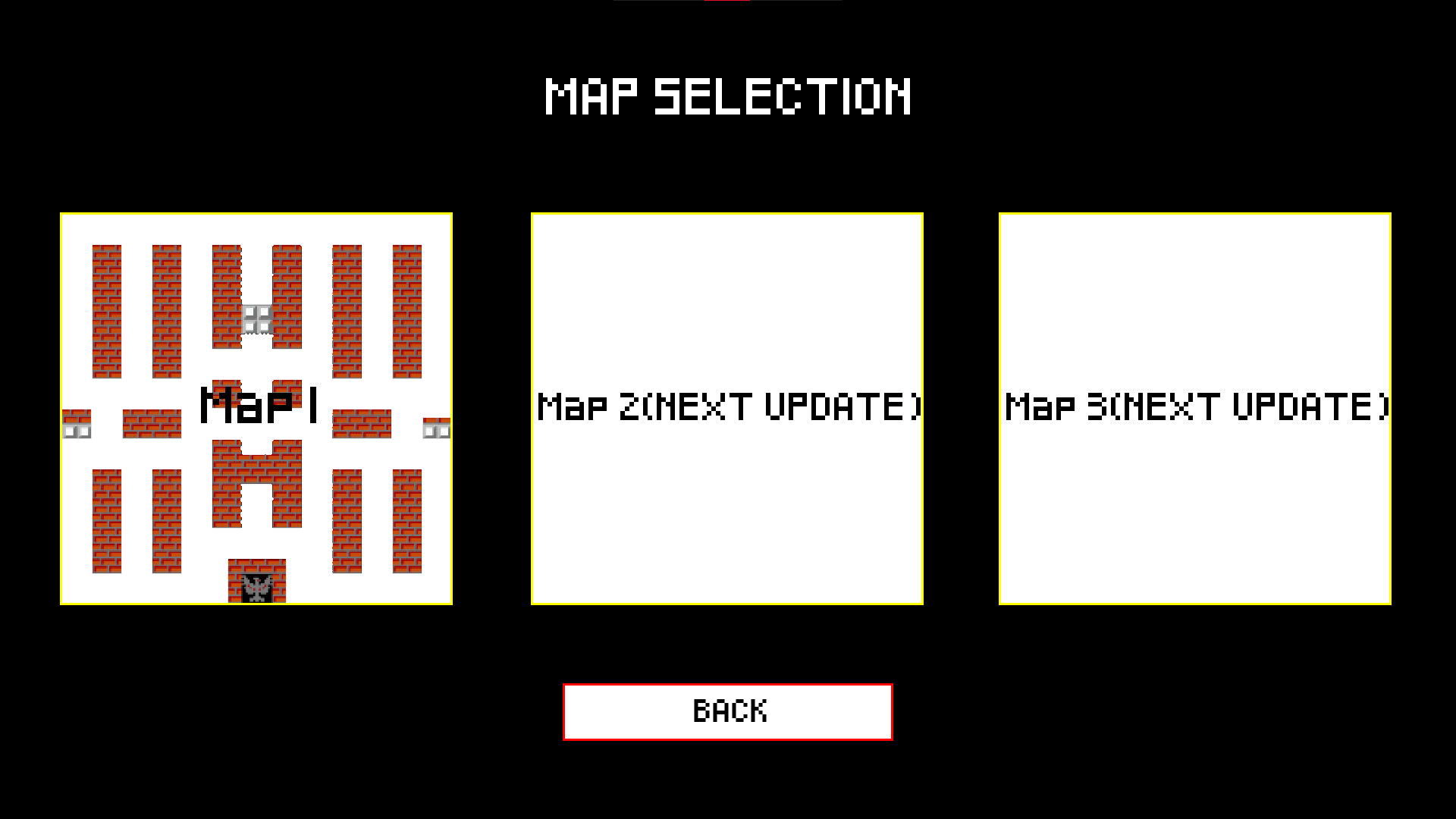


Рис.20. Выделена кнопка “Back”.

При нажатии “Enter” с выделенной картой открывается игровая сцена.

Рассмотрим более подробно пункт “Records”:

Перед вами отобразится список текущих рекордсменов. Возврат в главное меню осуществляется с помощью нажатия на кнопку “Back”(она всегда активна на данной форме). Для ее активации достаточно нажать клавишу “Enter” на клавиатуре.

Рассмотрим более подробно пункт с игровой сценой:

Перед вами отобразится игровая сцена. Ваш танк желтого цвета. Для победа Вы должны уничтожить все вражеские танки(серого цвета). Количество танков противника пишется под надписью “Remaining enemies”. Также Вы должны защищать свою “базу”, она изображена в виде орла. На форме помимо количества врагов отображены: счет(под надписью “Score”) и количество оставшихся жизней(в виде сердец).

Передвижение танка осуществляется с помощью стрелок на клавиатуре. Стрельба - клавишей “Enter”. На карте есть также и препятствия. Некоторые из них разрушаются пулей(кирпичные стены), а некоторые нет(металлические стены).



Рис.21. Игровая сцена.



Рис.22. Выстрел.



Рис.23. Движение влево.



Рис.24. Движение вправо.



Рис.25. Движение вверх.



Рис.26. Движение вниз.

При победе/поражении отобразится окно ввода никнейма.

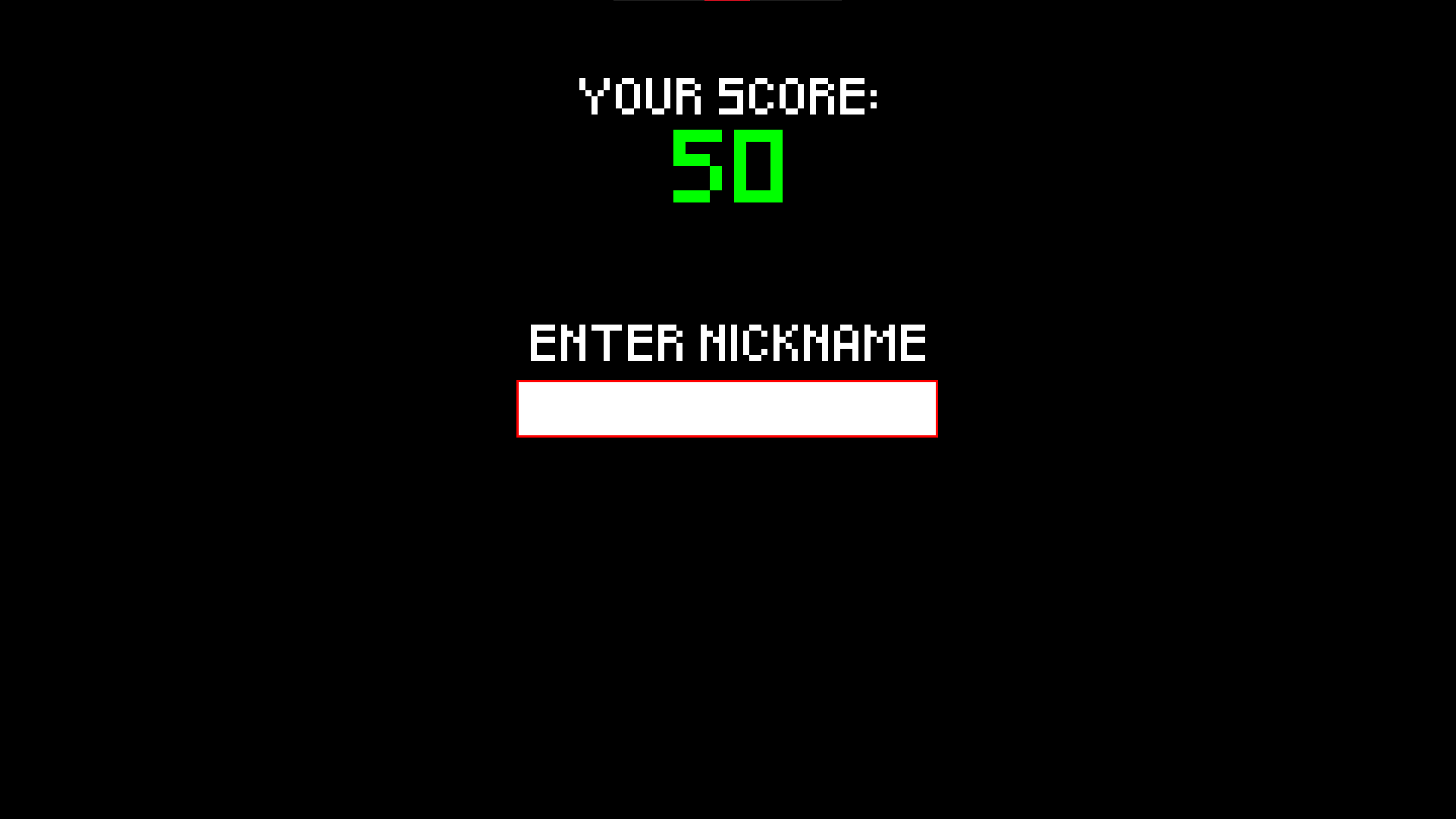


Рис.27. Окно ввода никнейма(окно записи рекорда).

Рассмотрим более подробно окно ввода никнейма(рекорда):

В данном окне с помощью клавиш на клавиатуре Вы вводите свой никнейм/прозвище или настоящее имя. Для подтверждения нажмите клавишу “Enter”. Клавиша “BackSpace” удаляет последнюю букву. Если ваш никнейм уже есть в списке рекордов, то Вам предложат выбор: перезаписать счет или не записывать его.

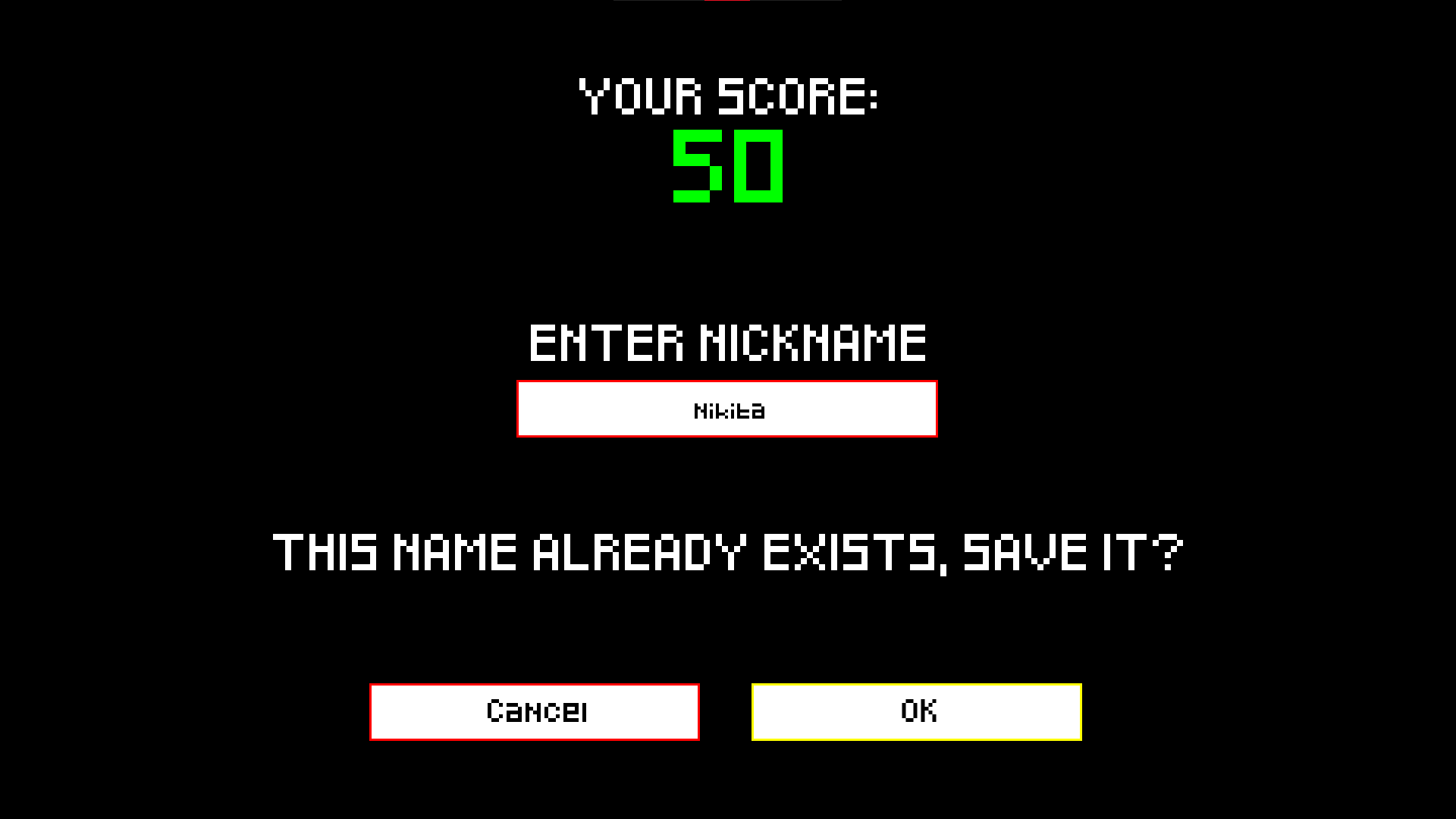


Рис.28. Предложение переписать в окне ввода рекорда.

Если Вы хотите перезаписать счет, то нажмите кнопку “Ok”(клавишу вправо и “Enter”, иначе кнопку “Cancel”(“Enter”). При выборе “перезаписать” Вы попадете в меню выбора карты, а при отмене операции Вам необходимо изменить никнейм в поле ввода.

### 4 История проекта на GitHub

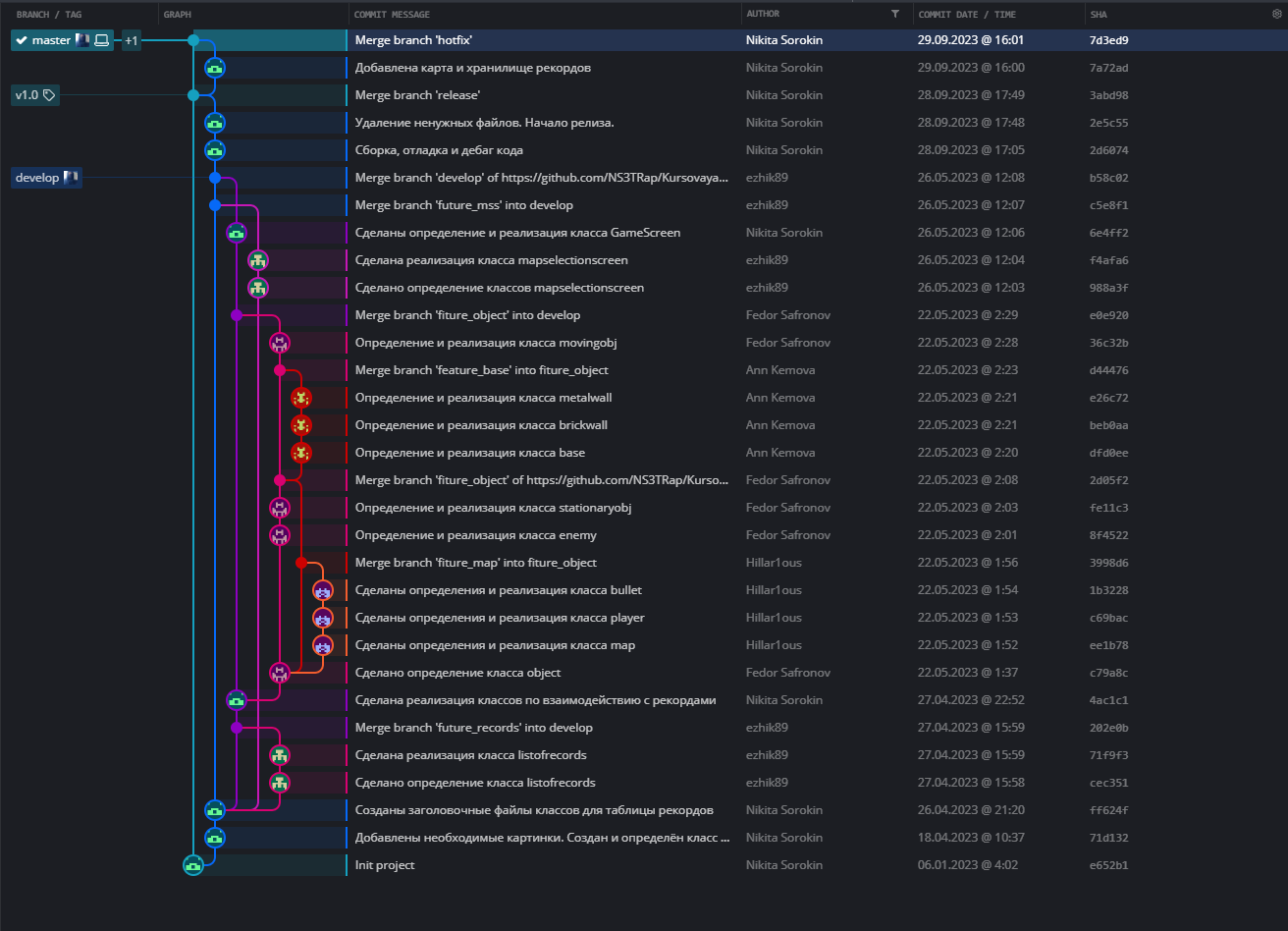


Рис.29. История проекта на GitHub.

На Рис.29 представлена история проекта с текстовым пояснением назначения каждого коммита, также видно имя пользователя, который вносил изменения.

Из-за невнимательности пришлось сделать hotfix, так как забыли добавить в отслеживаемые папку с картой и таблицей рекордов. Данную невнимательность расцениваем как возможность использовать ветку “hotfix” и показать все возможные варианты ветвления в истории проекта.

Ссылка на GitHub: <https://github.com/NS3TRap/Kursovaya> .

### 5 Заключение

Для выполнения проектного задания мы использовали среду разработки под названием Qt Creator, используя язык программирования C++, с применением системы контроля версий, реализованы все прецеденты, описанные на диаграмме вариантов использования, по предварительным тестам сбоев не было обнаружено, зависания как таковые отсутствуют, также использован метод раздельной компиляции, реализована очистка динамической памяти, отсутствуют неиспользуемые переменные, в программе избыточность не наблюдается, наша команда считает цель полностью достигнутой и выполненной, но все же есть направления в которых можно было бы доработать проект.