МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«КУРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет физики, математики, информатики
Кафедра программного обеспечения и администрирования информационных систем

КУРСОВОЙ ПРОЕКТ

по дисциплине
Программирование на языке высокого уровня
на тему: ПРОГРАММНАЯ РЕАЛИЗАЦИЯ ИГРЫ «ПАКМАН»

Обучающегося 2 курса
очной формы обучения
направления подготовки
02.03.03 Математическое обеспечение
и администрирование
информационных систем
Направленность (профиль)
Проектирование информационных
систем и баз данных
Козявина Максима Сергеевича

Руководитель:
старший преподаватель кафедры
ПОиАИС Ураева Елена Евгеньевна

Допустить к защите:
_____ / _____
«_____» _____ 20_____г.

Курск, 2022

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Современный человек ежедневно во всех сферах жизнедеятельности взаимодействует с компьютерными технологиями. С развитием технологий увеличивается и уровень взаимодействия. Одним из самых распространенных проявлений взаимодействия являются «компьютерные игры» или «видеоигры». В настоящее время игровая индустрия активно развивается. Каждый год на рынок выпускаются десятки тысяч новых игр, которые расходятся миллионами копий по всему миру.

Игра «Pac-Man» является известной игрой в жанре аркады, вышедшая в 1980 году в Японии. Эта игра является ярким представителем игр класса лабиринт с полной информацией. Несмотря на кажущуюся простоту, «Pac-Man» является хорошей средой для реализации и тестирования как объектно-ориентированного программирования, так и искусственного интеллекта, ввиду сложной структуры лабиринта, наличия очков, а также оппонентов, которых представляют призраки.

Целью данного проекта является разработка программного продукта, представляющего собой аналог известной компьютерной игры. Программа должна обеспечивать оптимальный игровой процесс, а также иметь понятный пользовательский интерфейс и использовать минимальное количество ресурсов компьютера.

В соответствии с целью данного проекта, можно выделить следующие *задачи*:

- изучить принципы и правила игры «Pac-Man»;
- разработать способы представления задачи;
- создать приложение обеспечивающее:

1) ввод имени игрока, которое будет отображаться в таблице рейтинга;
2) возможность управления при помощи клавиатуры;
3) игровой процесс, соответствующий игре «Pac-Man».

Аннотация: в данной пояснительной записке содержится информация о разработке программной реализации игры «Пакман». Приложение включает в себя множество функций, которые есть в программах аналогичных этой, таких как: ввод имени игрока с дальнейшим отображением его в таблице рейтинга, управление при помощи клавиатуры, реализация игрового процесса с подсчетом очков по прохождении уровня, а также наличие уровней сложности с увеличением количества противников и скоростью их перемещения.

Ключевые слова: «Пакман», игровой процесс, таблица рейтинга, уровни сложности, приложение.

Annotation: this explanatory note contains information about the development of the software implementation of the Pacman game. The application includes many functions that are in the program, such as: entering the player's name with its indication in the rating table, control using the application, the implementation of the gameplay with scoring for passing the level, as well as the presence of difficulty levels with the identification of the number of opponents and their speed of movement.

Keywords: Pac-Man, gameplay, rating table, difficulty levels, application.

1 Постановка задачи

1.1 Формулировка условия задачи

Реализовать приложение, представляющее собой аналог известной игры. Обеспечить игровой процесс, учитывающий скорость прохождения игроком уровня и число съеденных точек. При старте программы пользователь должен иметь возможность ввода имени, которое будет отображаться в таблице рейтинга игроков. Игровой лабиринт должен быть не менее, чем 20×20 клеток, предусмотреть наличие сквозного туннеля. Обеспечить в приложении возможность управления Пакманом при помощи клавиатуры. При движении персонажа необходимо обеспечить разворот его «лицевой части» при смене направления. Расположить на карте несколько «экстра-монет» (энерджайзеров), за поедание которых дается больше очков, а у Пакмана временно увеличивается скорость. Обеспечить возможность реализации игрового процесса в режиме двух игроков. Также должно присутствовать несколько уровней сложности. Игра завершается в случае уничтожения Пакманом всех точек.

1.2 Анализ и исследование задачи

1.2.1 Краткие теоретические сведения об объекте исследования

В ходе выполнения работы был изучен функционал аналогичных программ, а также отзывы пользователей к ним. В результате чего было выявлено, какие элементы игрового процесса являются более востребованными, какие дополнительными, а также те, которые большинством пользователей не используются (п.1.2.3).

1.2.2 Обзор и анализ возможных альтернатив

В качестве примера работающего примера игры была исследована бесплатная версия игры доступная на сайте Goggle, представленная на рисунке 1, и open source клон игры под названием «Kapman», представленный на рисунке 2, после чего был выбран подходящий внешний вид приложения, который описан в разделе 2.4.

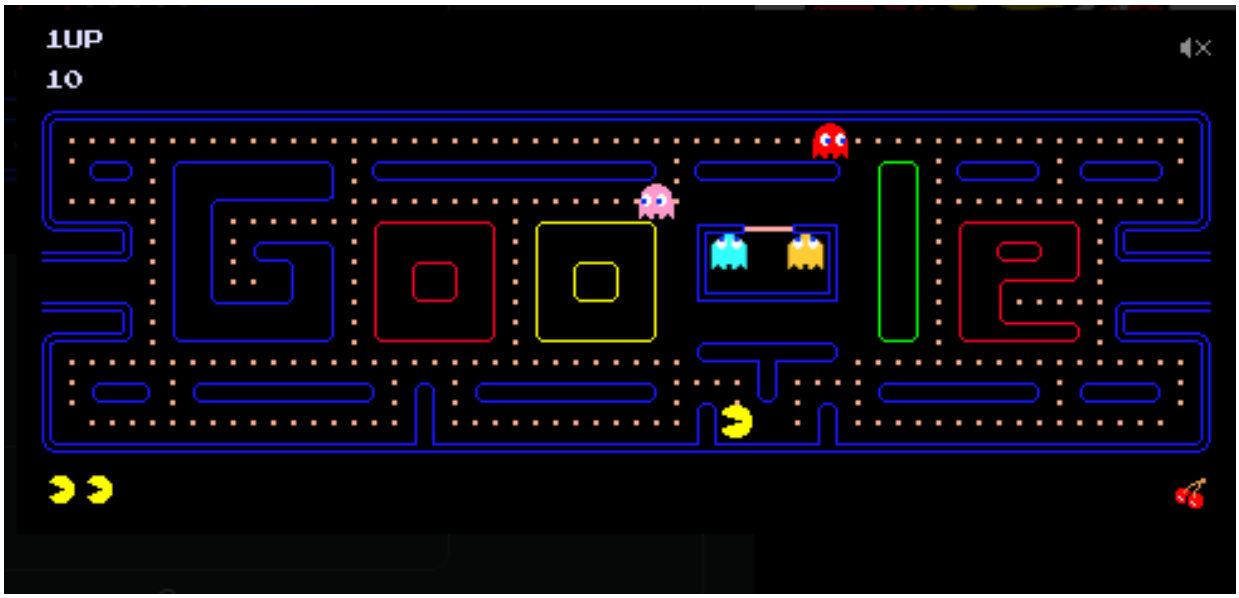

Рисунок 1 – Pacman от Google

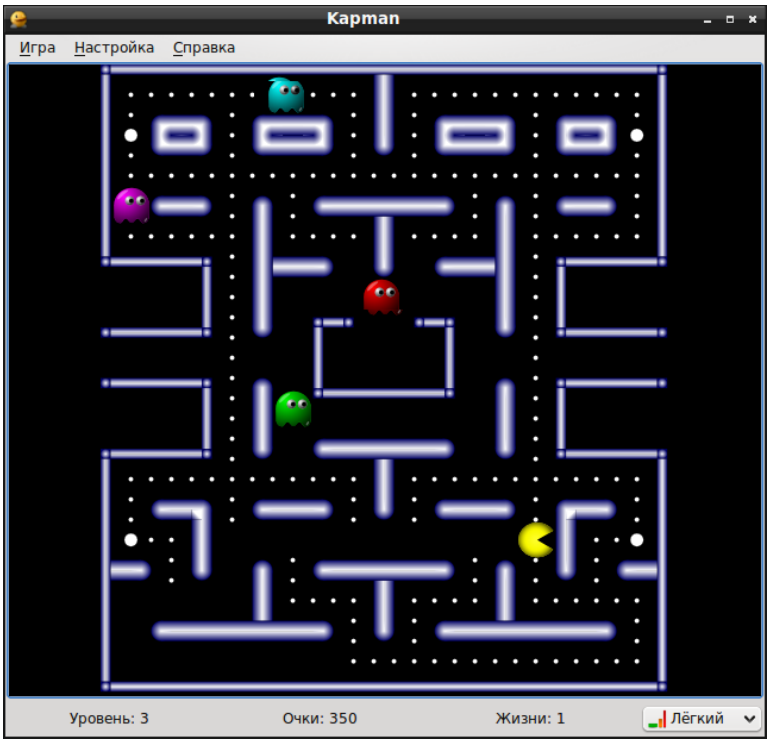

Рисунок 2 – Kapman

Рассмотренные альтернативы имеют следующие недостатки:

– Присутствует всего лишь один уровень;

– Отсутствует таблица рекордов;

– Отсутствует звуковое сопровождения;

– Не всегда враги передвигаются по оптимальному маршруту. [1]

При реализации приложения в рамках данного курсового проектирования будут учтены все особенности приложений подобного рода, устранены их уязвимости.

1.2.3 Разработка требований к приложению

Пользователь должен иметь возможность выполнять следующие действия:

– Использовать разнообразные файлы уровней из памяти компьютера;

– Настраивать управление;

– Играть, используя настроенные клавиши;

– Просматривать таблицу рекордов;

– Добавлять свою фоновую музыку.

1.2.4 Анализ инструментальных средств

Для проектирования и разработки приложения была выбрана среда разработки Qt Creator 7 с использованием Qt 5 C++. Данная IDE имеет достаточный и простой инструментарий для реализации оконного интерфейса приложения. Ещё одним фактором выбора данной среды является, знакомство автора проекта с доступным и понятным функционалом инструмента разработки. Qt является проверенной временем средой, по ней написано множество справочной литературы, способствующей простому пониманию и рациональному использованию возможностей данного продукта [2].

1.3 Методические ограничения

1.3.1 Стандарты

Разработка программной документации и программы должна производиться согласно ЕСПД, ГОСТ 19.701-90, ГОСТ 2.304-88.

1.3.2 Программная совместимость

Разработанная программа должна работать под управлением операционных систем Windows XP/Vista/7/8.1/10/11.

1.3.3 Требования к составу и параметрам технических средств

Для работы приложения желательно иметь персональный компьютер со следующими минимальными техническими характеристиками:

- микропроцессор Intel Pentium или совместимый с частотой не менее 1,6 ГГц (рекомендуется 2 ГГц и выше);
- базовый графический процессор любого поставщика, класса DirectX 9.0 или выше (Pixel Shader 2.0);
- аудио карта;
- 1 ГБ оперативной памяти (рекомендуется 2 ГБ и более);
- объём свободного места на жестком диске не менее 20 Мб;
- монитор с разрешением 1024×768 или выше;
- мышь или другое указывающее устройство;
- клавиатура.

1.3.4 Входные данные

Входными данными программы являются строки, символы и текстовые файлы (.TXT) кодировки UTF-8 и звуковые файлы формата Waveform (.WAV) и MPEG-1/2/2.5 Layer 3 (.MP3).

1.3.5 Выходные данные

Выходными данными программы являются текстовые файлы (.TXT).

1.3.6 Безопасность и секретность

Программа не требует защиты и может свободно распространяться.

1.3.7 Мобильность

Для копирования программы с внешнего накопителя на компьютер необходимо:

1. Скопировать папку с приложением и сопутствующими ей файлами, в какую-либо папку на жёстком диске компьютера.

2. Запустить файл pacman.exe.

2 Проектирование и разработка приложения

2.1 Разработка структуры данных

Программный объект представляет собой уровень, обладающий определённым набором параметров и хранящий в себе всё, что находится на игровом уровне. Для его представления был выбран класс Level, экземпляр которого генерирующийся каждый раз при старте игры.

```
class Level {
private:
    int h;
```

Высота карты

```
    int w;
```

Ширина карты

```
    int** map;
```

Двумерный массив представляющий собой карту уровня

```
public:
    Player p1;
```

Объект игрока 1

```
    Player p2;
```

Объект игрока 2

```
    QString p1name;
```

Имя 1 игрока

```
    QString p2name;
```

Имя 1 игрока

```
    QString mapName;
```

Название уровня

```
    Enemy* enemies;
```

Массив врагов

```
    int difficulty;
```

Сложность

```
    int enemiesCount;
```

Кол-во врагов на уровне

```
    int coinsCount;
```
Кол-во собранных монеток
```
    bool p2enabled;
```
Активен ли 2 игрок
```
    int p1Score;
```
Очки 1 игрока
```
    int p2Score;
```
Очки 2 игрока
```
    int score;
```
Общие очки
```
    int maxLives;
```
Максимальное кол-во жизней
```
    navCell** navMap;
```
Двумерный массив для поиска пути от врагов к игрокам
```
};
```

Поля класса Level хранят в себе классы Player и Enemy, являющимися наследниками класса Movable.

```
class Movable {
protected:
    int x;
```
Координата x на игровой карте
```
    int y;
```
Координата y на игровой карте
```
    int memAnim;
```
Сохранение анимации
```
    Direction direction;
```
Направление движения
```
public:
    float speed;
```
Скорость
```
    float movePhase;
```

```
Степень сдвига относительно точки на карте в сторону движения
};

class Player: public Movable { [3]
private:
    Direction memoryDirection;
Запоминание направления движения
    int spawnX;
Изначальные координаты появления игрока по x
    int spawnY;
Изначальные координаты появления игрока по y
public:
    int lives;
Текущее кол-во жизней
    bool targetable;
Может ли враг навредить игроку
};

class Enemy: public Movable {
public:
    int color;
Цвет врага
};
```

Для отображения рекордов используется класс Records и вспомогательные классы Entry1, Entry2 означающие одну запись в таблице рекордов для одного или двух игроков соответственно.

```
class Records {
public:
    std::map<QString,Entry1> map1pl;
Map для хранения результатов 1 игрока [4]
    std::map<QString,Entry2> map2pl;
```

Map для хранения результатов 2 игроков

```
    QTableWidget* table;
```

Таблица рекордов

```
};

class Entry1 {
public:
    QString map;
```

Название карты

```
    QString pl1name;
```

Имя 1 игрока

```
    int difficulty;
```

Сложность

```
    int lives;
```

Кол-во жизней

```
    int pl1score;
```

Очки 1 игрока

```
    float time;
```

Время прохождения уровня

```
};

class Entry2: public Entry1 {
public:
    QString pl2name;
```

Имя 2 игрока

```
     int pl2score;
```

Очки 2 игрока

```
};
```

Для удобного обозначения сторон используется класс Direction.

```
class Direction {
public:
    short horizontal;
```

Горизонтальное направл.

```
    short vertical;
```

Вертикальное направл.

```
};
```

2.2 Разработка модели поведения объекта

Приложение работает с объектом класса Level и его полями при игре и с классом Records, Entry1 и Entry2 при просмотре рекордов.

```
void Game::Game (QWidget *parent = nullptr)
```

(п. 2.5 рисунок 17 game.h)

Конструктор класса Game. Вместе с ним создаётся объект класса Level по данным, полученным из конструктора Game.

```
void Game::nextFrame()
```

Вызывается каждый раз перед отрисовкой кадра и изменяет состояния объектов на уровне

```
void Game::paintEvent(QPaintEvent *event)
```

Вызывается при отрисовке кадра. [5]

```
void Game::endGame()
```

Вызывается при завершении игры, когда у игрока не осталось жизней или, когда все монетки собраны. Выводит на экран результаты после игры.

```
Level::Level(QString filename, bool mode, QString p1, QString p2, int difficulty, int lives)
```

(п. 2.5 рисунок 17 level.h)

Создаёт уровень

`filename` – имя файла из которого следует создать уровень

`mode` – режим игры со вторым игроком если true

`p1, p2` – имена игроков

`difficulty` – сложность

`lives` – кол-во жизней

```
void Level:: recreateNavMap()
```

Обновляет карту поиска пути

`void Level::restoreNavMap()`

Очищает карту поиска пути

Для манипуляций с игроком и врагами используются методы:

`void Movable::move(int h, int w)`

(п. 2.5 рисунок 17 movable.h)

Передвигает игрока или врага;

h, w – размеры уровня необходимые для передвижения с одного края карты в другой.

`Player::Player(int lives, int x, int y, Direction dir)`

(п. 2.5 рисунок 17 player.h)

`x, y` – координаты

`lives` – кол-во жизней

`dir` – направление движения

`void Player::toSpawn()`

Перемещает игрока на его изначальное место появления;

`Record::Records(QWidget *parent = nullptr);`

(п. 2.5 рисунок 17 records.h)

Конструктор класса Record. Вместе с ним считываются данные методом load и заполняется таблица методом fill1player.

`void Records::load()`

Загружает данные о рекордах из файлов;

`void Records::fill1player()`

Переключает отображение рекордов для одного игрока;

`void Records::fill2player()`

Переключает отображение рекордов для двух игроков;

`void Entry1::addTo (QTableWidget* table)`

Добавление записи в таблицу `table`

2.3 Формализация расчетов

Основные алгоритмы, обеспечивающие стабильную работу программы реализуют методы, описанные в модулях level.h, movable.h и records.h.

```
void Movable::move(int h, int w);
```

Общий метод для игроков и врагов передвигающий их по игровой карте. Алгоритм метода представлен на рисунке 3.

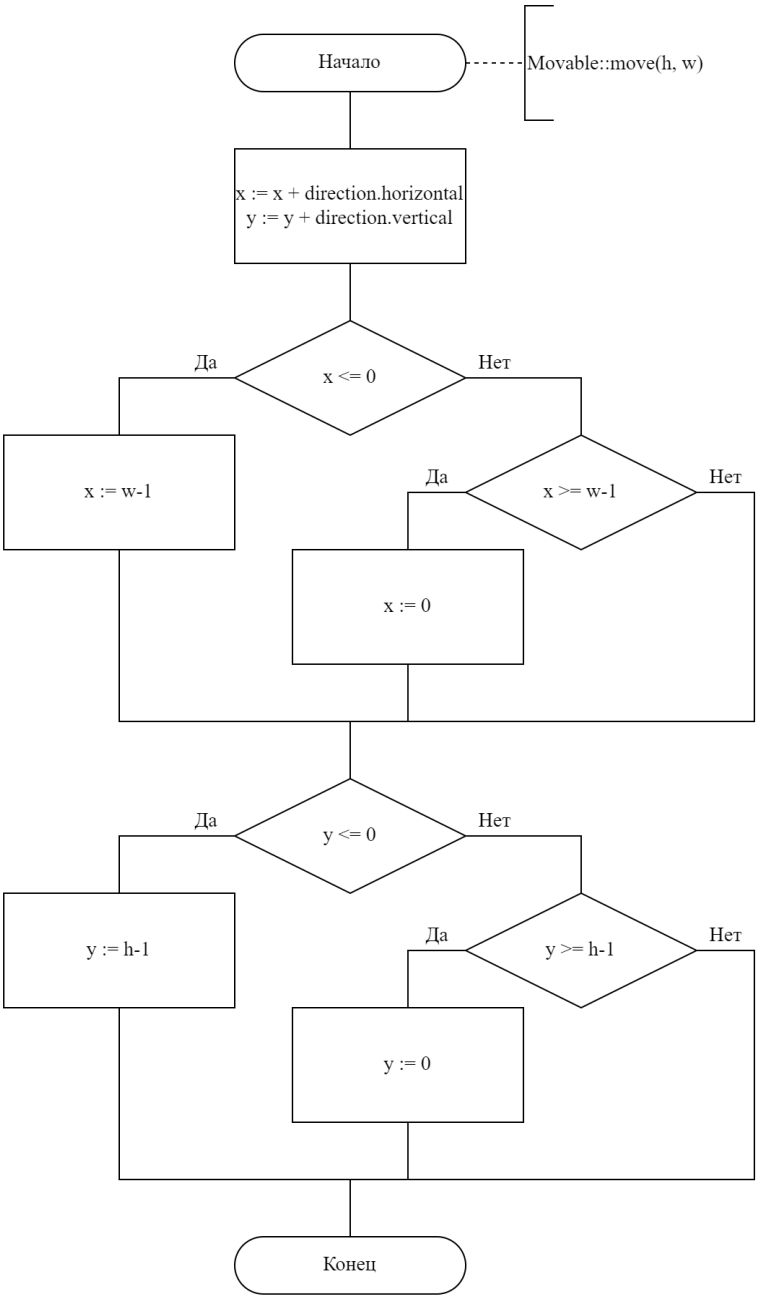

Рисунок 3 – Алгоритм метода void Movable::move(int h, int w)

```
void Level::restoreNavMap();
```

Очищает карту поиска пути. Алгоритм метода представлен на рисунке 4.

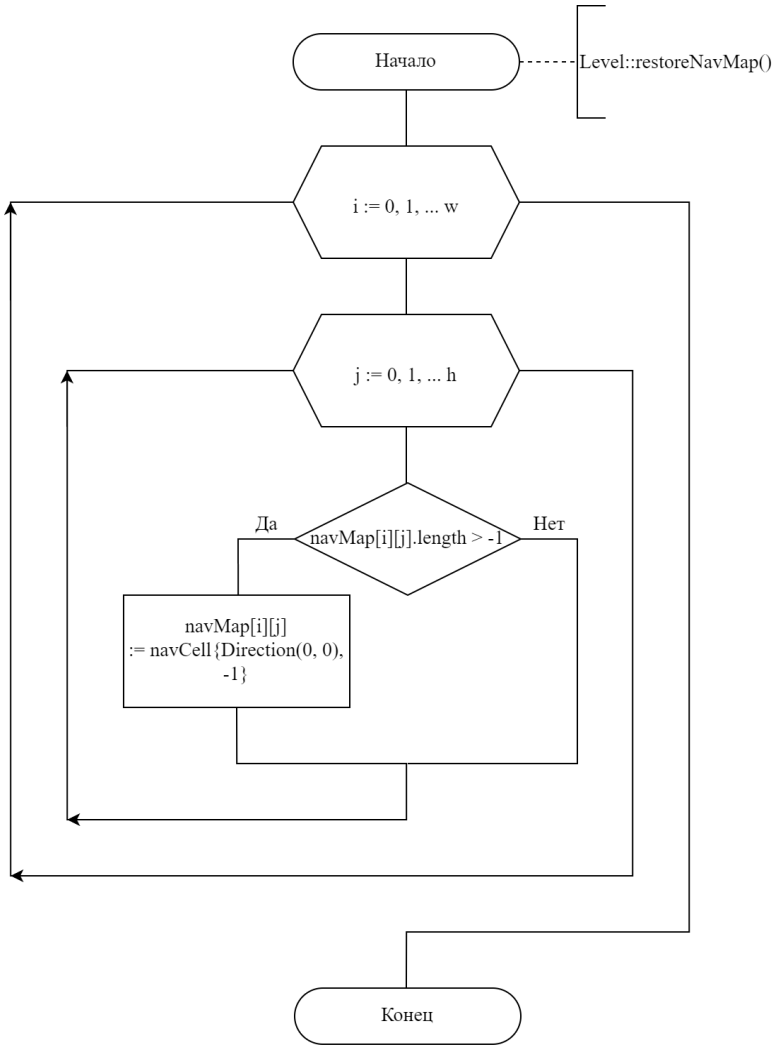

Рисунок 4 – Алгоритм метода void Level::restoreNavMap()

```
void Level::recreateNavMap();
```

Пересоздаёт карту поиска пути при изменении положения объектов на игровом уровне. Алгоритм метода представлен на рисунке 5.

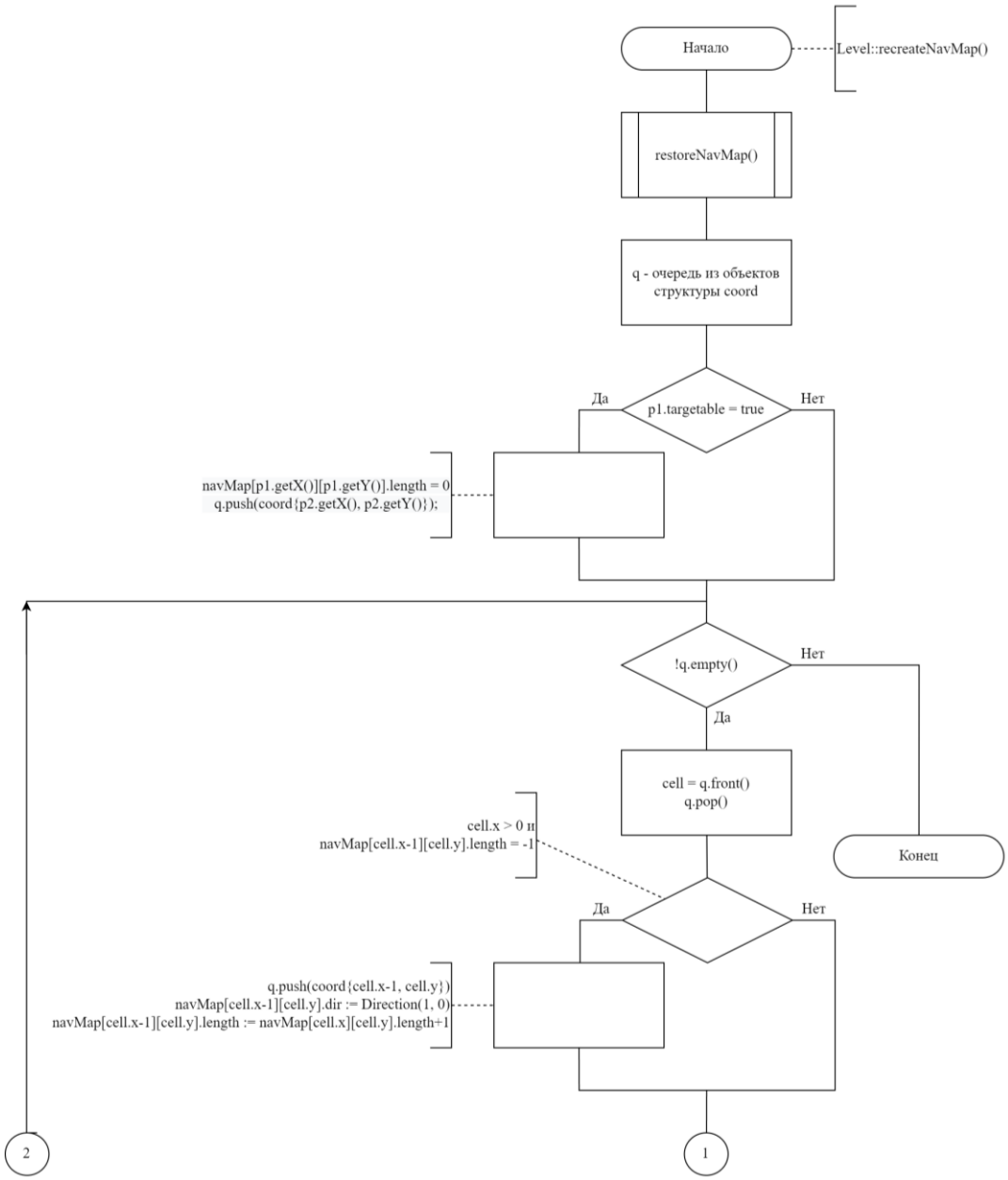

Рисунок 5 – Алгоритм метода void Level::recreateNavMap()

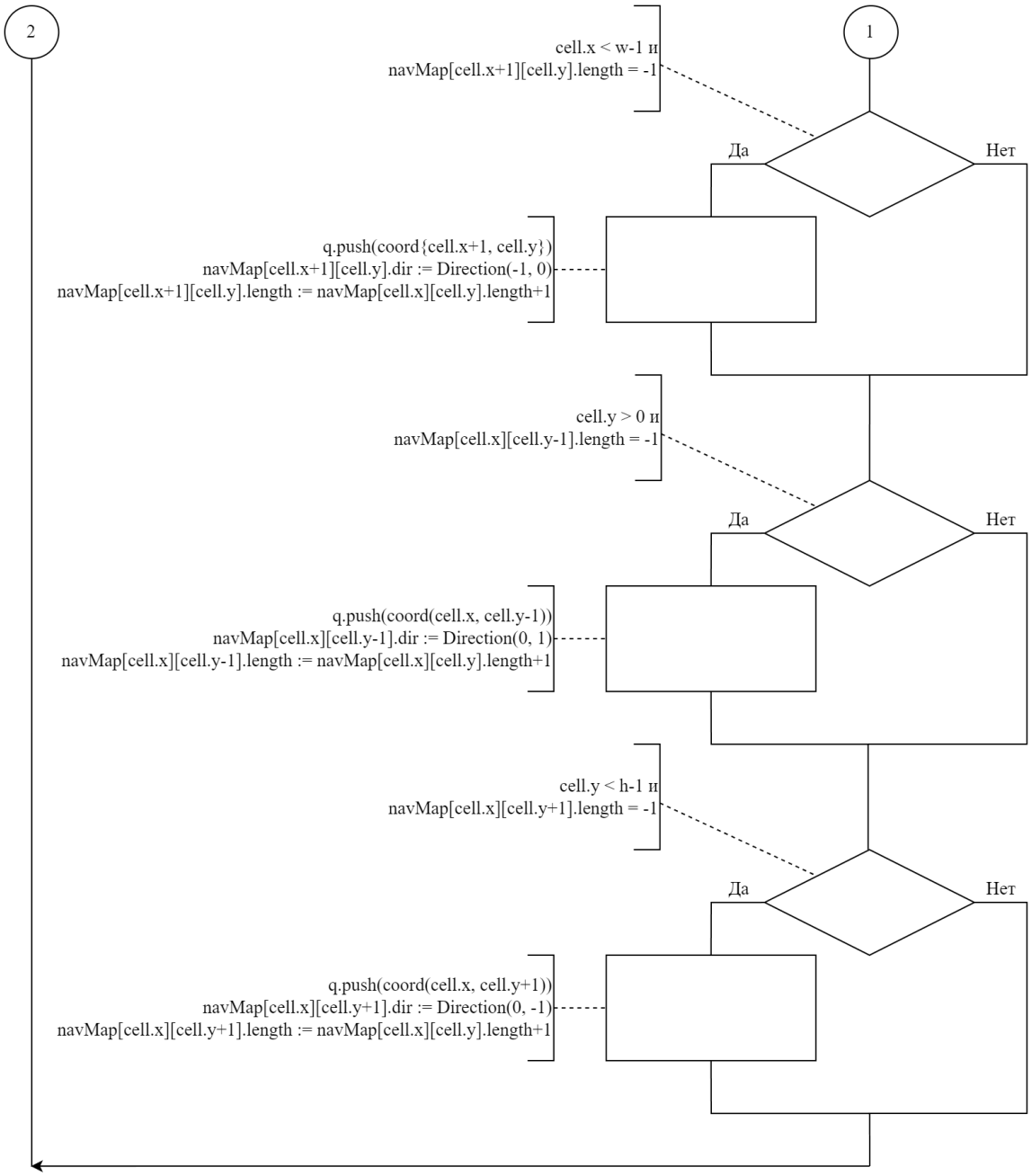

Рисунок 5 – продолжение

```
void Records::load();
```

Метод загружает из разных файлов, хранящихся в директории приложения, рекорды для одного и двух игроков. Алгоритм метода представлен на рисунке 6.

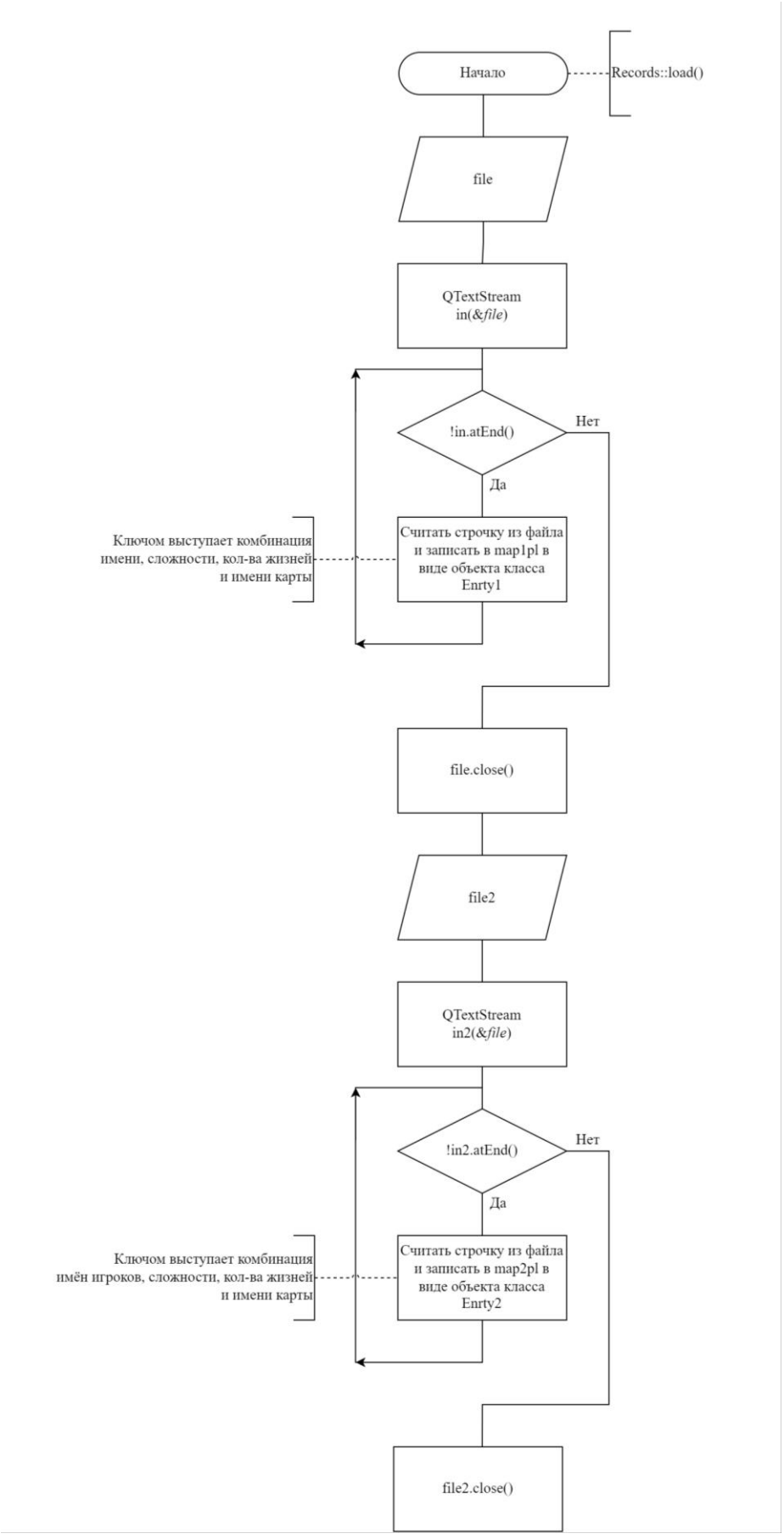

Рисунок 6 – алгоритм метода void Recrds::load()

```
void fill1player();
```

Метод выводит все рекорды одного игрока на таблицу на экране. Алгоритм метода представлен на рисунке 7.

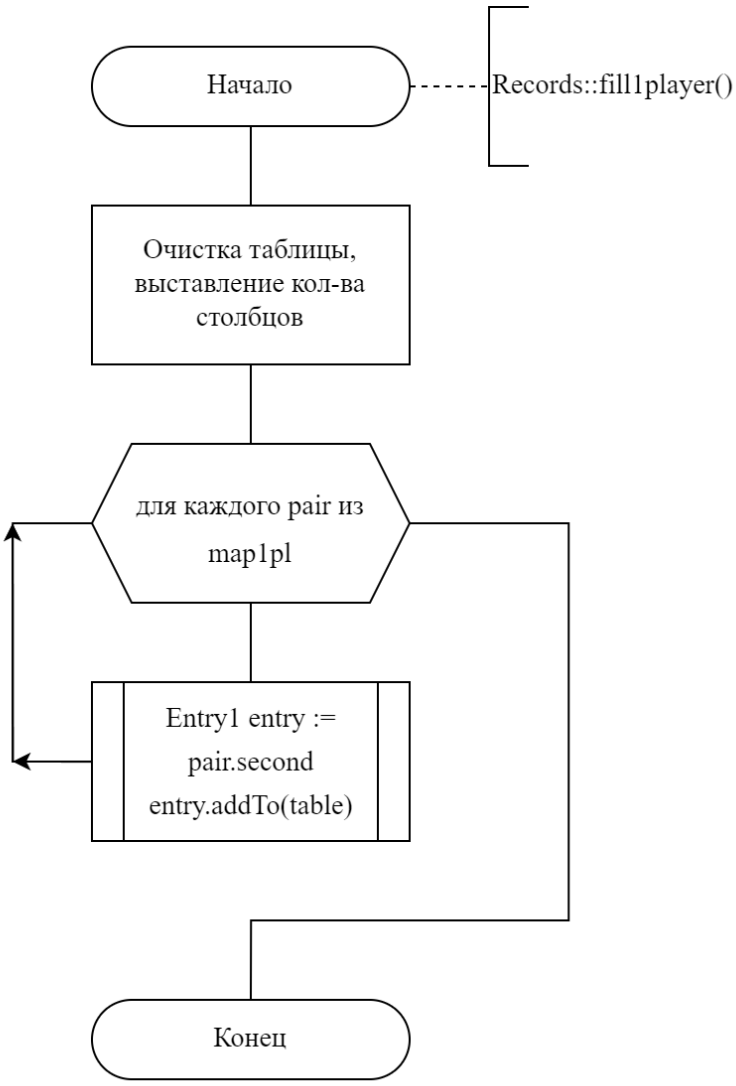

Рисунок 7 – Алгоритм метода void Records::fill1player()

```
void Records::fill2players();
```

Метод выводит все рекорды двух игроков на таблицу на экране. Работает аналогично void Records::fill1player().

```
void Entry1::addTo(QTableWidget* table);
```

Метод добавляет для одного игрока элемент в таблицу. Алгоритм метода представлен на рисунке 8.

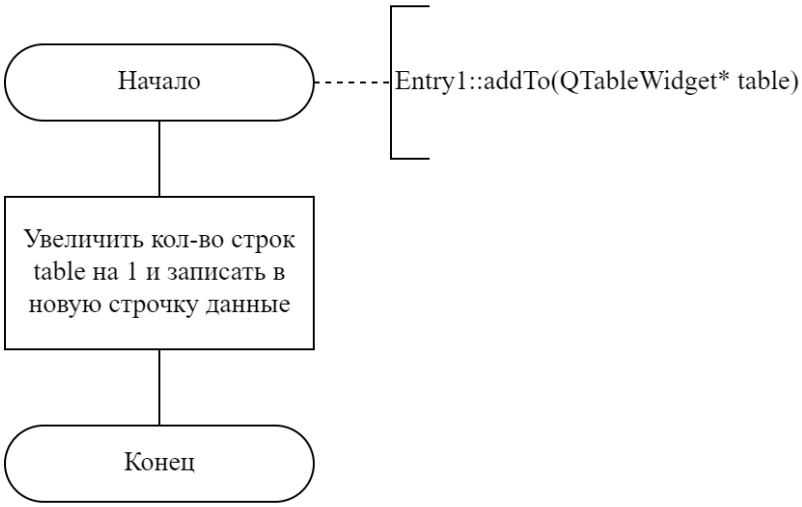

Рисунок 8 – Алгоритм метода Entry1::addTo(QTableWidget *table)

```
void Entry2::addTo(QTableWidget* table);
```

Метод добавляет для двух игроков в таблицу. Работает аналогично Entry2::addTo(QTableWidget *table).

На рисунке 9 представлена UML диаграмма классов проекта. (приложение Б.1)

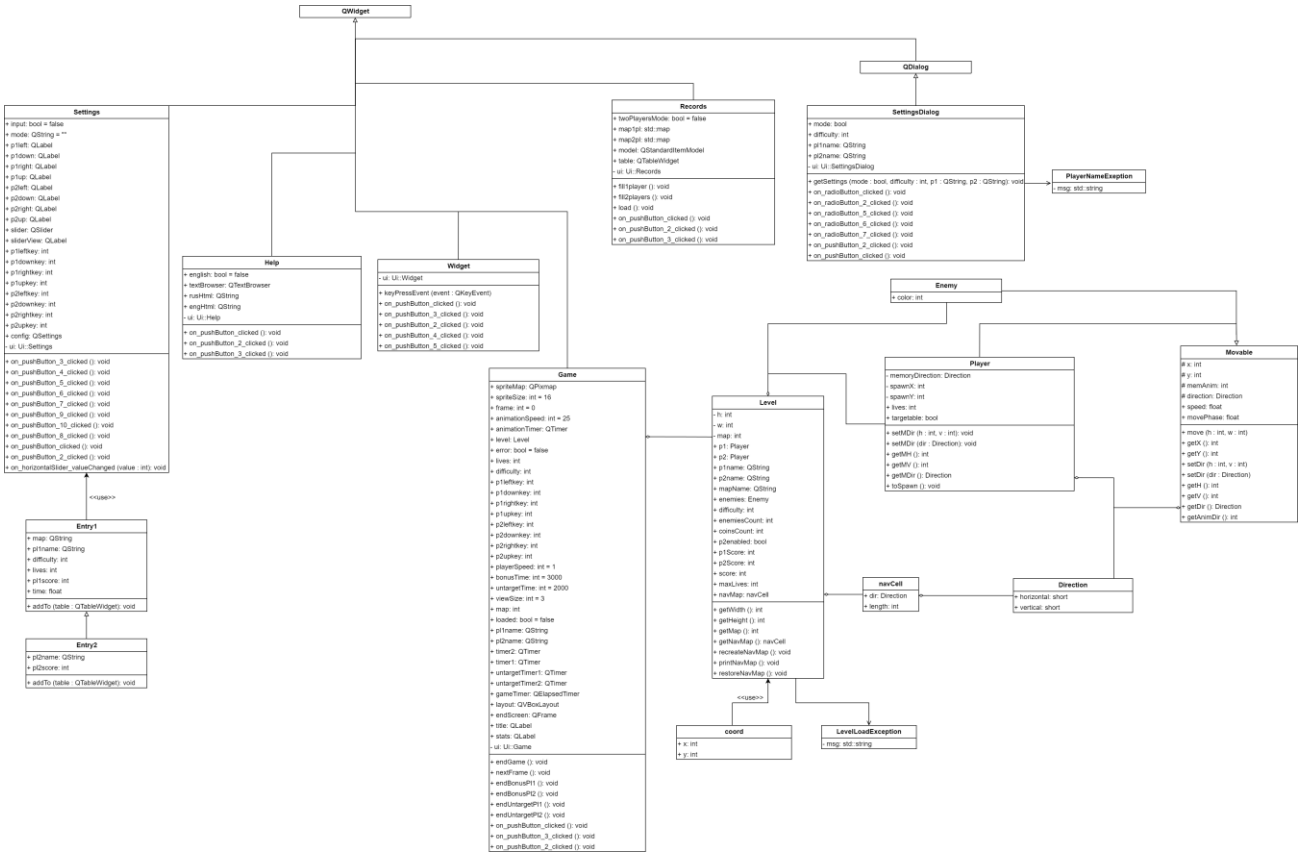

Рисунок 9 – UML диаграмма проекта

2.4 Разработка интерфейса приложения

Для решение поставленной задачи было создано приложение, интерфейс которого включает в себя 7 окон.

Окно Widget имеет вид, представленный на рисунке 9.

Рисунок 10 – окно Widget

Значения свойств компонентов окна Widget представлены в таблице 1.

Таблица 1 – Свойства компонентов окна Widget

Имя компонента	Свойство		Значение
Widget	windowTitle		Menu
label_2	text		TextLabel
	font	size	36
		family	Press Start P2
label	text		PAC-MAN
	font	size	20
		family	Press Start P2
playButton	text		PLAY
	font	size	20
		family	Press Start P2
recordsButton	text		RECORDS
	font	size	20
		family	Press Start P2
settingsButton	text		SETTINGS
	font	size	20
		family	Press Start P2
helpButton	text		HELP
	font	size	20
		family	Press Start P2
exitButton	text		EXIT
	font	size	20
		family	Press Start P2

Обработчики событий и их назначение представлены в таблице 2.

Таблица 2 – Обработчики событий и их назначение окна Widget

Название обработчика	Назначение
`void Widget::on_playButton_clicked()`	Запуск игры
`void Widget::on_exitButton_clicked()`	Выход из приложения
`void Widget::on_settingsButton_clicked()`	Настройки управления и кол-ва жизней
`void Widget::on_recordsButton_clicked()`	Просмотр рекордов
`void Widget::on_helpButton_clicked()`	Просмотр окна помощи

Окно Game имеет вид, представленный на рисунке 10.

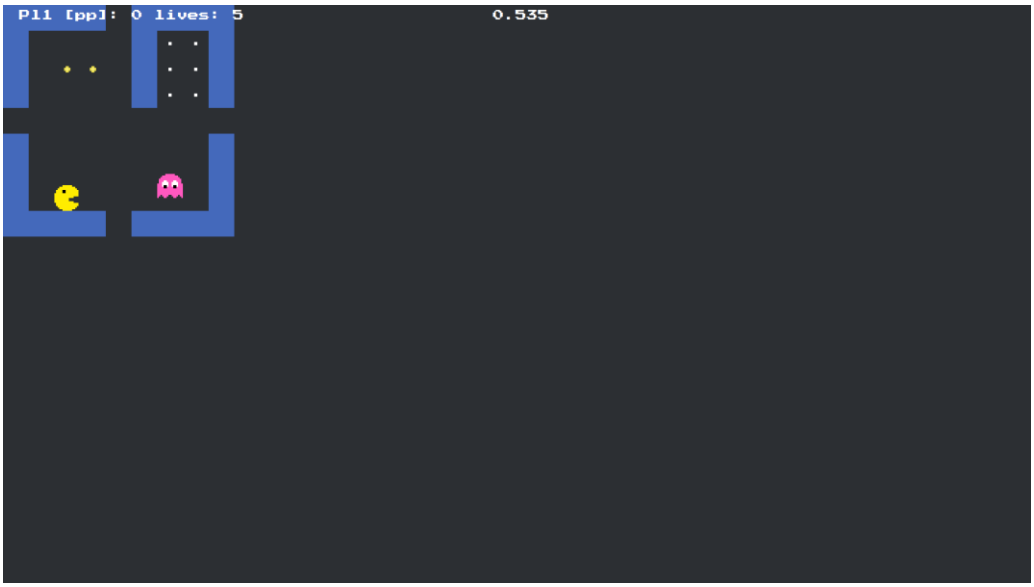

Рисунок 11 – окно Game во время игры

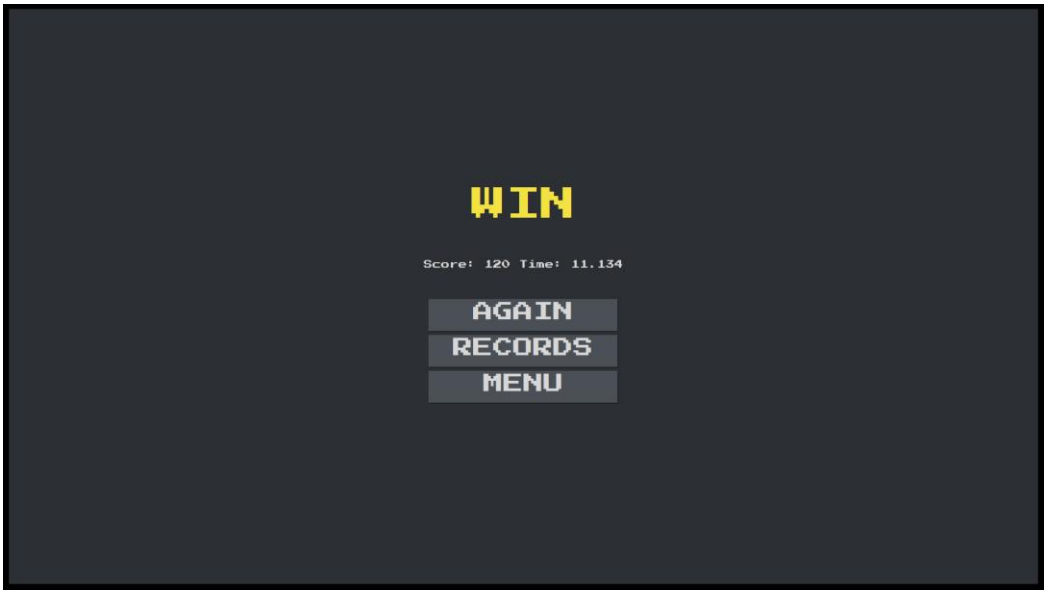

Рисунок 12 – окно Game после игры

Значения свойств компонентов формы Game представлены в таблице 3.

Таблица 3 – Свойства компонентов окна Game

Имя компонента	Свойство		Значение
Game	windowTitle		Game
againButton	text		MENU
	font	size	28
		family	Press Start P2
menuButton	text		MENU
	font	size	28
		family	Press Start P2

Продолжение таблицы 3

recordsButton	text		RECORDS
	font	size	28
		family	Press Start P2
label	text		WIN
	font	size	48
		family	Press Start P2
label_2	text		TextLabel
	font	size	12
		family	Press Start P2

Назначение компонентов ясно из текста на них и их расположения на форме Game. Обработчики событий и их назначение представлены в таблице 4.

Таблица 4 – Обработчики событий и их назначение формы Game

Название обработчика	Назначение
`void Game::on_againButton_clicked()`	Запуск игры заново
`void Game::on_recordsButton_clicked()`	Переход к рекордам
`void Game::on_menuButton_clicked()`	Переход в меню

Окно settingsDialog имеет вид, представленный на рисунке 11.

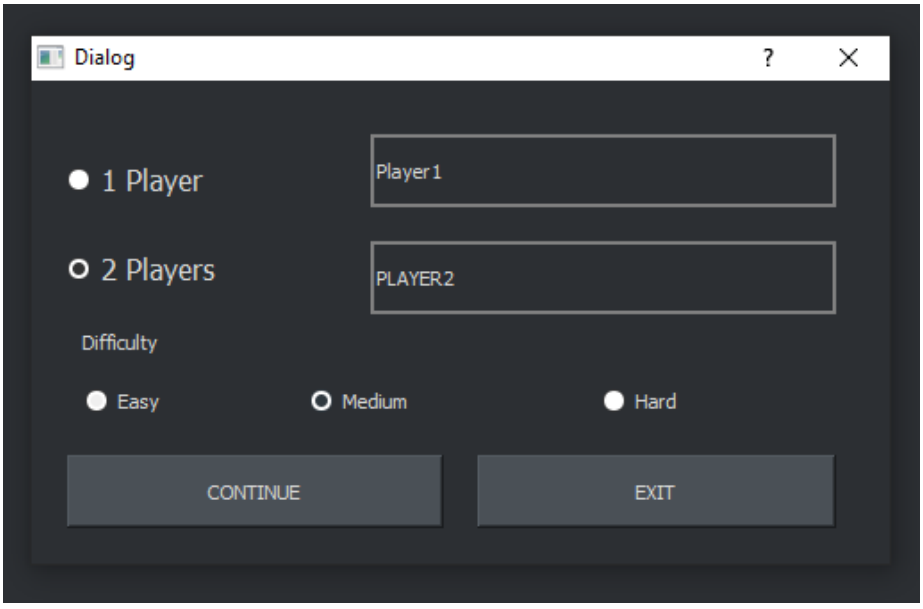

Рисунок 13 – окно settingsDialog

Значения свойств компонентов окна settingsDialog представлены в таблице 5.

Таблица 5 – Свойства компонентов окна settingsDialog

Имя компонента	Свойство	Значение
settingsDialog	windowTitle	Dialog

Продолжение таблицы 5

ContinueButton	text		CONTINUE
	font	size	12
exitButton	text		EXIT
	font	size	12
radioButton	text		1 Player
RadioButton_2	text		2 Players
RadioButton_5	text		Easy
RadioButton_6	text		Medium
RadioButton_7	text		Hard
lineEdit	placeholderText		PL1 NAME
lineEdit_2	placeholderText		PL2 NAME

Назначение компонентов ясно из текста на них и их расположения на форме settingsDialog. Обработчики событий и их назначение представлены в таблице 6.

Таблица 6 – Обработчики событий и их назначение формы settingsDialog

Название обработчика	Назначение
`void SettingsDialog::on_radioButton_clicked()`	Установка режима одного игрока
`void SettingsDialog::on_radioButton_2_clicked()`	Установка режима двух игроков
`void SettingsDialog::on_radioButton_5_clicked()`	Переключение на лёгкую сложность
`void SettingsDialog::on_radioButton_6_clicked()`	Переключение на среднюю сложность
`void SettingsDialog::on_radioButton_7_clicked()`	Переключение на тяжёлую сложность
`void SettingsDialog::on_continueButton_clicked()`	Завершение работы с диалоговым окном
`void SettingsDialog::on_exitButton_clicked()`	Отмена запуска игры

Рисунок 14 – окно Settings

Значения свойств компонентов окна Settings представлены в таблице 7.

Таблица 7 – Свойства компонентов окна Settings

Имя компонента	Свойство		Значение
Settings	windowTitle		Settings
label	text		PAC-MAN
	font	size	36
		family	Press Start P2
Label_2	text		SETTINGS
	font	size	16
		family	Press Start P2
Label_3	text		LIVES:
	font	size	16
		family	Press Start P2
horizontalSlider	minimum		1
	maximum		50
label_4	text		PLAYER 1
	font	size	16
		family	Press Start P2
label_5	text		PLAYER 2
	font	size	16
		family	Press Start P2
label_6	text		left
	font	size	16
		family	Press Start P2
label_7	text		down
	font	size	16
		family	Press Start P2
label_8	text		right
	font	size	16
		family	Press Start P2

Продолжение таблицы 7

label_9	text		up
	font	size	16
		family	Press Start P2
label_10	text		left
	font	size	16
		family	Press Start P2
label_11	text		left
	font	size	16
		family	Press Start P2
label_12	text		left
	font	size	16
		family	Press Start P2
label_13	text		left
	font	size	16
		family	Press Start P2
label_14	text		left
	font	size	16
		family	Press Start P2
label_15	text		left
	font	size	16
		family	Press Start P2
label_16	text		up
	font	size	16
		family	Press Start P2
label_17	text		left
	font	size	16
		family	Press Start P2
label_18	text		left
	font	size	16
		family	Press Start P2
label_19	text		left
	font	size	16
		family	Press Start P2
label_20	text		right
	font	size	16
		family	Press Start P2
label_21	text		down
	font	size	16
		family	Press Start P2
label_22	text		0
	font	size	16
		family	Press Start P2
saveButton	text		SAVE
backButton	text		BACK
pushButton_3	text		enter
pushButton_4	text		enter
pushButton_5	text		enter
pushButton_6	text		enter

Продолжение таблицы 7

pushButton_7	text	enter
pushButton_8	text	enter
pushButton_9	text	enter
pushButton_10	text	enter

Назначение компонентов ясно из текста на них и их расположения на форме Settings. Обработчики событий и их назначение представлены в таблице 8.

Таблица 8 – Обработчики событий и их назначение окна Settings

Название обработчика	Назначение
`void SettingsDialog::on_saveButton_clicked()`	Сохранение настроек
`void SettingsDialog::on_backButton_clicked()`	Выход в главное меню
`void SettingsDialog::on_pushButton_3_clicked()`	Выбор клавиши для передвижения влево 1 игрока
`void SettingsDialog::on_pushButton_4_clicked()`	Выбор клавиши для передвижения вниз 1 игрока
`void SettingsDialog::on_pushButton_5_clicked()`	Выбор клавиши для передвижения вправо 1 игрока
`void SettingsDialog::on_pushButton_6_clicked()`	Выбор клавиши для передвижения вверх 1 игрока
`void SettingsDialog::on_pushButton_7_clicked()`	Выбор клавиши для передвижения влево 2 игрока
`void SettingsDialog::on_pushButton_8_clicked()`	Выбор клавиши для передвижения вниз 2 игрока
`void SettingsDialog::on_pushButton_9_clicked()`	Выбор клавиши для передвижения вправо 2 игрока
`void SettingsDialog::on_pushButton_10_clicked()`	Выбор клавиши для передвижения вверх 2 игрока

Окно Help имеет вид, представленный на рисунке 15.

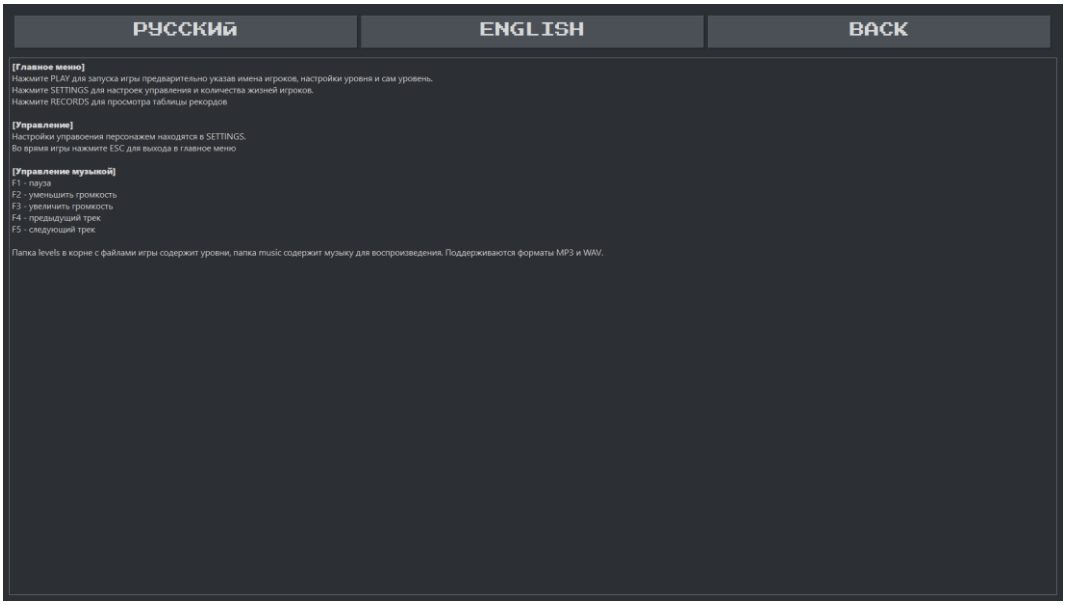

Рисунок 15 – окно Help

Значения свойств компонентов окна Settings представлены в таблице 9.

Таблица 9 – Свойства компонентов окна Help

Имя компонента	Свойство		Значение
Help	windowTitle		Help
backButton	text		BACK
rusButton	text		РУССКИЙ
engButton	text		ENGLISH
textBrowser	font	size	9
		family	Segoe UI

Назначение компонентов ясно из текста на них и их расположения на форме Help. Обработчики событий и их назначение представлены в таблице 8.

Таблица 10 – Обработчики событий и их назначение окна Help

Название обработчика	Назначение
`void SettingsDialog::on_engButton_clicked()`	Переключение на английский язык
`void SettingsDialog::on_rusButton_clicked()`	Переключение на русский язык
`void SettingsDialog::on_backButton_clicked()`	Выход в главное меню

Окно Records имеет вид, представленный на рисунке 16.

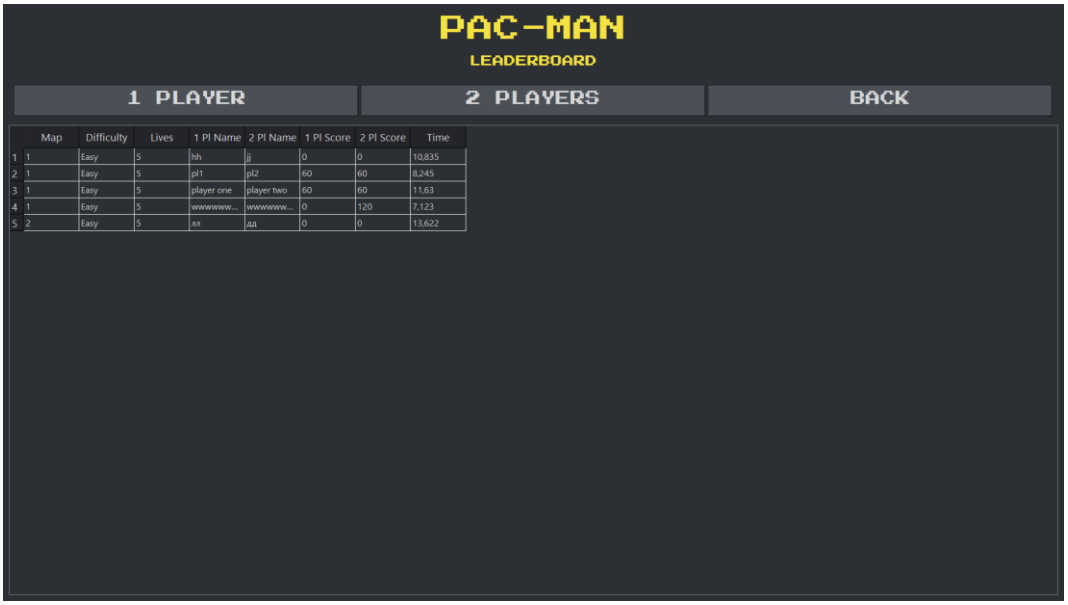

Рисунок 16 - окно Records

Значения свойств компонентов окна Records представлены в таблице 11.

Таблица 11 – Свойства компонентов окна Records

Имя компонента	Свойство		Значение
Records	windowTitle		Records
backButton	text		BACK
oneButton	text		1 PLAYER
twoButton	text		2 PLAYERS
label	text		PAC-MAN
	font	size	36
		family	Press Start P2
label_2	text		LEADERBOARD
	font	size	16
		family	Press Start P2

Назначение компонентов ясно из текста на них и их расположения на форме Records. Обработчики событий и их назначение представлены в таблице 12.

Таблица 12 – Обработчики событий и их назначение окна Records

Название обработчика	Назначение
`void Records::on_oneButton_clicked()`	Переключение на режим просмотра рекордов двух игроков

Продолжение таблицы 12

`void Records::on_twoButton_clicked()`	Переключение на режим просмотра рекордов одного игрока
`void SettingsDialog::on_backButton_clicked()`	Выход в главное меню

2.5 Описание структуры приложения

Приложение состоит из следующих модулей:

main.cpp – модуль отвечающий за инициализацию и запуск графического интерфейса. Так же на его уровне находится музыкальный плеер, который активен вне зависимости от окна с которым взаимодействует пользователь.

widget.cpp, widget.h – модуль содержащий логику окна главного меню, которое позволяет переходить на другие окна.

settings.cpp, settings.h – модуль содержащий логику окна настроек. Считывает и записывает настройки из файлов программы.

records.cpp, records.h – модуль содержащий логику окна рекордов. Считывает и отбирает лучшие рекорды из файлов программы и выводит их в таблицу.

help.cpp, help.h – модуль содержащий логику окна помощи. Переключает язык справочной информации.

game.cpp, game.h – модуль содержащий логику окна игры. Отвечает за всю игровую логику, вывод результатов игры и запись их в соответствующий файл.

settingsdialog.cpp, settingsdialog.h – модуль содержащий логику диалогового окна настроек появляющимся перед началом игры. Принимает имена игроков, сложность и файл уровня.

level.h – модуль содержащий в себе класс Level и содержащий всю необходимую логику для взаимодействия с игровым уровнем.

movable.h – модуль содержащий в себе класс Movable и его производные классы Player и Enemy. Отвечает за логику движущихся на игровом уровне объектов.

direction.h – модуль содержащий в себе вспомогательный класс Direction. Он необходим для задания направления для непосредственно движения или его планирования при создании карты поиска пути.

Диаграмма компонентов приложения представлена на рисунке 16. (приложение Б.2)

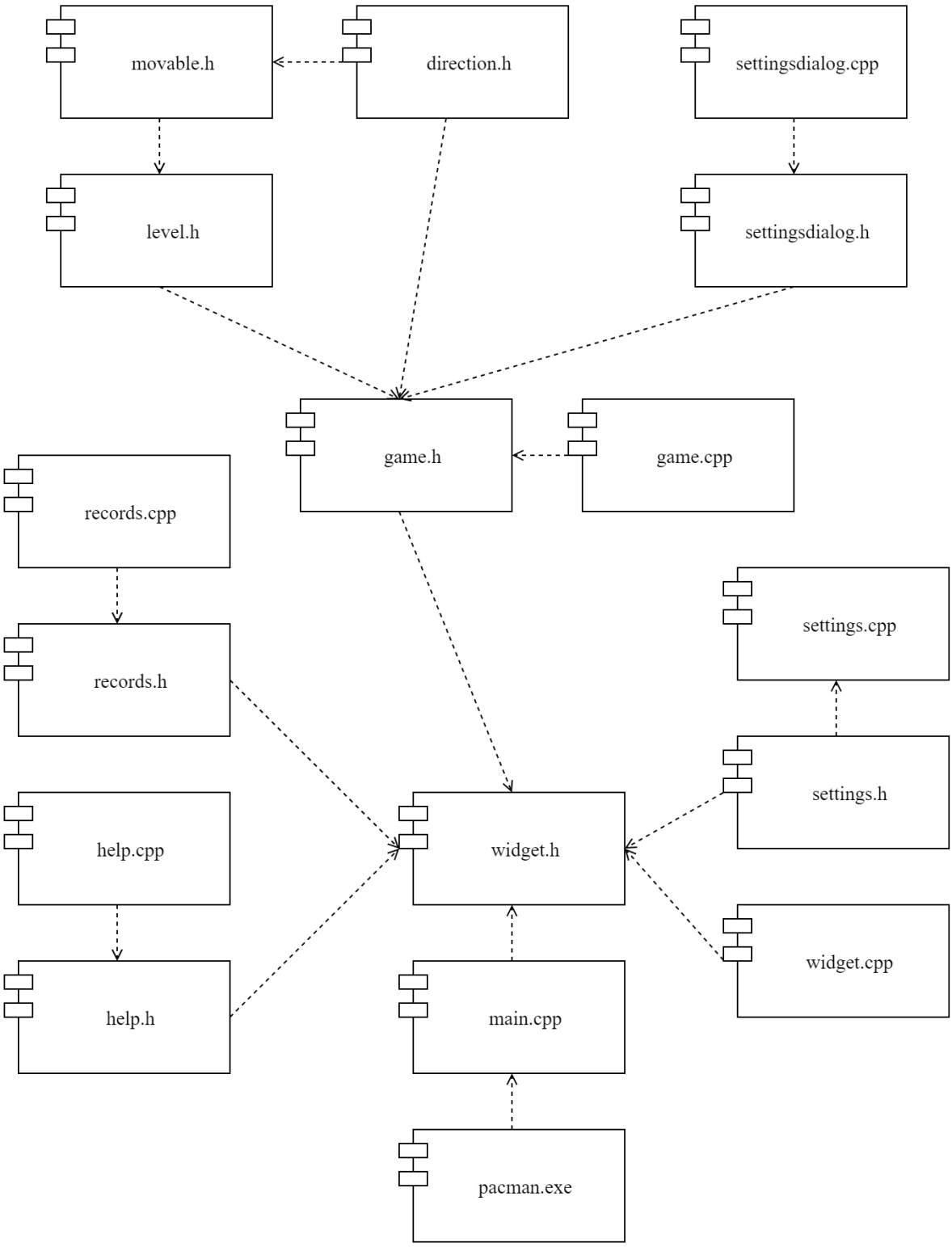

Рисунок 17 – Диаграмма компонентов

3 Анализ проекта и тестирование

3.1 Защита приложения от ошибок пользователя

С целью предотвращения некорректной работы приложения в результате действий пользователя были использованы следующие средства:

1. При некорректном имени игрока (т.е. использование недопустимых символов или передача пустой строки), выбрасывается исключение, которое, после обработки, показывает сообщение о ошибке. Неверная информация при этом игнорируется.

```
class PlayerNameExeption : public std::exception{
    std::string msg; // сообщение об ошибке
public:
    const char * what() const noexcept override {
        return msg.c_str();
    }
    PlayerNameExeption(std::string msg) {this->msg = msg;};
          };
```

2. При передаче файла уровня, содержащего недопустимые символы или повреждённого файла, выбрасывается исключение, которое, после обработки, показывает сообщение о ошибке. Неверная информация при этом игнорируется.

```
class LevelLoadException : public std::exception {
    std::string msg;
public:
    const char * what() const noexcept override {
        return msg.c_str();
    }
    LevelLoadException(std::string msg) {this->msg = msg;};
};
```

Приложение отслеживает возможные ошибки ввода данных, предотвращает совершение ошибок пользователя при выполнении действий над данными. Таким образом, можно считать, что приложение имеет достаточный уровень защиты от ошибок пользователя.

3.2 Тестирование приложения

Проверка функционирования приложения в нормальных условиях. Имена игроков не пустые и не содержат запрещённые символы, файл с уровнем не повреждён и не использует недопустимые символы. Тесты на рисунках 18-21.

1) Начать игру нажатием кнопки PLAY.

Рисунок 18 – Тест в нормальных условиях

1) Выбрать сложность, кол-во игроков, ввести имена игроков и выбрать уровень из файла.

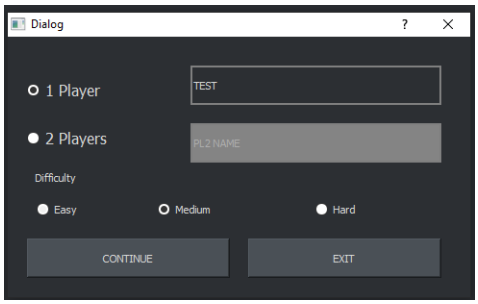

Рисунок 19 – Тест в нормальных условиях

2) Пройти уровень.

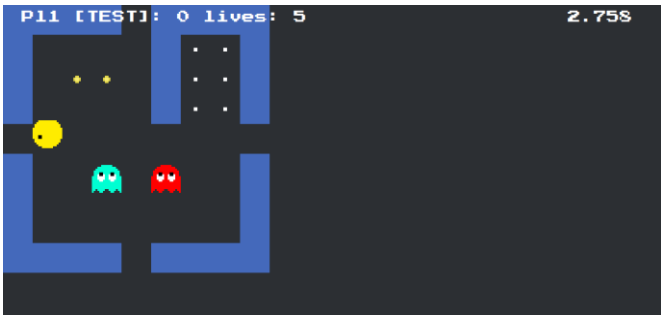

Рисунок 20 – Тест в нормальных условиях

3) На экран будут выведены результаты, а в рекордах появится соответствующая запись.

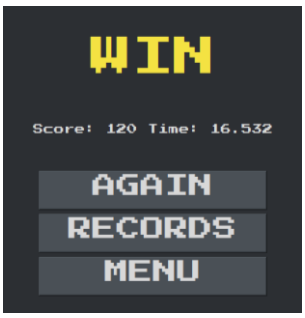

Рисунок 20 – Тест в нормальных условиях

3	1	Medium	5	TEST	120	16,539

Рисунок 21 – Тест в нормальных условиях

Проверка функционирования приложения в экстремальных условиях. Файл с уровнем содержит несколько игроков на игровом поле. Программа проигнорирует лишних игроков и продолжит своё выполнение. Тесты на рисунках 22-23.

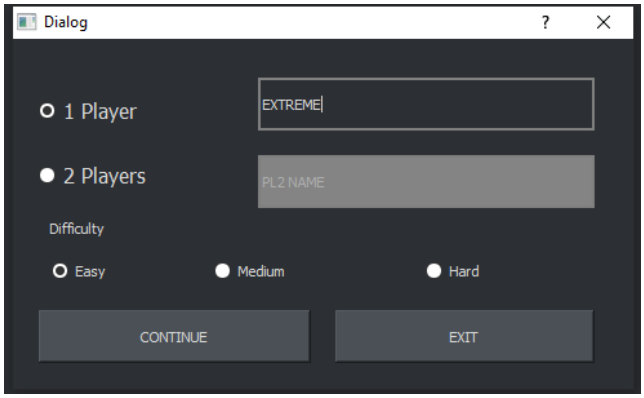

Рисунок 22 – Тест в экстремальных условиях

morePlayers	Easy	5	EXTREME	0	18,876

Рисунок 23 – Тест в экстремальных условиях

Проверка функционирования приложения в исключительных условиях. В этом случае выводится соответствующее предупреждение и происходит переход в главное меню.

Файл с уровнем не содержит игроков – Рисунок 24;

Файл повреждён – Рисунок 25;

Файл содержит недопустимые символы – Рисунок 26;

Имя игрока пустое – Рисунок 27;

Имя содержит недопустимые символы – Рисунок 28;

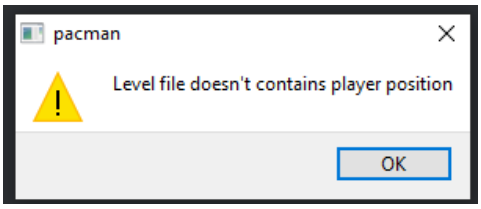

Рисунок 24 – Тест в исключительных условиях

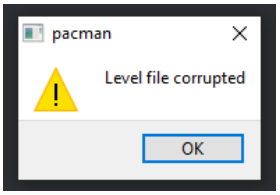

Рисунок 25 – Тест в исключительных условиях

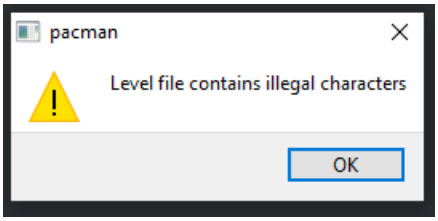

Рисунок 26 – Тест в исключительных условиях

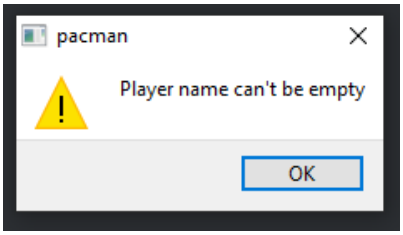

Рисунок 27 – Тест в исключительных условиях

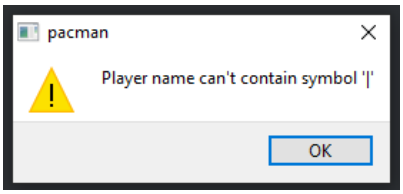

Рисунок 28 – Тест в исключительных условиях

ЗАКЛЮЧЕНИЕ

В процессе разработки программного обеспечения для решения конкретной задачи была изучена специфическая литература по теме проекта: грамотное и рациональное распределение задач, документация по среде разработки Qt Creator, документация по языку C++; был разработан алгоритм интеллектуальный алгоритм поиска пути противниками, интерфейс приложения, разработаны классы и продуманы их связи. Так же были изучены библиотеки языка, работающие с потоками файлов, вводом и выводом информации, воспроизведением звуковых файлов.

Программа может быть использована пользователем, для проведения своего досугового времени. Приложение позволяет играть нескольким пользователям одновременно.

Усовершенствовать данное приложения можно с помощью улучшения интерфейса, добавления полноценного редактора уровней и добавления режима игры по сети.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Алгоритм поиска в ширину [Сайт]. URL: https://ru.algorithmica.org/cs/shortest-paths/bfs/ (дата обращения 20.10.2022)
2. Шлее М. Qt 5.10. Профессиональное программирование на C++. — СПб.: БХВ-Петербург, 2018. — 1072 с.
3. Павловская, Т.А. C/C++. Программирование на языке высокого уровня. —СПб.: Питер, 2003. — 461с: ил.
4. Справка по C++ – std::map [Сайт]. URL: https://en.cppreference.com/w/cpp/container/map (дата обращения 12.10.2022)
5. Справка по Qt 6 – QPainter [Сайт]. URL: https://doc.qt.io/qt-6/qpainter.html (дата обращения 20.09.2022)

ПРИЛОЖЕНИЕ А

Текст программы

```
// main.cpp
#include "widget.h"

#include <QApplication>
#include <QMediaPlayer>
#include <QMediaPlaylist>
QMediaPlayer *musicPlayer;
int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    Widget w;
    QMediaPlaylist *playlist = new QMediaPlaylist();
    QDir musicDir = QCoreApplication::applicationDirPath() +
"/music/";
    QStringList musicList = musicDir.entryList(QStringList() <<
"*.mp3" << "*.MP3" << "*.wav" << "*.WAV",QDir::Files);
    foreach(QString filename, musicList) {
        playlist-
>addMedia(QUrl(QCoreApplication::applicationDirPath() +
"/music/" + filename));
    }

    playlist->setPlaybackMode(QMediaPlaylist::Loop);
    musicPlayer = new QMediaPlayer();
    musicPlayer->setPlaylist(playlist);
    musicPlayer->setVolume(5);
    musicPlayer->play();
    w.show();

    return a.exec();

}

void Widget::keyPressEvent( QKeyEvent *k )
{
    switch ( k->key() ) {
    case Qt::Key::Key_F4:
        musicPlayer->playlist()->previous();
        break;
    case Qt::Key::Key_F5:
        musicPlayer->playlist()->next();
        break;
    case Qt::Key::Key_F1:
        if (musicPlayer->state() == QMediaPlayer::PlayingState)
            musicPlayer->pause();
        else
            musicPlayer->play();
```

```
        break;
    case Qt::Key::Key_F2:
        if (musicPlayer->volume() >= 5)
            musicPlayer->setVolume(musicPlayer->volume()-5);
        break;
    case Qt::Key::Key_F3:
        if (musicPlayer->volume() <= 95 )
            musicPlayer->setVolume(musicPlayer->volume()+5);
        break;
    }
}
// widget.h
#ifndef WIDGET_H
#define WIDGET_H

#include <QWidget>
#include <game.h>
#include <settings.h>
#include <records.h>
#include <help.h>
#include <QFont>
#include <QFontDatabase>

QT_BEGIN_NAMESPACE
namespace Ui { class Widget; }
QT_END_NAMESPACE

class Widget : public QWidget
{
    Q_OBJECT

public:
    void keyPressEvent(QKeyEvent *event);
    Widget(QWidget *parent = nullptr); //окно главного меню
    ~Widget();



private slots:
    void on_exitButton_clicked();

    void on_helpButton_clicked();

    void on_recordsButton_clicked();

    void on_settingsButton_clicked();

    void on_playButton_clicked();

private:
    Ui::Widget *ui;
};
```

```
#endif

// widget.cpp
#include "widget.h"
#include "ui_widget.h"
#include <QMovie>
#include <QSound>

Widget::Widget(QWidget *parent)
    : QWidget(parent)
    , ui(new Ui::Widget)
{
    ui->setupUi(this);
    int id =
QFontDatabase::addApplicationFont(":/fonts/font.ttf");
    QString family =
QFontDatabase::applicationFontFamilies(id).at(0);
    QFont font = QFont(family);
    QWidget::showFullScreen();
    QMovie *movie = new QMovie(":/images/gif.gif");
    QLabel* label = ui->label_2;
    label->setMovie(movie);
    movie->start();

}
Widget::~Widget()
{
    delete ui;
}

void Widget::on_playButton_clicked()
{
    QSound::play(":/music/ok.wav");
    Game* game = new Game();
    game->show();
    this->close();
}

void Widget::on_exitButton_clicked()
{
    QSound::play(":/music/close.wav");
    qApp->exit();
}

void Widget::on_settingsButton_clicked()
{
    QSound::play(":/music/ok.wav");
    Settings* settings = new Settings();
    settings->show();
    this->close();
}
```

```
void Widget::on_recordsButton_clicked()
{
    QSound::play(":/music/ok.wav");
    Records* records = new Records();
    records->show();
    this->close();
}

void Widget::on_helpButton_clicked()
{
    QSound::play(":/music/ok.wav");
    Help* help = new Help();
    help->show();
    this->close();
}

// settingsdialog.h

#ifndef SETTINGSDIALOG_H
#define SETTINGSDIALOG_H

#include <QDialog>
#include <widget.h>

class PlayerNameExeption : public std::exception{ // ошибка в
имени игрока
    std::string msg; // сообщение об ошибке
public:
    const char * what() const noexcept override { // вывод
сообщения
        return msg.c_str();
    }
    PlayerNameExeption(std::string msg) {this->msg = msg;};
};

namespace Ui {
class SettingsDialog;
}

class SettingsDialog : public QDialog // диалоговое окно
настроек перед началом игры
{
    Q_OBJECT

public:
    explicit SettingsDialog(QWidget *parent = nullptr);
    bool mode; // режим игры 1/2 игрока
    int difficulty; // сложность
    QString pl1name; // имя 1 игрока
    QString pl2name; // имя 2 игрока
```

```
    void getSettings(bool& mode, int& difficulty, QString& p1,
QString& p2); // получение настроек из диалогового окна
    ~SettingsDialog();
private slots:
    void on_radioButton_clicked();

    void on_radioButton_2_clicked();

    void on_radioButton_5_clicked();

    void on_radioButton_6_clicked();

    void on_radioButton_7_clicked();

    void on_continueButton_clicked();

    void on_exitButton_clicked();


private:
    Ui::SettingsDialog *ui;
};

#endif

// settingsdialog.cpp

#include "settingsdialog.h"
#include "ui_settingsdialog.h"

SettingsDialog::SettingsDialog(QWidget *parent) :
    QDialog(parent),
    ui(new Ui::SettingsDialog)
{
    setWindowFlags(Qt::Dialog |
Qt::MSWindowsFixedSizeDialogHint);
    mode = false;
    difficulty = 1;
    ui->setupUi(this);
}

SettingsDialog::~SettingsDialog()
{
    delete ui;
}

void SettingsDialog::on_radioButton_clicked()
{
    ui->lineEdit_2->setEnabled(false);
    mode = false;
}
```

```
void SettingsDialog::on_radioButton_2_clicked()
{
    ui->lineEdit_2->setEnabled(true);
    mode = true;
}


void SettingsDialog::on_exitButton_clicked()
{
    this->reject();
}

void SettingsDialog::on_radioButton_5_clicked()
{
    difficulty = 1;
}

void SettingsDialog::on_radioButton_6_clicked()
{
    difficulty = 2;
}

void SettingsDialog::on_radioButton_7_clicked()
{
    difficulty = 3;
}

void SettingsDialog::on_continueButton_clicked()
{
    pl2name = ui->lineEdit_2->text();
    pl1name = ui->lineEdit->text();
    this->accept();

}

void SettingsDialog::getSettings(bool& m, int& difficulty,
QString& p1, QString& p2) {
    if (pl1name == "" || (mode && pl2name == "")) {
        throw PlayerNameExeption("Player name can't be empty");
    }
    if (pl1name.contains('|') || pl2name.contains('|')) {
        throw PlayerNameExeption("Player name can't contain
symbol '|'");
    }
    m = this->mode;
    p1 = this->pl1name;
    p2 = this->pl2name;
    difficulty = this->difficulty;
}

// settings.h
```

```
#ifndef SETTINGS_H
#define SETTINGS_H

#include <QWidget>
#include <QKeyEvent>
#include <QDebug>
#include <QLabel>
#include <widget.h>
#include <QSettings>
#include <QSlider>
namespace Ui {
class Settings;
}

class Settings : public QWidget // окно настроек
{
    Q_OBJECT

public:
    explicit Settings(QWidget *parent = nullptr);
    bool input = false; // активен ли ввод
    QString mode = ""; // какая кнопка настраивается в данный
момент
    QLabel *p1left;
    QLabel *p1down;
    QLabel *p1right;
    QLabel *p1up;
    QLabel *p2left;
    QLabel *p2down;
    QLabel *p2right;
    QLabel *p2up;
    QSlider *slider;
    QLabel *sliderView;
    int p1leftkey; // переменные для заданных клавиш
    int p1downkey;
    int p1rightkey;
    int p1upkey;
    int p2leftkey;
    int p2downkey;
    int p2rightkey;
    int p2upkey;
    QSettings* config; // для записи/чтения данных внешнего .ini
файла
    ~Settings();

private slots:
    void on_pushButton_3_clicked();

    void on_pushButton_4_clicked();

    void on_pushButton_5_clicked();
```

```
    void on_pushButton_6_clicked();

    void on_pushButton_7_clicked();

    void on_pushButton_9_clicked();

    void on_pushButton_10_clicked();

    void on_pushButton_8_clicked();

    void on_saveButton_clicked();

    void on_backButton_clicked();

    void on_horizontalSlider_valueChanged(int value);

private:
    void keyPressEvent(QKeyEvent *event) override;
    Ui::Settings *ui;
};

#endif

// settings.cpp

#include "settings.h"
#include "ui_settings.h"
#include <QSound>

Settings::Settings(QWidget *parent) :
    QWidget(parent),
    ui(new Ui::Settings)
{
    ui->setupUi(this);
    p1left = ui->label_10;
    p1down = ui->label_11;
    p1right = ui->label_12;
    p1up = ui->label_13;
    p2left = ui->label_15;
    p2down = ui->label_18;
    p2right = ui->label_19;
    p2up = ui->label_17;
    slider = ui->horizontalSlider;
    sliderView = ui->label_22;

    QWidget::showFullScreen();

    QString Path = QCoreApplication::applicationDirPath();
    QString endPath = Path + "/config/config.ini";
    QFileInfo fileinfo(endPath);
    if (fileinfo.isFile()) {
```

```
        config = new QSettings(endPath, QSettings::IniFormat);

        sliderView->setNum(config->value("lives", "").toInt());
        slider->setValue(config->value("lives", "").toInt());
        p1leftkey = config->value("p1left", "").toInt();
        p1left->setText(QKeySequence(p1leftkey).toString());
        p1downkey = config->value("p1down", "").toInt();
        p1down->setText(QKeySequence(p1downkey).toString());
        p1rightkey = config->value("p1right", "").toInt();
        p1right->setText(QKeySequence(p1rightkey).toString());
        p1upkey = config->value("p1up", "").toInt();
        p1up->setText(QKeySequence(p1upkey).toString());
        p2leftkey = config->value("p2left", "").toInt();
        p2left->setText(QKeySequence(p2leftkey).toString());
        p2downkey = config->value("p2down", "").toInt();
        p2down->setText(QKeySequence(p2downkey).toString());
        p2rightkey = config->value("p2right", "").toInt();
        p2right->setText(QKeySequence(p2rightkey).toString());
        p2upkey = config->value("p2up", "").toInt();
        p2up->setText(QKeySequence(p2upkey).toString());
    }

}

Settings::~Settings()
{
    delete ui;
}

void Settings::keyPressEvent(QKeyEvent *event) {
    if (input) {
        if (mode == "p1left") {
            p1left->setText(QKeySequence(event-
>key()).toString());
            p1leftkey = event->key();
        }
        if (mode == "p1down") {
            p1down->setText(QKeySequence(event-
>key()).toString());
            p1downkey = event->key();
        }
        if (mode == "p1right") {
            p1right->setText(QKeySequence(event-
>key()).toString());
            p1rightkey = event->key();
        }
        if (mode == "p1up") {
            p1up->setText(QKeySequence(event-
>key()).toString());
            p1upkey = event->key();
        }
        if (mode == "p2left") {
```

```
            p2left->setText(QKeySequence(event-
>key()).toString());
            p2leftkey = event->key();
        }
        if (mode == "p2down") {
            p2down->setText(QKeySequence(event-
>key()).toString());
            p2downkey = event->key();
        }
        if (mode == "p2right") {
            p2right->setText(QKeySequence(event-
>key()).toString());
            p2rightkey = event->key();
        }
        if (mode == "p2up") {
            p2up->setText(QKeySequence(event-
>key()).toString());
            p2upkey = event->key();
        }
        input = false;
    }

}

void Settings::on_pushButton_3_clicked()
{
    input = true;
    mode = "p1left";
}


void Settings::on_pushButton_4_clicked()
{
    input = true;
    mode = "p1down";
}


void Settings::on_pushButton_5_clicked()
{
    input = true;
    mode = "p1right";
}


void Settings::on_pushButton_6_clicked()
{
    input = true;
    mode = "p1up";
}
```

```
void Settings::on_pushButton_7_clicked()
{
    input = true;
    mode = "p2left";
}


void Settings::on_pushButton_9_clicked()
{
    input = true;
    mode = "p2down";
}


void Settings::on_pushButton_10_clicked()
{
    input = true;
    mode = "p2right";
}


void Settings::on_pushButton_8_clicked()
{
    input = true;
    mode = "p2up";
}


void Settings::on_backButton_clicked()
{
    QSound::play(":/music/close.wav");
    Widget* widget = new Widget();
    widget->show();
    this->close();
}


void Settings::on_saveButton_clicked()
{
    QSound::play(":/music/ok.wav");
    QString Path = QCoreApplication::applicationDirPath();
    QString endPath = Path + "/config/config.ini";
    config = new QSettings(endPath, QSettings::IniFormat);
    config->setValue("lives", slider->value());
    config->setValue("p1left", p1leftkey);
    config->setValue("p1down", p1downkey);
    config->setValue("p1right", p1rightkey);
    config->setValue("p1up", p1upkey);
    config->setValue("p2left", p2leftkey);
    config->setValue("p2down", p2downkey);
    config->setValue("p2right", p2rightkey);
    config->setValue("p2up", p2upkey);
```

```
}

void Settings::on_horizontalSlider_valueChanged(int value)
{
    sliderView->setNum(value);
}

// records.h

#ifndef RECORDS_H
#define RECORDS_H

#include <QWidget>
#include <QTableWidget>
#include <QFile>
#include <QTextStream>
#include <QStandardItemModel>
#include <widget.h>
#include <map>

class Entry1 { // одна запись в таблице для 1 игрока
public:
    QString map; // названиe карты
    QString pl1name; // имя
    int difficulty; // сложность
    int lives; // кол-во жизней
    int pl1score; // очки 1 игррока
    float time; // время прохождения уровня
    Entry1(QStringList arr) {
        map = arr.at(0);
        difficulty = arr.at(1).toInt();
        lives = arr.at(2).toInt();
        pl1name = arr.at(3);
        pl1score = arr.at(4).toInt();
        time = arr.at(5).toFloat();
    }
    Entry1 () {pl1score = -1;};
    virtual void addTo(QTableWidget* table) { // добавление в
таблицу
        int count = table->rowCount();
        table->setRowCount(count+1);
        table->setItem(count, 0, new QTableWidgetItem(map));
        switch (difficulty) {
        case 1:
            table->setItem(count, 1, new
QTableWidgetItem("Easy"));
            break;
        case 2:
            table->setItem(count, 1, new
QTableWidgetItem("Medium"));
            break;
```

```
        case 3:
            table->setItem(count, 1, new
QTableWidgetItem("Hard"));
            break;
        default:
            break;
        }
        QTableWidgetItem* item = new QTableWidgetItem();
        item->setData(Qt::DisplayRole, lives);
        table->setItem(count, 2, item);
        item = new QTableWidgetItem();
        table->setItem(count, 3, new QTableWidgetItem(pl1name));
        item->setData(Qt::DisplayRole, pl1score);
        table->setItem(count, 4, item);
        item = new QTableWidgetItem();
        item->setData(Qt::DisplayRole, time);
        table->setItem(count, 5, item);
    }
};

class Entry2: public Entry1 { // одна запись в таблице для 2
игроков
public:
    QString pl2name; // имя 2 игрока
    int pl2score; // очки 2 игрока
    Entry2 (QStringList arr) {
        map = arr.at(0);
        difficulty = arr.at(1).toInt();
        lives = arr.at(2).toInt();
        pl1name = arr.at(3);
        pl2name = arr.at(4);
        pl1score = arr.at(5).toInt();
        pl2score = arr.at(6).toInt();
        time = arr.at(7).toFloat();
    }

    Entry2 () {pl1score = -1;};

    void addTo(QTableWidget* table) { // добавление в таблицу
        int count = table->rowCount();
        table->setRowCount(count+1);
        table->setItem(count, 0, new QTableWidgetItem(map));
        switch (difficulty) {
        case 1:
            table->setItem(count, 1, new
QTableWidgetItem("Easy"));
            break;
        case 2:
            table->setItem(count, 1, new
QTableWidgetItem("Medium"));
            break;
        case 3:
```

```
            table->setItem(count, 1, new
QTableWidgetItem("Hard"));
            break;
        default:
            break;
        }
        QTableWidgetItem* item = new QTableWidgetItem();
        item->setData(Qt::DisplayRole, lives);
        table->setItem(count, 2, item);
        table->setItem(count, 3, new QTableWidgetItem(pl1name));
        table->setItem(count, 4, new QTableWidgetItem(pl2name));
        item = new QTableWidgetItem();
        item->setData(Qt::DisplayRole, pl1score);
        table->setItem(count, 5, item);
        item = new QTableWidgetItem();
        item->setData(Qt::DisplayRole, pl2score);
        table->setItem(count, 6, item);
        item = new QTableWidgetItem();
        item->setData(Qt::DisplayRole, time);
        table->setItem(count, 7, item);
    }
};

namespace Ui {
class Records;
}

class Records : public QWidget // окно записей рекордов игроков
{
    Q_OBJECT

public:
    explicit Records(QWidget *parent = nullptr);
    void fill1player(); // заполнение таблицы для 1 игрока
    void fill2players(); // заполнение таблицы для 2 игроков
    void load(); // загрузка из внешнего файла
    std::map<QString,Entry1> map1pl; // map для хранения
результатов 1 игрока
    std::map<QString,Entry2> map2pl; // map для хранения
результатов 2 игроков
    QStandardItemModel* model;
    QTableWidget* table;
    ~Records();

private slots:
    void on_oneButton_clicked();

    void on_twoButton_clicked();

    void on_backButton_clicked();

private:
```

```
    Ui::Records *ui;
};

#endif

// records.cpp

#include "records.h"
#include "ui_records.h"
#include <QDebug>
#include <QSound>

Records::Records(QWidget *parent) :
    QWidget(parent),
    ui(new Ui::Records)
{
    ui->setupUi(this);
    QWidget::showFullScreen();
    table = ui->tableWidget;
    table->setStyleSheet("QTableCornerButton::section {
background-color: rgb(35, 37, 40); }QTableWidget {color:
rgb(216, 216, 216); gridline-color:
#fffff8;}QHeaderView::section {color: rgb(216, 216,
216);background-color: rgb(35, 37, 40);gridline-color:
#fffff8;padding: 4px;font-size: 14pt;}");
    model = new QStandardItemModel();
    load();
    fill1player();
}

Records::~Records()
{
    delete ui;
}

void Records::load() {
    QString endPath = QCoreApplication::applicationDirPath() +
"/records/records1player.txt";
    QFile file(endPath);
    try {
    file.open(QIODevice::ReadOnly | QIODevice::Text);
    QTextStream in(&file);
    while (!in.atEnd()) {
        QString data = in.readLine();
        QStringList splited = data.split('|');
        if (splited.size() < 6) {continue;};
        Entry1 entry1 = Entry1(splited);
        auto id = entry1.pl1name + entry1.difficulty +
entry1.lives + entry1.map;
        int count = map1pl.count(id);
```

```
        if (count == 0 || (count > 0 &&
map1pl.at(id).pl1score*(1/map1pl.at(id).time) <
entry1.pl1score*(1/entry1.time))) {
            map1pl[id] = entry1;
        }
    }
    file.close();


    endPath = QCoreApplication::applicationDirPath() +
"/records/records2players.txt";
    QFile file2(endPath);
    file2.open(QIODevice::ReadOnly | QIODevice::Text);
    QTextStream in2(&file2);
    while (!in2.atEnd()) {
        QString data = in2.readLine();
        QStringList splited = data.split('|');
        if (splited.size() < 8) {continue;};
        Entry2 entry2 = Entry2(splited);
        auto id = entry2.pl1name + entry2.pl2name +
entry2.difficulty + entry2.lives + entry2.map;
        int count = map1pl.count(id);
        int score = entry2.pl2score + entry2.pl1score;
        if (count == 0 || (count > 0 && (map2pl.at(id).pl1score
+ map2pl.at(id).pl2score)*(1/map2pl.at(id).time) <
score*(1/entry2.time))) {
            map2pl[id] = entry2;
        }
    }
    file2.close();
    } catch (std::exception const&e) {
        QMessageBox msgBox;
        msgBox.setText(e.what());
        msgBox.setIcon(QMessageBox::Warning);
        msgBox.setDefaultButton(QMessageBox::Ok);
        msgBox.exec();
        Widget* widget = new Widget();
        widget->show();
        this->close();
    }

}

void Records::fill1player() {
    table->clear();
    table->setSortingEnabled(false);
    table->setRowCount(0);
    table->setColumnCount(6);
    table->setHorizontalHeaderItem(0, new
QTableWidgetItem("Map"));
    table->setHorizontalHeaderItem(1, new
QTableWidgetItem("Difficulty"));
```

```
    table->setHorizontalHeaderItem(2, new
QTableWidgetItem("Lives"));
    table->setHorizontalHeaderItem(3, new
QTableWidgetItem("Name"));
    table->setHorizontalHeaderItem(4, new
QTableWidgetItem("Score"));
    table->setHorizontalHeaderItem(5, new
QTableWidgetItem("Time"));
    for (std::pair<QString, Entry1> pair : map1pl) {
        Entry1 entry = pair.second;
        entry.addTo(table);
    }
    table->setSortingEnabled(true);
}

void Records::fill2players() {
    table->clear();
    table->setSortingEnabled(false);
    table->setRowCount(0);
    table->setColumnCount(8);
    table->setHorizontalHeaderItem(0, new
QTableWidgetItem("Map"));
    table->setHorizontalHeaderItem(1, new
QTableWidgetItem("Difficulty"));
    table->setHorizontalHeaderItem(2, new
QTableWidgetItem("Lives"));
    table->setHorizontalHeaderItem(3, new QTableWidgetItem("1 Pl
Name"));
    table->setHorizontalHeaderItem(4, new QTableWidgetItem("2 Pl
Name"));
    table->setHorizontalHeaderItem(5, new QTableWidgetItem("1 Pl
Score"));
    table->setHorizontalHeaderItem(6, new QTableWidgetItem("2 Pl
Score"));
    table->setHorizontalHeaderItem(7, new
QTableWidgetItem("Time"));
    for (std::pair<QString, Entry2> pair : map2pl) {
        Entry2 entry = pair.second;
        entry.addTo(table);
    }
    table->setSortingEnabled(true);
}

void Records::on_oneButton_clicked()
{
    QSound::play(":/music/ok.wav");
    fill1player();
}

void Records::on_twoButton_clicked()
{
    QSound::play(":/music/ok.wav");
```

```
    fill2players();
}

void Records::on_backButton_clicked()
{
    QSound::play(":/music/close.wav");
    Widget* widget = new Widget();
    widget->show();
    this->close();
}

//movable.h

#ifndef MOVABLE_H
#define MOVABLE_H

#include <direction.h>
#include <cstdlib>

class Movable // класс для движущегося объекта
{
protected:
    int x; // x координата
    int y; // y координата
    int memAnim; // сохранение анимации
    Direction direction; // направление движения

public:
    float speed; // скорость
    float movePhase; // степень сдвига
    void move(int h, int w) { // движение
        this->x += direction.horizontal;
        this->y += direction.vertical;

        if (this->x <= 0) {
            this->x = w-1;
        } else if (this->x >= w-1) {
            this->x = 0;
        }

        if (this->y <= 0) {
            this->y = h-1;
        } else if (this->y >= h-1) {
            this->y = 0;
        }
    };
    int getX() {return x;};
    int getY() {return y;};
    void setDir(int h, int v) {
        this->direction.horizontal = h;
        this->direction.vertical = v;
    };
```

```
    void setDir(Direction dir) {
        this->direction = dir;
    };
    int getH() {return this->direction.horizontal;};
    int getV() {return this->direction.vertical;};
    Direction getDir() {return this->direction;};
    int getAnimDir() {
        if (direction.horizontal == -1) {
            memAnim = 2;
            return 2;
        }
        else if (direction.horizontal == 1) {
            memAnim = 0;
            return 0;
        }
        else if (direction.vertical == -1) {
            memAnim = 3;
            return 3;
        }
        else if (direction.vertical == 1) {
            memAnim = 1;
            return 1;
        }
        else { return memAnim; };
    }
    Movable() {direction.horizontal = 0; direction.vertical = 0;
memAnim = 0;};
    Movable(int x, int y)  {
        this->x = x;
        this->y = y;
        direction.horizontal = 0;
        direction.vertical = 0;
        memAnim = 0;
    };
};

class Player: public Movable { // игрок
private:
    Direction memoryDirection; // запоминание направления
движения
    int spawnX; // изначальные координаты появления иргока
    int spawnY;
public:
    int lives; // жизни
    bool targetable; // может ли враг навредить игроку
    void setMDir(int h, int v) {
        this->memoryDirection.horizontal = h;
        this->memoryDirection.vertical = v;
    };
    void setMDir(Direction dir) {
        this->memoryDirection = dir;
    };
```

```
    int getMH() {return this->memoryDirection.horizontal;};
    int getMV() {return this->memoryDirection.vertical;};
    Direction getMDir() {return this->memoryDirection;};
    void toSpawn() { // перемещение в изначальную позицию
        targetable = false;
        x = spawnX;
        y = spawnY;
        setDir(0, 0);
        lives--;
        movePhase = 0;
    }

    Player() {
        direction.horizontal = 0;
        direction.vertical = 0;
        memoryDirection.horizontal = 0;
        memoryDirection.vertical = 0;
        x = 0;
        y = 0;
        movePhase = 0;
        speed = 1.0;
        lives = 3;
        spawnX = x;
        spawnY = y;
        targetable = true;
    }

    Player(int lives): Player() {
        this->lives = lives;
    }

    Player(int lives, int x, int y): Player(lives) {
        this->x = x;
        this->y = y;
        spawnX = x;
        spawnY = y;
    }

    Player(int lives, int x, int y, int h, int v): Player(lives,
x, y) {
        this->setDir(h, v);
    }

    Player(int lives, int x, int y, Direction dir):
Player(lives, x, y) {
        this->setDir(dir);
    }

};

class Enemy: public Movable { // враг
public:
```

```
    int color; // цвет
    Enemy() {
        this->color = rand()%4;
        this->movePhase = 0;
        this->speed = 0.8;
        direction.horizontal = 0;
        direction.vertical = 0;
    };
    Enemy(int x, int y, float spd): Enemy() {
        this->x = x;
        this->y = y;
        this->speed = spd;
    };

};

#endif

//level.h

#ifndef LEVEL_H
#define LEVEL_H

#include <QString>
#include <QDebug>
#include <QFile>
#include <movable.h>
#include <queue>
#include <QMessageBox>

class LevelLoadException : public std::exception { // ошибка
загрузки уровня
    std::string msg; // сообщение
public:
    const char * what() const noexcept override { // вывод
сообщения
        return msg.c_str();
    }
    LevelLoadException(std::string msg) {this->msg = msg;};
};

struct navCell { // ячейки которыми запоолняется таблица для
поиска пути для врагов
    Direction dir; // направление движения
    int length; // расстояние до игрока
};

struct coord { // координаты
    int x;
    int y;
    coord(int xx, int yy) {
        x = xx;
```

```
        y = yy;
    };
};

class Level // уровень игры
{
private:
    int h; // высота
    int w; // ширина
    int** map; // карта

public:
    Player p1; // 1 игрок
    Player p2; // 2 игрок
    QString p1name; // имена игроков
    QString p2name;
    QString mapName;
    Enemy* enemies; // массив из врагов
    int difficulty; // сложность
    int enemiesCount; // кол-во врагов на уровне
    int coinsCount; // кол-во собранных монеток
    bool p2enabled; // активен ли 2 игрок
    int p1Score; // очки игроков
    int p2Score;
    int score; // общие очки
    int maxLives; // максимальное кол-во жизней
    int getWidth() {return w;};
    int getHeight() {return h;};
    navCell** navMap; // массив - таблица для поиска пути от
врагов к игрокам
    int** getMap() {return map;};
    navCell** getNavMap() {return navMap;};
    void recreateNavMap() { // пересоздать/обновить таблицу
навигации
        this->restoreNavMap();
        std::queue <coord> q;
        if (p1.targetable) {
            navMap[p1.getX()][p1.getY()].length = 0;
            q.push(coord{p1.getX(), p1.getY()});
        }

        if (p2enabled && p2.targetable) {
            navMap[p2.getX()][p2.getY()].length = 0;
            q.push(coord{p2.getX(), p2.getY()});
        }

        while (!q.empty()) {
            coord cell = q.front();
            q.pop();
            if (cell.x > 0 && navMap[cell.x-1][cell.y].length ==
-1) {
                q.push(coord{cell.x-1, cell.y});
```

```
                navMap[cell.x-1][cell.y].dir = Direction(1, 0);
                navMap[cell.x-1][cell.y].length =
navMap[cell.x][cell.y].length+1;
            }
            if (cell.x < w-1 && navMap[cell.x+1][cell.y].length
== -1) {
                q.push(coord{cell.x+1, cell.y});
                navMap[cell.x+1][cell.y].dir = Direction(-1, 0);
                navMap[cell.x+1][cell.y].length =
navMap[cell.x][cell.y].length+1;
            }
            if (cell.y > 0 && navMap[cell.x][cell.y-1].length ==
-1) {
                q.push(coord(cell.x, cell.y-1));
                navMap[cell.x][cell.y-1].dir = Direction(0, 1);
                navMap[cell.x][cell.y-1].length =
navMap[cell.x][cell.y].length+1;
            }
            if (cell.y < h-1 && navMap[cell.x][cell.y+1].length
== -1) {
                q.push(coord(cell.x, cell.y+1));
                navMap[cell.x][cell.y+1].dir = Direction(0, -1);
                navMap[cell.x][cell.y+1].length =
navMap[cell.x][cell.y].length+1;
            }
        }


    }

    void restoreNavMap() { // очищение таблицы
        for (int i = 0; i < w; i++) {
            for (int j = 0; j < h; j++) {
                if (navMap[i][j].length > -1) {
                    navMap[i][j] = navCell{Direction(0, 0), -1};
                }
            }
        }
    }

    Level(QString filename, bool mode, QString p1, QString p2,
int difficulty, int lives) {
        this->maxLives = lives;
        this->difficulty = difficulty;
        this->coinsCount = 0;
        this->p2enabled = mode;
        this->p1name = p1;
        this->p2name = p2;
        this->mapName =
filename.mid(filename.lastIndexOf('/')+1,
filename.lastIndexOf('.')-filename.lastIndexOf('/')-1);
        QFile file(filename);
```

```
        file.open(QIODevice::ReadOnly);
        QString data;
        data = file.readAll();
        QStringList splited = data.split('|');
        QRegExp re("\\D*");
        if (splited.size() < 8 ||
            re.exactMatch(splited.at(0))||
            re.exactMatch(splited.at(1))||
            re.exactMatch(splited.at(2))||
            re.exactMatch(splited.at(3))||
            re.exactMatch(splited.at(4))) {
            throw LevelLoadException("Level file corrupted");
        }
        this->w = QString(splited.at(0)).toInt();
        this->h = QString(splited.at(1)).toInt();
        switch (difficulty) {
        case 1:
            this->enemiesCount = QString(splited.at(2)).toInt();
            break;
        case 2:
            this->enemiesCount = QString(splited.at(3)).toInt();
            break;
        case 3:
            this->enemiesCount = QString(splited.at(4)).toInt();
            break;
        default:
            this->enemiesCount = QString(splited.at(2)).toInt();
            this->difficulty = 1;
            break;
        }
        this->enemies = new Enemy[enemiesCount];
        QString lvlMap = splited.at(4+difficulty);
        lvlMap.remove('\r');
        lvlMap.remove('\n');
        map = new int*[w];
        navMap = new navCell*[w];
        int x = 0;
        bool pl1created = false;
        bool pl2created = false;
        for (int i = 0; i < this->w; i++) {
            this->map[i] = new int[h];
            this->navMap[i] = new navCell[h];
            for (int j = 0; j < this->h; j++) {
                if (i+j*w > lvlMap.length()) {
                    throw LevelLoadException("Level file
corrupted");
                }
                if (!lvlMap.at(i+j*w).isNumber()) {
                    throw LevelLoadException("Level file
contains illegal characters");
                }
```

```
                this->map[i][j] =
QString(lvlMap.at(i+j*w)).toInt();

                if (this->map[i][j] == 4) {
                    this->p1 = Player(lives, i, j);
                    this->map[i][j] = 0;
                    pl1created = true;
                } else if (this->map[i][j] == 5 && x <
enemiesCount) {
                    this->enemies[x] = Enemy(i, j,
0.3+(0.2*difficulty));
                    this->map[i][j] = 0;
                    x++;
                } else if (this->map[i][j] == 6 && p2enabled) {
                    this->p2 = Player(lives, i, j);
                    this->map[i][j] = 0;
                    pl2created = true;
                } else if (this->map[i][j] == 6 && !p2enabled){
                    this->map[i][j] = 0;
                    pl2created = true;
                } else if (this->map[i][j] == 2 || this-
>map[i][j] == 3) {
                    this->coinsCount++;
                } else if (this->map[i][j] == 1) {
                    navMap[i][j] = navCell{Direction(0, 0), -2};
                } else if (this->map[i][j] == 0) {
                    navMap[i][j] = navCell{Direction(0, 0), -1};
                } else {
                    throw LevelLoadException("Level file
contains illegal characters");
                }
            }
        }
        score = 0;
        p1Score = 0;
        p2Score = 0;

        if (!pl1created || !pl2created) {
            throw LevelLoadException("Level file doesn't
contains player position");
        }
    }

    Level() {h = 0; w = 0;};
};

#endif

// direction.h
#ifndef DIRECTION_H
#define DIRECTION_H
```

```
class Direction { // класс показывающий направление
public:
    short horizontal; // горизонтальное напр.
    short vertical; // вертикальное напр.
    Direction(int h, int v) {
        this->horizontal = h;
        this->vertical = v;
    }
    Direction() {
        this->horizontal = 0;
        this->vertical = 0;
    }
};

#endif

// help.h

#ifndef HELP_H
#define HELP_H

#include <QWidget>
#include <QTextBrowser>
#include <QSound>
#include <widget.h>

namespace Ui {
class Help;
}

class Help : public QWidget // окно помощи с информацией об
управлении
{
    Q_OBJECT

public:
    explicit Help(QWidget *parent = nullptr);
    bool english = false; // язык
    QTextBrowser* textBrowser;
    // тексты для вывода на двух языках
    QString rusHtml = "<!DOCTYPE HTML PUBLIC>”
    QString engHtml = "<!DOCTYPE HTML PUBLIC>”;
    ~Help();

private slots:
    void on_backButton_clicked();

    void on_rusButton_clicked();

    void on_engButton_clicked();

private:
```

```
    Ui::Help *ui;
};

#endif

// help.cpp

#include "help.h"
#include "ui_help.h"

Help::Help(QWidget *parent) :
    QWidget(parent),
    ui(new Ui::Help)
{
    ui->setupUi(this);
    QWidget::showFullScreen();
    textBrowser = ui->textBrowser;
}

Help::~Help()
{
    delete ui;
}

void Help::on_rusButton_clicked()
{
    QSound::play(":/music/ok.wav");
    textBrowser->setText(rusHtml);
}

void Help::on_engButton_clicked()
{
    QSound::play(":/music/ok.wav");
    textBrowser->setText(engHtml);
}

void Help::on_backButton_clicked()
{
    QSound::play(":/music/close.wav");
    Widget* widget = new Widget();
    widget->show();
    this->close();
}

// game.h

#ifndef GAME_H
#define GAME_H

#include <QWidget>
#include <QPainter>
#include <QPixmap>
```

```
#include <QTimer>
#include <QFileDialog>
#include <QDebug>
#include <QKeyEvent>
#include <QSettings>
#include <QVBoxLayout>
#include <QInputDialog>
#include <QElapsedTimer>
#include <QLabel>
#include <cmath>

#include <level.h>
#include <widget.h>
#include <settingsdialog.h>
#include <direction.h>

namespace Ui {
class Game;
}

class Game : public QWidget // окно игры
{
    Q_OBJECT

public:
    explicit Game(QWidget *parent = nullptr);
    void paintEvent(QPaintEvent *event) override;
    void endGame(); // завершение игры
    QPixmap *spriteMap; // графические ресурсы игры
    const int spriteSize = 16; // размер 1 клетки
    int frame = 0; // текущий кадр. необходим для анимации
    int animationSpeed = 25; // скорость анимации
    QTimer *animationTimer; // таймер анимации
    Level level; // объект уровня
    bool error = false; // выявлена ошибка
    int lives; // кол-во жизней
    int difficulty; // сложность
    int p1leftkey; // клавиши управления
    int p1downkey;
    int p1rightkey;
    int p1upkey;
    int p2leftkey;
    int p2downkey;
    int p2rightkey;
    int p2upkey;
    int playerSpeed = 1; // скорость игрока
    int bonusTime = 3000; // время ускорения при поднятии бонуса
    int untargetTime = 2000; // время невосприимчивости к урону
после его получаения
    int viewSize = 3; // масштабирование камеры
    int** map; // карта
```

```
    bool loaded = false; // все ресурсы загружены и всё готово к
игре
    QString pl1name; // имена игроков
    QString pl2name;
    QTimer *timer2; // таймеры для ускорения после бонуса
    QTimer *timer1;
    QTimer *untargetTimer1; // таймеры невосприимчивости к урону
    QTimer *untargetTimer2;
    QElapsedTimer gameTimer; // время прохождения уровня
    //QVBoxLayout *layout;
    QFrame* endScreen; // экран завершения игры
    QLabel* title; // победа/порожение на экране завершения
    QLabel* stats; // статистика на экране завершения
    ~Game();

private slots:
    void nextFrame();
    void endBonusPl1(); // окончиние бонусов
    void endBonusPl2();
    void endUntargetPl1(); // окончание невосприимчивости
    void endUntargetPl2();

    void on_recordsButton_clicked();

    void on_againButton_clicked();

    void on_menuButton_clicked();

private:
    void keyPressEvent(QKeyEvent *event) override;
    Ui::Game *ui;
};

#endif

// game.cpp

#include "game.h"
#include "ui_game.h"
#include <QSound>

Game::Game(QWidget *parent) :
    QWidget(parent),
    ui(new Ui::Game)
{
    ui->setupUi(this);
    QWidget::showFullScreen();
    QWidget::setAttribute( Qt::WA_DeleteOnClose, true );
    endScreen = ui->frame_5;
    endScreen->setVisible(false);
    title = ui->label;
    stats = ui->label_2;
```

```
    bool mode = true;
    try {
    SettingsDialog dialog(this);
    QString filename = QFileDialog::getOpenFileName();
    if (dialog.exec() == QDialog::Rejected || filename == "") {
        error = true;
    } else {
        dialog.getSettings(mode, difficulty, pl1name, pl2name);
        spriteMap = new QPixmap(":/images/sprites.png");

        QString Path = QCoreApplication::applicationDirPath();
        QString endPath = Path + "/config/config.ini";
        QFileInfo fileinfo(endPath);
        if (fileinfo.isFile()) {
            QSettings* config = new QSettings(endPath,
QSettings::IniFormat);
            lives = config->value("lives", "").toInt();
            p1leftkey = config->value("p1left", "").toInt();
            p1downkey = config->value("p1down", "").toInt();
            p1rightkey = config->value("p1right", "").toInt();
            p1upkey = config->value("p1up", "").toInt();
            p2leftkey = config->value("p2left", "").toInt();
            p2downkey = config->value("p2down", "").toInt();
            p2rightkey = config->value("p2right", "").toInt();
            p2upkey = config->value("p2up", "").toInt();
        }
        level = Level(filename, mode, pl1name, pl2name,
difficulty, lives);


        level.recreateNavMap();
        for (int x = 0; x < level.enemiesCount; x++) {

level.enemies[x].setDir(level.getNavMap()[level.enemies[x].getX(
)][level.enemies[x].getY()].dir);
        }
        animationTimer = new QTimer();
        connect(animationTimer, SIGNAL(timeout()), this,
SLOT(nextFrame()));

        timer1 = new QTimer(this);
        timer2 = new QTimer(this);
        untargetTimer1 = new QTimer(this);
        untargetTimer2 = new QTimer(this);
        animationTimer->start(animationSpeed);

        gameTimer.start();
        loaded = true;
    }
    } catch(std::exception const&e) {
        QMessageBox msgBox;
        msgBox.setText(e.what());
```

```
        msgBox.setIcon(QMessageBox::Warning);
        msgBox.setDefaultButton(QMessageBox::Ok);
        msgBox.exec();
        error = true;
    };

}

Game::~Game()
{
    delete ui;
}

void Game::paintEvent(QPaintEvent *event) {
    if (error) {
        Widget* widget = new Widget();
        widget->show();
        this->close();
    };
    if (loaded) {
        QPainter canv(this);
        canv.setViewport(0, 0, canv.viewport().width()*viewSize,
canv.viewport().height()*viewSize);
        canv.fillRect(QRect(0, 0, canv.device()->width(),
canv.device()->height()), QColor(44, 47, 51));

        map = level.getMap();
        for (int i = 0; i < level.getWidth(); i++) {
            for (int j = 0; j < level.getHeight(); j++) {
                if (map[i][j] == 1) {
                    canv.drawPixmap(spriteSize*i, spriteSize*j,
*spriteMap, 6*spriteSize, 1*spriteSize, spriteSize, spriteSize);
                } else if (map[i][j] == 2) {
                    canv.drawPixmap(spriteSize*i, spriteSize*j,
*spriteMap, 6*spriteSize, 0, spriteSize, spriteSize);
                } else if (map[i][j] == 3) {
                    canv.drawPixmap(spriteSize*i, spriteSize*j,
*spriteMap, 7*spriteSize, 0, spriteSize, spriteSize);
                }

            }
        }
        if (!level.p1.targetable) {
            canv.setOpacity(0.5);
        }

canv.drawPixmap(spriteSize*level.p1.getX()+level.p1.movePhase*le
vel.p1.getH(),
spriteSize*level.p1.getY()+level.p1.movePhase*level.p1.getV(),
*spriteMap, (frame/10)*spriteSize,
level.p1.getAnimDir()*spriteSize, spriteSize, spriteSize);
        canv.setOpacity(1);
```

```
        if (!level.p2.targetable) {
            canv.setOpacity(0.5);
        }
        if (level.p2enabled) {

canv.drawPixmap(spriteSize*level.p2.getX()+level.p2.movePhase*le
vel.p2.getH(),
spriteSize*level.p2.getY()+level.p2.movePhase*level.p2.getV(),
*spriteMap, (3+frame/10)*spriteSize,
level.p2.getAnimDir()*spriteSize, spriteSize, spriteSize);
        }
        canv.setOpacity(1);

        for (int x = 0; x < level.enemiesCount; x++) {
            Enemy enemy = level.enemies[x];

canv.drawPixmap(spriteSize*enemy.getX()+enemy.movePhase*enemy.ge
tH(), spriteSize*enemy.getY()+enemy.movePhase*enemy.getV(),
*spriteMap, (enemy.color*2+frame/15)*spriteSize,
(4+enemy.getAnimDir())*spriteSize, spriteSize, spriteSize);
        }

        canv.setPen(Qt::white);
        canv.setFont(QFont("Press Start 2P", 5));
        canv.drawText(10, 10, QString("Pl1 [" + level.p1name +
"]: ") + QString::number(level.p1Score) + " lives: " +
QString::number(level.p1.lives));
        canv.drawText(canv.device()->width()/2/viewSize-15, 10,
QString::number(gameTimer.elapsed()/1000.0));
        if (level.p2enabled) {
            canv.drawText(canv.device()->width()/2/viewSize, 20,
QString::number(level.score));
            canv.drawText(canv.device()->width()/viewSize-230,
10, QString("Pl2 [" + level.p2name + "]: ") +
QString::number(level.p2Score) + " lives: " +
QString::number(level.p2.lives));
        }
        canv.end();
    }
}

void Game::endGame() {
    if (!QDir(QCoreApplication::applicationDirPath() +
"/records").exists()) {
        QDir().mkdir(QCoreApplication::applicationDirPath() +
"/records");
    }

    if (level.p2enabled) {
        if (level.p1.lives > 0 && level.p2.lives > 0) {
            title->setText("WIN");
            QSound::play(":/music/win.wav");
```

```
        } else {
            title->setText("LOSE");
            QSound::play(":/music/lose.wav");
        }
        stats->setText("Pl1 Score: " +
QString::number(level.p1Score) + " Pl2 Score: " +
QString::number(level.p2Score) + " Time: " +
QString::number(gameTimer.elapsed()/1000.0));

        QString endPath = QCoreApplication::applicationDirPath()
+ "/records/records2players.txt";
        QFile file(endPath);
        file.open(QIODevice::ReadWrite | QIODevice::Append);
        QTextStream out(&file);
        out << QString(level.mapName + "|" +
QString::number(difficulty) + "|" +
QString::number(level.maxLives) + "|" + level.p1name + "|" +
level.p2name + "|" + QString::number(level.p1Score) + "|" +
QString::number(level.p2Score) + "|" +
QString::number(gameTimer.elapsed()/1000.0) + '\n');
        file.close();
    } else {
        if (level.p1.lives > 0) {
            title->setText("WIN");
            QSound::play(":/music/win.wav");
        } else {
            title->setText("LOSE");
            QSound::play(":/music/lose.wav");
        }
        stats->setText("Score: " +
QString::number(level.p1Score) + " Time: " +
QString::number(gameTimer.elapsed()/1000.0));

        QString endPath = QCoreApplication::applicationDirPath()
+ "/records/records1player.txt";
        QFile file(endPath);
        file.open(QIODevice::ReadWrite | QIODevice::Append);
        QTextStream out(&file);
        out << QString(level.mapName + "|" +
QString::number(difficulty) + "|" +
QString::number(level.maxLives) + "|" + level.p1name + "|" +
QString::number(level.p1Score) + "|" +
QString::number(gameTimer.elapsed()/1000.0) + '\n');
        file.close();
    }
    this->setStyleSheet("background-color: black;");
    endScreen->setVisible(true);
}

void Game::nextFrame() {
    frame++;
    for (int x = 0; x < level.enemiesCount; x++) {
```

```
        if (level.enemies[x].getH() != 0 ||
level.enemies[x].getV() != 0) {
            level.enemies[x].movePhase +=
level.enemies[x].speed;
            if (level.enemies[x].movePhase >= spriteSize/2) {
                level.enemies[x].movePhase = -
level.enemies[x].movePhase;
                level.enemies[x].move(level.getHeight(),
level.getWidth());
            }
        }
        if (abs(level.enemies[x].movePhase) <= 0.1) {
            level.enemies[x].movePhase = 0;

level.enemies[x].setDir(level.getNavMap()[level.enemies[x].getX(
)][level.enemies[x].getY()].dir);
        }


    }
    if (level.p1.getH() != 0 || level.p1.getV() != 0) {
        level.p1.movePhase += level.p1.speed;
        if (level.p1.movePhase >= spriteSize/2) {
            level.p1.movePhase = -level.p1.movePhase;
            level.p1.move(level.getHeight(), level.getWidth());
            level.recreateNavMap();
        }
        if (level.p1.getX() > 0 && level.p1.getX() <
level.getWidth()-1 && level.p1.getY() > 0 && level.p1.getY() <
level.getHeight()-1 &&
map[level.p1.getX()+level.p1.getH()][level.p1.getY()+level.p1.ge
tV()] == 1 && level.p1.movePhase == 0) {
            level.p1.setDir(0, 0);
        }
        if
(map[level.p1.getX()+level.p1.getMH()][level.p1.getY()+level.p1.
getMV()] != 1 && level.p1.movePhase == 0 && (level.p1.getMH() !=
0 || level.p1.getMV() != 0)) {
            level.p1.setDir(level.p1.getMH(), level.p1.getMV());
        }
        if (map[level.p1.getX()][level.p1.getY()] == 2) {
            level.getMap()[level.p1.getX()][level.p1.getY()] =
0;
            level.p1Score += 10;
            level.score += 10;
            level.coinsCount--;

        }
        if (map[level.p1.getX()][level.p1.getY()] == 3) {
            level.getMap()[level.p1.getX()][level.p1.getY()] =
0;
            level.coinsCount--;
```

```
            level.p1Score += 30;
            level.score += 30;
            level.p1.speed = 2.0;
            if (timer1->isActive()) {
                timer1->stop();
            }
            QSound::play(":/music/bonus.wav");
            connect(timer1, SIGNAL(timeout()), this,
SLOT(endBonusPl1()));
            timer1->setSingleShot(true);
            timer1->setInterval(untargetTime);
            timer1->start();
        }
    }
    for (int i = 0; i < level.enemiesCount; i++) {
        if (level.p1.targetable && level.p1.getX() ==
level.enemies[i].getX() && level.p1.getY() ==
level.enemies[i].getY()) {
            QSound::play(":/music/hit.wav");
            level.p1.toSpawn();
            connect(untargetTimer1, SIGNAL(timeout()), this,
SLOT(endUntargetPl1()));
            untargetTimer1->setSingleShot(true);
            untargetTimer1->setInterval(untargetTime);
            untargetTimer1->start();
            level.recreateNavMap();
        }
        if (level.p2enabled && level.p2.targetable &&
level.p2.getX() == level.enemies[i].getX() && level.p2.getY() ==
level.enemies[i].getY()) {
            QSound::play(":/music/hit.wav");
            level.p2.toSpawn();
            connect(untargetTimer2, SIGNAL(timeout()), this,
SLOT(endUntargetPl2()));
            untargetTimer2->setSingleShot(true);
            untargetTimer2->setInterval(untargetTime);
            untargetTimer2->start();
            level.recreateNavMap();
        }
    }

    if (level.p2enabled && (level.p2.getH() != 0 ||
level.p2.getV() != 0)) {
        level.p2.movePhase += level.p2.speed;
        if (level.p2.movePhase >= spriteSize/2) {
            level.p2.movePhase = -level.p2.movePhase;
            level.p2.move(level.getHeight(), level.getWidth());
            level.recreateNavMap();
        }
        if
(map[level.p2.getX()+level.p2.getH()][level.p2.getY()+level.p2.g
etV()] == 1 && level.p2.movePhase == 0) {
```

```
            level.p2.setDir(0, 0);
        }
        if
(map[level.p2.getX()+level.p2.getMH()][level.p2.getY()+level.p2.
getMV()] != 1 && level.p2.movePhase == 0 && (level.p2.getMH() !=
0 || level.p2.getMV() != 0)) {
            level.p2.setDir(level.p2.getMH(), level.p2.getMV());
        }
        if (map[level.p2.getX()][level.p2.getY()] == 2) {
            level.getMap()[level.p2.getX()][level.p2.getY()] =
0;
            level.coinsCount--;
            level.p2Score += 10;
            level.score += 10;
        }
        if (map[level.p2.getX()][level.p2.getY()] == 3) {
            level.getMap()[level.p2.getX()][level.p2.getY()] =
0;
            level.coinsCount--;
            level.p2Score += 30;
            level.score += 30;
            level.p2.speed = 2.0;
            if (timer2->isActive()) {
                timer2->stop();
            }
            QSound::play(":/music/bonus.wav");
            connect(timer2, SIGNAL(timeout()), this,
SLOT(endBonusPl2()));
            timer2->setSingleShot(true);
            timer2->setInterval(bonusTime);
            timer2->start();
        }
    }

    if (loaded && (level.coinsCount <= 0 || level.p1.lives == 0
|| level.p2.lives == 0)) {
        loaded = false;
        animationTimer->stop();
        endGame();
    }
    frame = frame%30;
    repaint();
}

void Game::endBonusPl1() {
    level.p1.speed = 1.0;
}

void Game::endBonusPl2() {
    level.p2.speed = 1.0;
}
```

```
void Game::endUntargetPl1() {
    level.p1.targetable = true;
    level.recreateNavMap();
}

void Game::endUntargetPl2() {
    level.p2.targetable = true;
    level.recreateNavMap();
}

void Game::keyPressEvent(QKeyEvent *event) {
    int key = event->key();
    if (key == Qt::Key_Escape) {
        loaded = false;
        Widget* widget = new Widget();
        widget->show();
        this->close();
    }
    if (key == p1leftkey && (level.p1.getH() == 0
||level.p1.getH() == 1)) {
        if (level.p1.getX() > 0 && (map[level.p1.getX()-
1][level.p1.getY()] == 1 || (level.p1.movePhase != 0 &&
level.p1.getH() != 1))) {
            level.p1.setMDir(-1, 0);
        } else {
            level.p1.setDir(-1, 0);
            level.p1.setMDir(0, 0);
            level.p1.movePhase = -level.p1.movePhase;
        }
    }
    else if (key == p1downkey && (level.p1.getV() == 0
||level.p1.getV() == -1)) {
        if (level.p1.getY() < level.getHeight()-1 &&
(map[level.p1.getX()][level.p1.getY()+1] == 1 ||
(level.p1.movePhase != 0 && level.p1.getV() != -1))) {
            level.p1.setMDir(0, 1);
        } else {
            level.p1.setDir(0, 1);
            level.p1.setMDir(0, 0);
            level.p1.movePhase = -level.p1.movePhase;
        }
    }
    else if (key == p1rightkey && (level.p1.getH() == 0
||level.p1.getH() == -1)) {
        if (level.p1.getX() < level.getWidth()-1 &&
(map[level.p1.getX()+1][level.p1.getY()] == 1 ||
(level.p1.movePhase != 0 && level.p1.getH() != -1))) {
            level.p1.setMDir(1, 0);
        } else {
            level.p1.setDir(1, 0);
            level.p1.setMDir(0, 0);
            level.p1.movePhase = -level.p1.movePhase;
```

```
        }
    }
    else if (key == p1upkey && (level.p1.getV() == 0
||level.p1.getV() == 1)) {
        if (level.p1.getY() > 0 &&
(map[level.p1.getX()][level.p1.getY()-1] == 1 ||
(level.p1.movePhase != 0 && level.p1.getV() != 1))) {
            level.p1.setMDir(0, -1);
        } else {
            level.p1.setDir(0, -1);
            level.p1.setMDir(0, 0);
            level.p1.movePhase = -level.p1.movePhase;
        }
    }
    else if (key == p2leftkey && level.p2enabled &&
(level.p2.getH() == 0 ||level.p2.getH() == 1)) {
        if (level.p2.getX() > 0 && (map[level.p2.getX()-
1][level.p2.getY()] == 1 || (level.p2.movePhase != 0 &&
level.p2.getH() != 1))) {
            level.p2.setMDir(-1, 0);
        } else {
            level.p2.setDir(-1, 0);
            level.p2.setMDir(0, 0);
            level.p2.movePhase = -level.p2.movePhase;
        }
    }
    else if (key == p2downkey && level.p2enabled &&
(level.p2.getV() == 0 ||level.p2.getV() == -1)) {
        if (level.p2.getY() < level.getHeight()-1 &&
(map[level.p2.getX()][level.p2.getY()+1] == 1 ||
(level.p2.movePhase != 0 && level.p2.getV() != -1))) {
            level.p2.setMDir(0, 1);
        } else {
            level.p2.setDir(0, 1);
            level.p2.setMDir(0, 0);
            level.p2.movePhase = -level.p2.movePhase;
        }
    }
    else if (key == p2rightkey && level.p2enabled &&
(level.p2.getH() == 0 ||level.p2.getH() == -1)) {
        if (level.p2.getX() < level.getWidth()-1 &&
(map[level.p2.getX()+1][level.p2.getY()] == 1 ||
(level.p2.movePhase != 0 && level.p2.getH() != -1))) {
            level.p2.setMDir(1, 0);
        } else {
            level.p2.setDir(1, 0);
            level.p2.setMDir(0, 0);
            level.p2.movePhase = -level.p2.movePhase;
        }
    }
    else if (key == p2upkey && level.p2enabled &&
(level.p2.getV() == 0 ||level.p2.getV() == 1)) {
```

```
        if (level.p2.getY() > 0 &&
(map[level.p2.getX()][level.p2.getY()-1] == 1 ||
(level.p2.movePhase != 0 && level.p2.getV() != 1))) {
            level.p2.setMDir(0, -1);
        } else {
            level.p2.setDir(0, -1);
            level.p2.setMDir(0, 0);
            level.p2.movePhase = -level.p2.movePhase;
        }
    }
}

void Game::on_againButton_clicked() {
    QSound::play(":/music/ok.wav");
    Game* game = new Game();
    game->show();
    this->close();
}

void Game::on_recordsButton_clicked()
{
    QSound::play(":/music/ok.wav");
    Records* records = new Records();
    records->show();
    this->close();
}

void Game::on_menuButton_clicked()
{
    QSound::play(":/music/close.wav");
    Widget* widget = new Widget();
    widget->show();
    this->close();
}
```

ПРИЛОЖЕНИЕ Б

Внешний вид графического материала

На рисунке Б.1 представлена диаграмма классов.

На рисунке Б.2 представлена диаграмма компонентов.

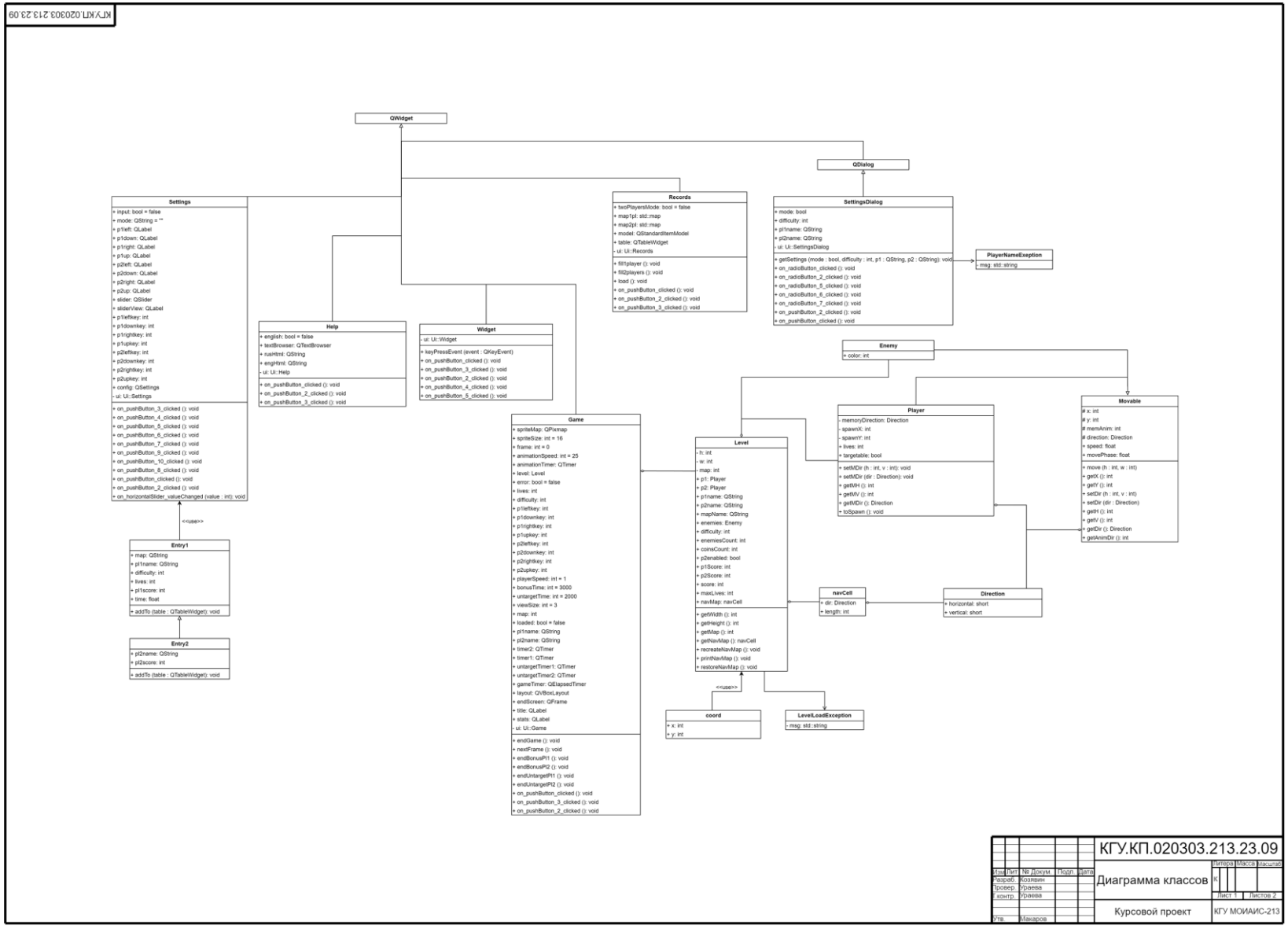

Рисунок Б.1 – Диаграмма классов

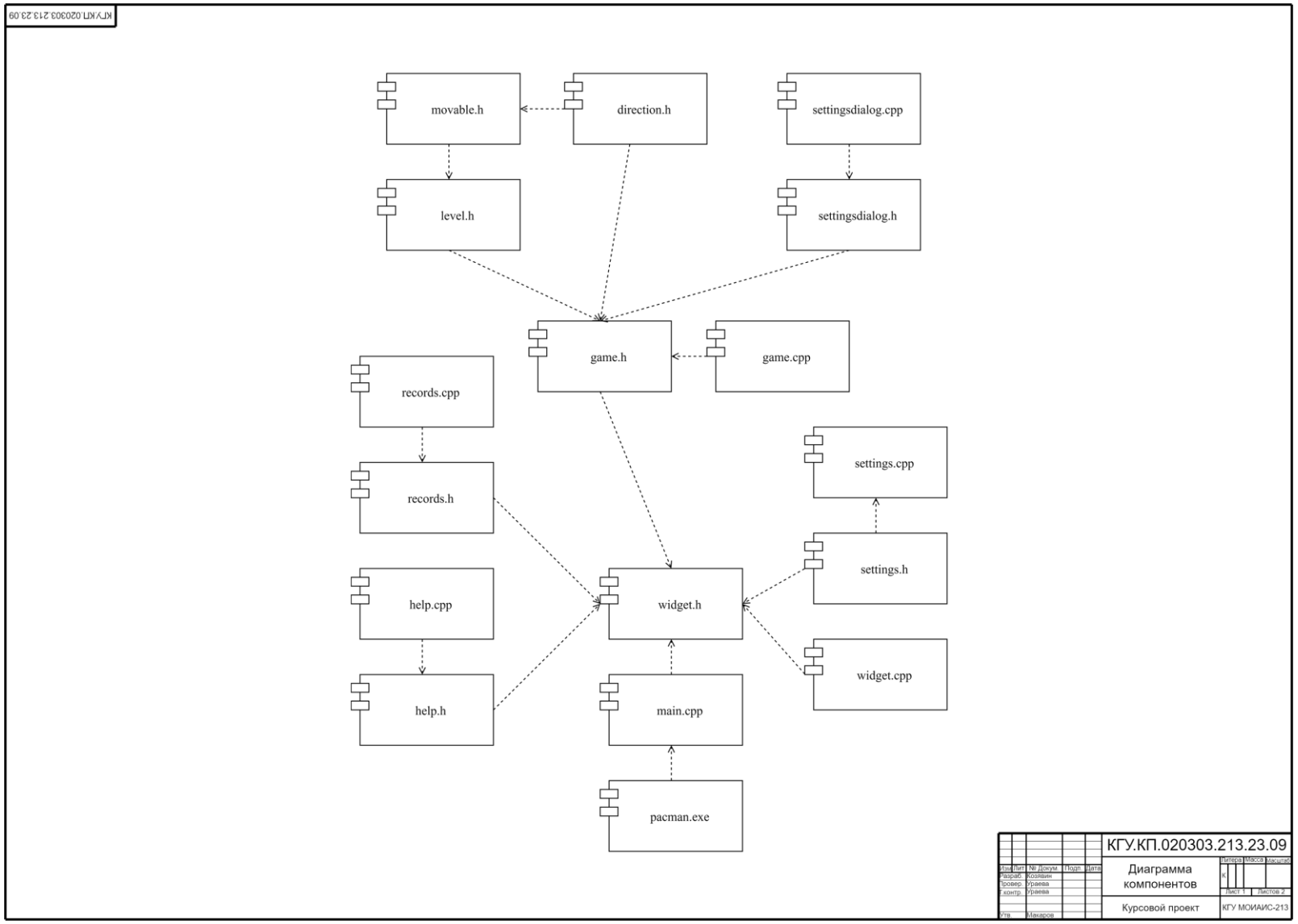

Рисунок Б.2 – Диаграмма компонентов