МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение высшего образования

«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ   
АЭРОКОСМИЧЕСКОГО ПРИБОРОСТРОЕНИЯ»

КАФЕДРА КОМПЬЮТЕРНЫХ ТЕХНОЛОГИЙ И ПРОГРАММНОЙ ИНЖЕНЕРИИ

КУРСОВОЙ ПРОЕКТ  
ЗАЩИЩЕН С ОЦЕНКОЙ

РУКОВОДИТЕЛЬ

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Ст. преподаватель |  |  |  | Е. О. Шумова |
| должность, уч. степень, звание |  | подпись, дата |  | инициалы, фамилия |

|  |
| --- |
| ПОЯСНИТЕЛЬНАЯ ЗАПИСКА К КУРСОВОМУ ПРОЕКТУ |
| Разработка приложения для организации взаимодействия объектов при заданных критериях |
| по дисциплине: ОБЪЕКТНО-ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ |
|  |
|  |

РАБОТУ ВЫПОЛНИЛА

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| СТУДЕНТКА ГР. | 4932 |  |  |  | Е.А. Цыганкова |
|  |  |  | подпись, дата |  | инициалы, фамилия |

Санкт-Петербург 2021

**Содержание**

1. Постановка задачи ……………………………………………………………3

1.1. Анализ предметной области ……………………………………...…3

1.2. Формулировка технического задания ………………………....…....4

2. Проектирование классов ……………………………………………………..6

2.1. Разработанные классы…..…………………………………………....6

2.2. Диаграмма классов ….………………………………………………..8

3. Разработка приложения ……………………………………………………...10

3.1. Разработка интерфейса приложения ……………………………….10

3.2. Разработка элементов графического интерфейса………………….16

3.3. Реализация классов ………………………………………………….20

3.3.1. Разработка класса загрузочного модуля…………………...21

3.3.2. Разработка класса меню……………………………………..21

3.3.3. Разработка класса для описания игры……………………...22

3.3.4. Разработка класса для описания управления в игре………22

3.3.5. Разработка класса уровней сложности……………………..23

3.3.6. Разработка класса музыки…………………………………..24

3.3.7. Разработка класса выигрышного окна……………………..24

3.3.8. Разработка класса проигрышного окна…………………….25

3.3.9. Разработка класса изображений…………………………….26

3.3.10. Разработка класса для храненияконстант………………..26

3.3.11. Разработка класса игры…………………………………….27

3.4. Архитектура программного модуля………………………………...28

3.5. Разработка тестового приложения .……....………………………....29

4. Тестирование…………………………………………………………………..33

5. Приложение 1………………………………………………………………….43

1. **Постановка задачи**
   1. **Анализ предметной области**

Предметной областью курсового проекта является компьютерная игра «Snake».

Правила игры: чтобы выиграть, нужно съесть все яблоки на полянке, количество которых формируется рандомно или константно, в зависимости от уровня сложности, и выйти через норку, которая появится вверху. С каждым съеденным яблоком вы становитесь длиннее. Если уползете за пределы полянки, вы умрете. Если вы врежетесь в себя, то умрете. Если вам в голову попадет один из футбольных мячей, вы умрете. Если вы слишком долго не будете есть яблоки, они будут менять свои позиции. Скорость змейки и мячей задается рандомно в зависимости от выбранного уровня сложности.

Список терминов и определений:

* Игра “Snake” - игра впервые появилась на компьютерах и игровых автоматах в 70-х годах прошлого века. Цель игры заключалась в управлении существом (похожего на змею), которое ползает по плоскости и собирает еду, тем самым удлиняясь в собственных размерах.
* Пользователь - человек, который имеет, имел, или, возможно, будет иметь доступ к программе для игры.
* Операция – совокупность действий в приложении.
* Управление – координация действий.
* Выбор пользователя - наличие различных вариантов для осуществления воли.

Словарь предметной области:

* Полянка – поле по которому ползает змейка. Характеризуется заданным размером.
* Змеюка Майк – живое существо, обладающее разумом, которое ползает, увиливая от мячей, и ест яблоки. Характеризуется заданным начальным размером, скоростью передвижения.
* Футбольный мяч – этот объект передвигается по полю и пытается врезаться в голову змеюки. Характеризуется заданным размером и скоростью перемещения.
* Яблоко – фрукт, который ест змеюка Майк. Характеризуется заданным размером.
  1. **Формулировка технического задания**

В результате анализа предметной области можно сформулировать следующие функциональные требования:

* Разработать классы для организации взаимодействия объектов при заданных критериях
* Пользователь должен иметь возможность выбирать уровень сложности в игре.
* Каждый уровень сложности должен быть снабжен уникальной скорость перемещения змеюки и мячей, а также количеством генерируемых яблок.
* Система должна предоставлять пользователю средство для просмотра игрового процесса.
* Система должна оповещать пользователя о конечном исходе игры (выигрыш/проигрыш).
* Система должна позволять пользователю взаимодействовать со всеми элементами в меню.
* Система не должна аварийно завершаться.
* Система должна предоставлять музыкальное сопровождение на протяжении всего игрового процесса.
* Система должна предоставлять управление с клавиатуры во время игры.

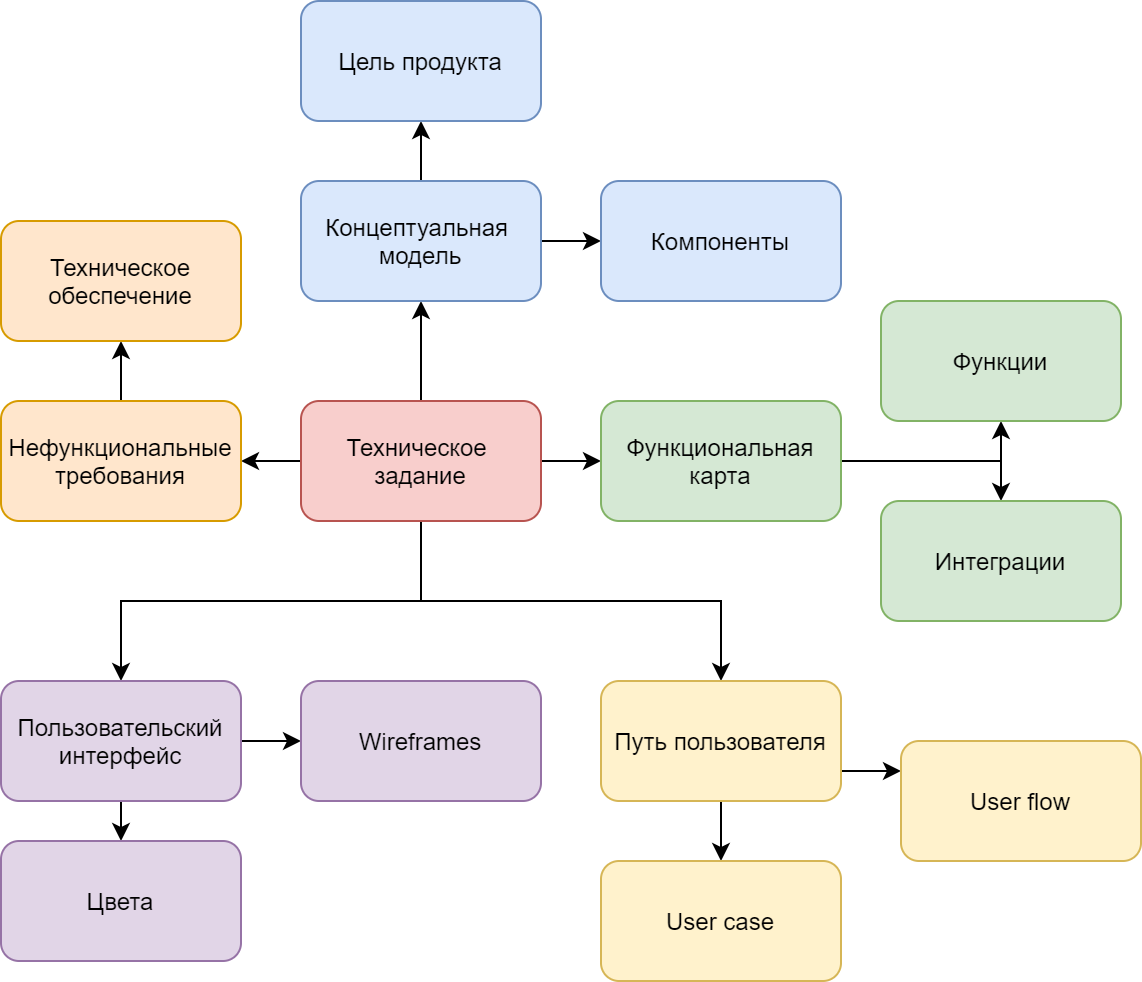


Рисунок 1 - Техническое задание

Концептуальная модель:

Приложение “Snake” предназначено для игры в змеюку Майка. Игра отличается по своим правилам и игровым элементам от классической змейки. Возрастного ограничения нет, подходит как детям, так и взрослым.

Компонентом является само приложение, которым пользуются пользователи для игры в змеюку Майка.

Функциональная карта:

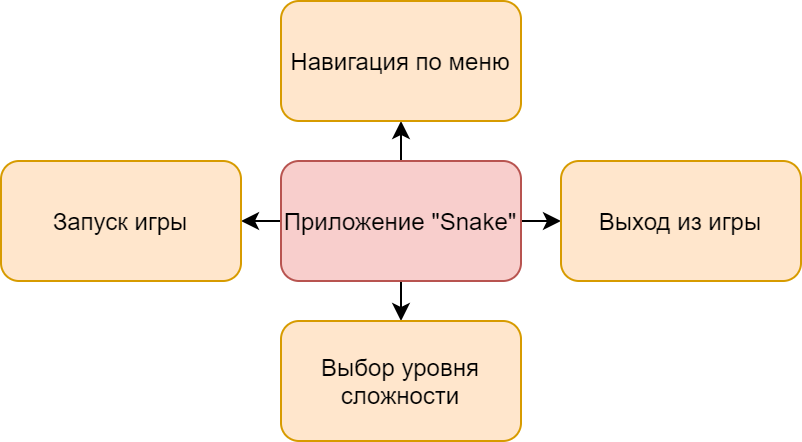


Рисунок 2 - Функциональная карта

Путь пользователя:

Userflow: пользователь запускает игру. Он выбирает в меню игры нужную вкладку (играть, уровень сложности, об игре, управление, выход). Во вкладке уровень сложности выпадает выбор этой самой сложности.

Usercase:

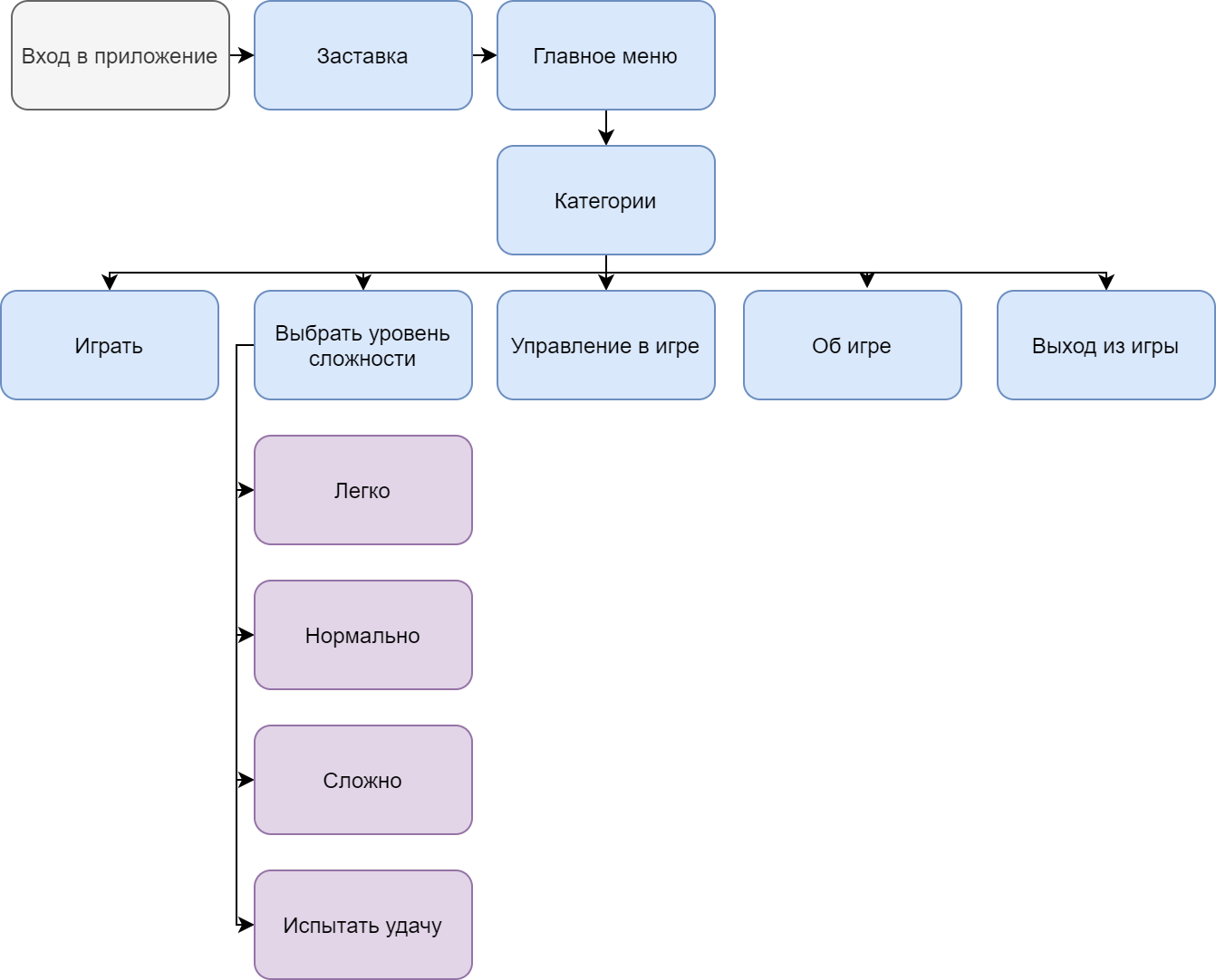


Рисунок 3 - Usercase

Нефункциональные требования:

Поддерживаются только устройства с операционной системой Windows(любой версии).

1. **Проектирование классов**
   1. **Разработанные классы**

* Snake - основной класс с игрой, в котором хранятся данные о всех сущностях игры.
* LevelWindow - класс, в котором выбирается уровень сложности. Задается скорость движения змеюки Майка и мячей, а также количество генерируемых яблок. Все это передается в класс Snake.
* Snake – основной класс, в котором координируются все действия во время игры.
* MainWindow – класс главного меню, контролирует открытие других окон (вкладок) меню.
* Images- класс с изображениями игровых элементов. Вызывается из класса Snake.
* Music- класс, содержащий музыкальные композиции в игре.Вызывается из класса Snake.
* Loading – класс, содержащий функцию загрузки игры.
* AboutWindow- класс, в котором написана информация об игре. Вызывается из класса MainWindow.
* ControlWindow- класс, в котором написано, как управлять в игре. Вызывается из класса MainWindow.
* WinWindow- класс, содержащий окно победы в игре. Вызывается из класса Snake.
* GameOverWindow- класс, содержащий окно проигрыша в игре. Вызывается из класса Snake.
* Паттерн Singleton. Был выбран для хранения константных значений, используемых в игре.

**2.2. Диаграмма классов**

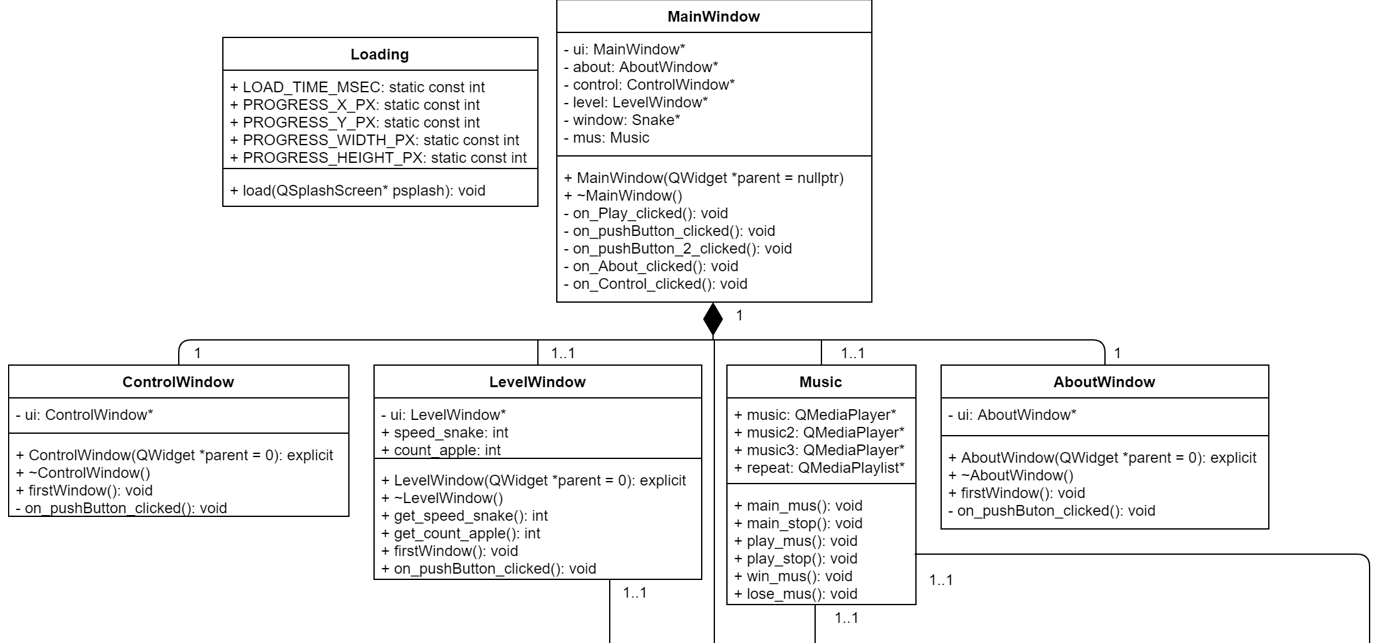


Рисунок 4 - Диаграмма классов

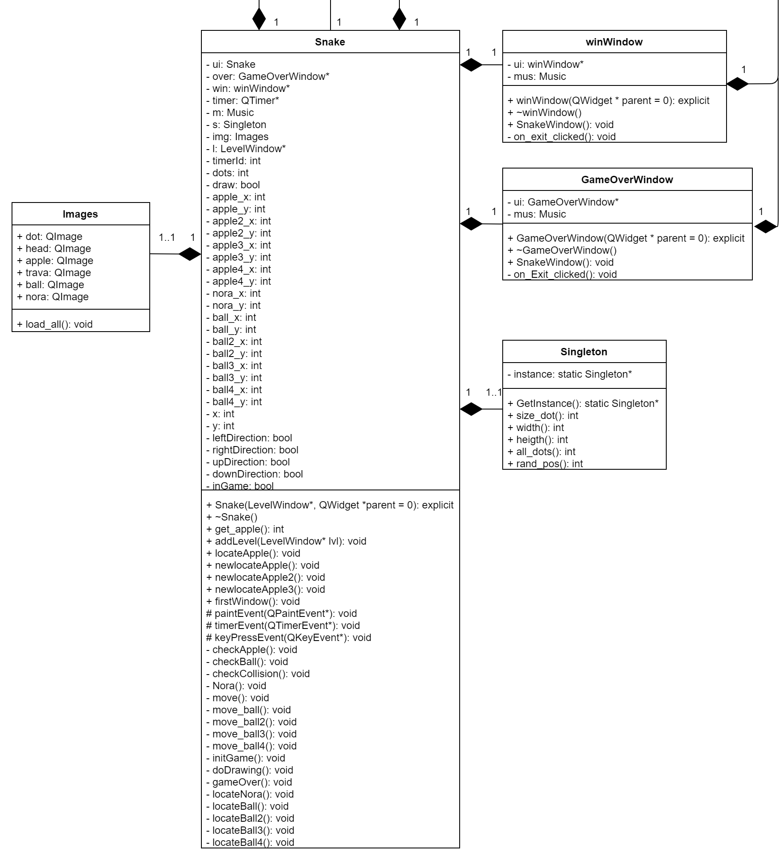


Рисунок 5 - Диаграмма классов

1. **Разработка приложения**

**3.1. Разработка интерфейса приложения**

Приступим к разработке интерфейса игры. В главное меню включим все разделы игры, по которым при помощи кнопок пользователь сможет переходить. Интерфейс меню будет выглядеть примерно так.



Рисунок 6 - Интерфейс главного меню

По кнопке «Сложность» пользователь сможет выбрать уровень сложности.

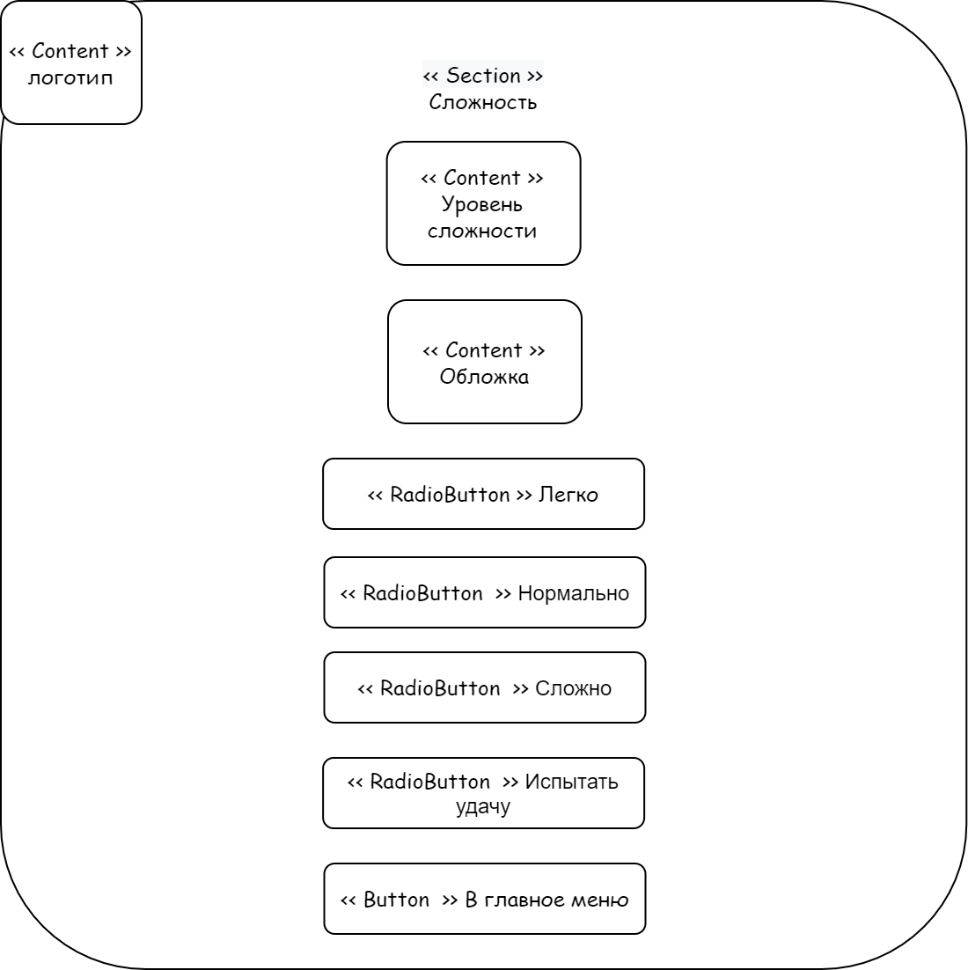


Рисунок 7 - Интерфейс «Сложность»

По кнопке «Как играть» пользователь сможет прочитать информацию про управление в игре.

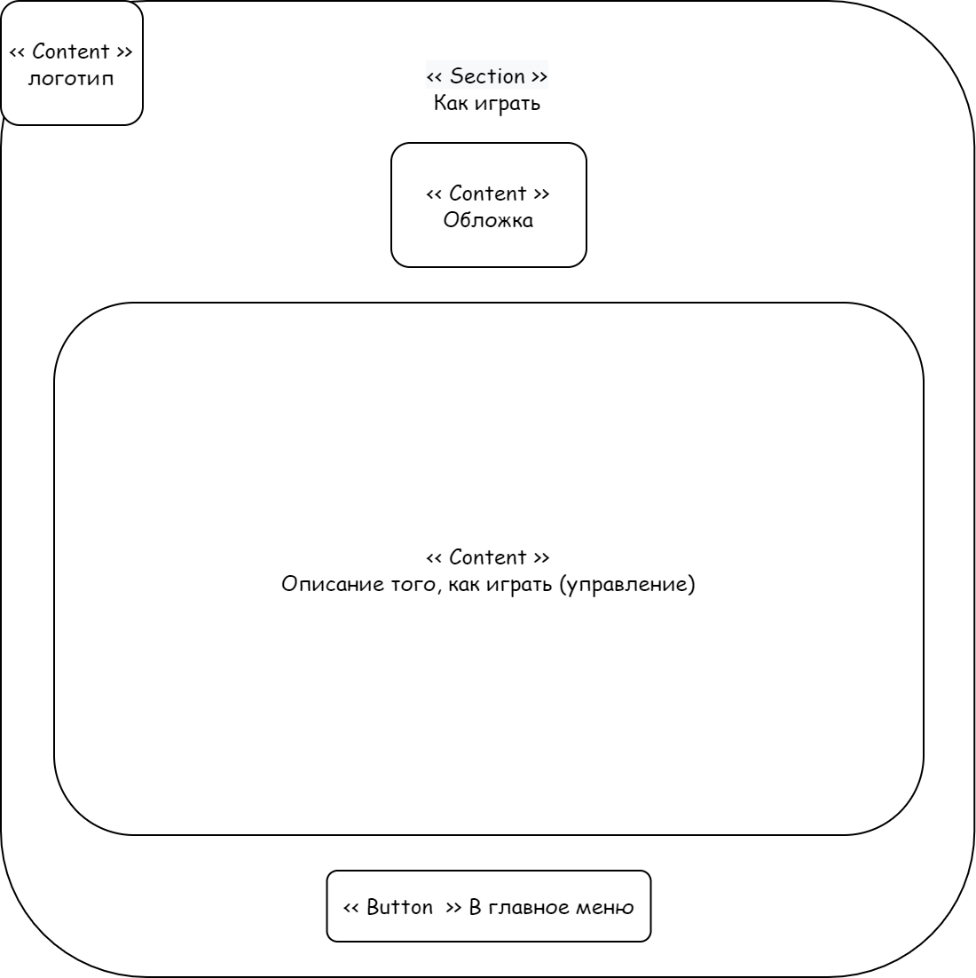


Рисунок 8 - Интерфейс «Как играть»

По кнопке «Об игре» пользователь сможет прочитать информацию об игре, познакомиться с персонажем змеюкой Майком, узнать, что нужно делать для того, чтобы пройти игру.

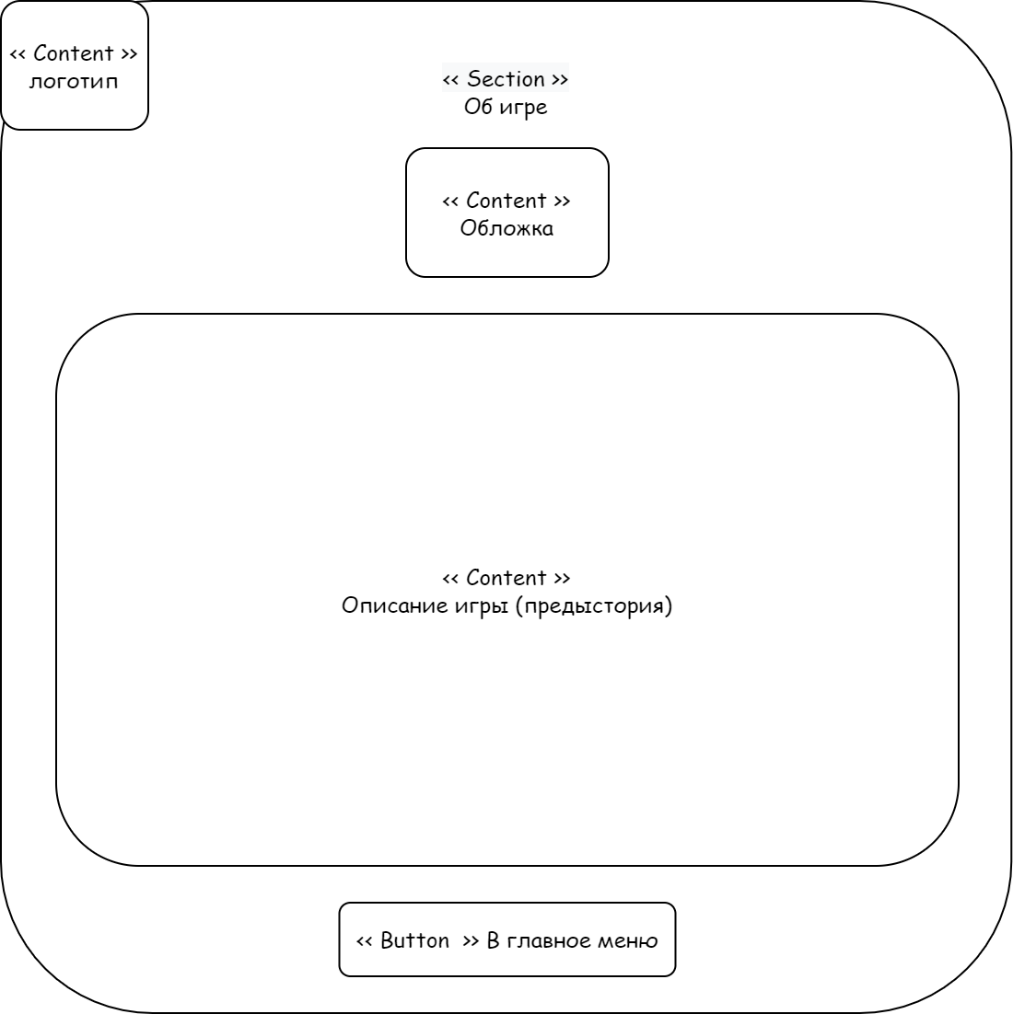


Рисунок 9 - Интерфейс «Об игре»

По кнопке «Играть» пользователь сможет начать играть в игру.

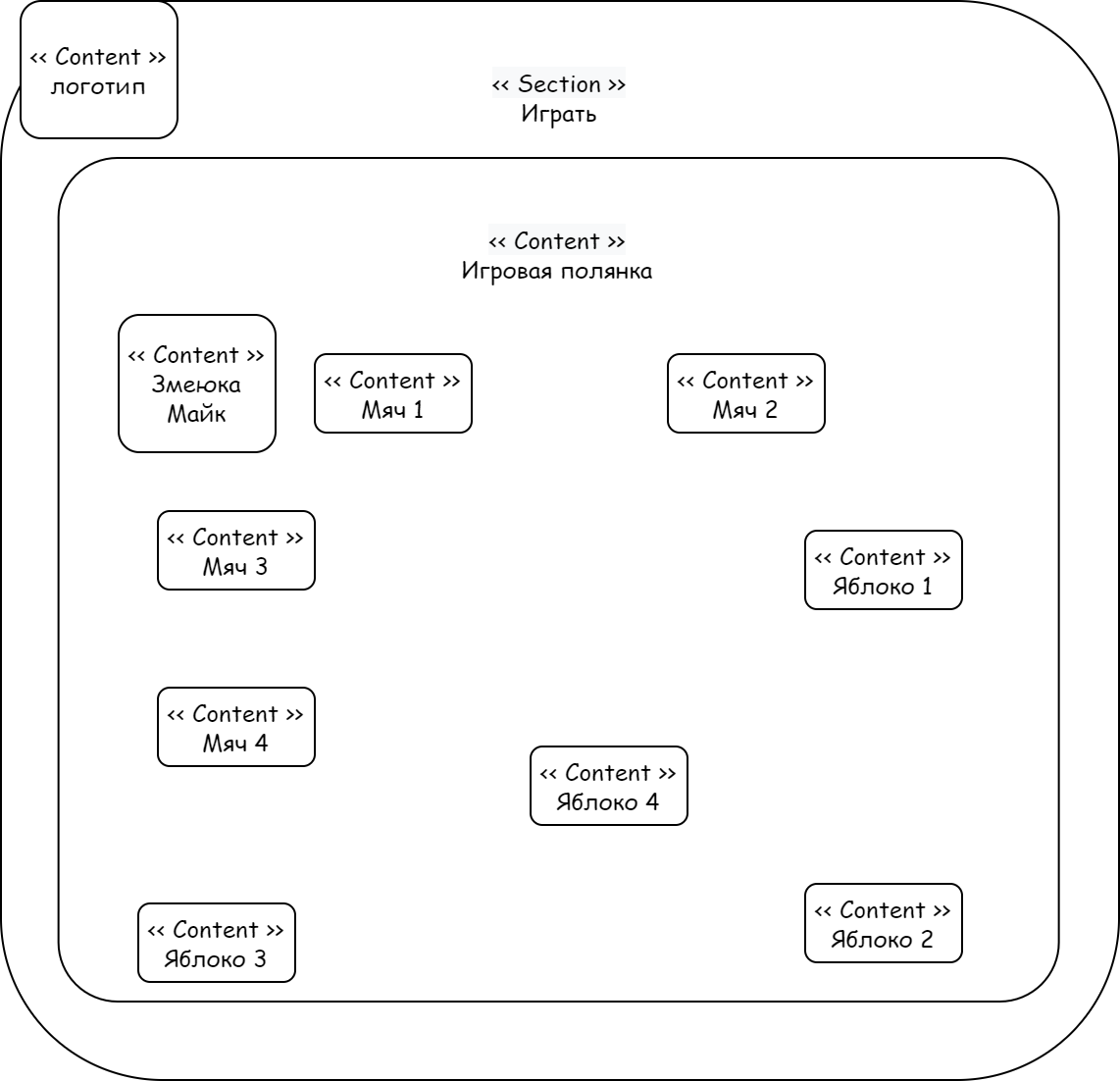


Рисунок 10 - Интерфейс «Играть»

Данное окно будет всплывать в случаевыигрыша.

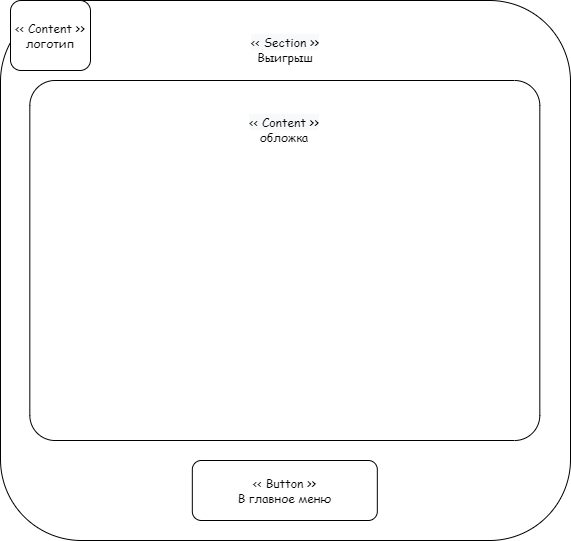


Рисунок 11 - Интерфейс «Выигрыш»

Данное окно будет всплывать в случаепроигрыша.

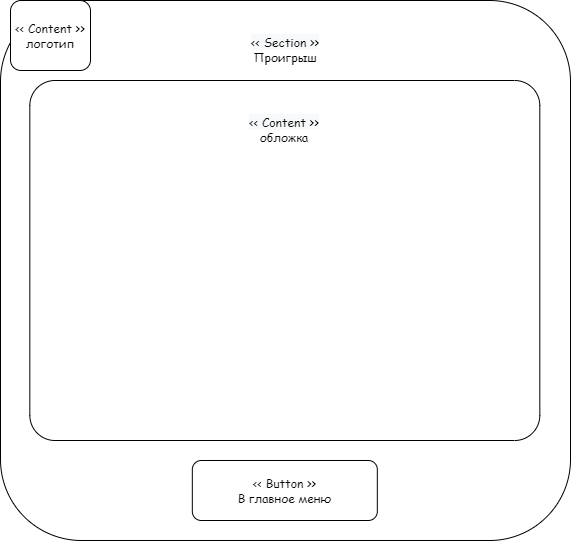


Рисунок 12 - Интерфейс «Проигрыш»

**3.2. Разработка элементов графического интерфейса**

Проанализировав предметную область, разработаем следующие графические элементы.

Основным цветовым решением приложения выбран зеленый цвет. Все заставки, изображения, игровые элементы были разработаны собственноручно. Ниже будут представлены данные изображения.



Рисунок 13 - Загрузочная заставка



Рисунок 14 - Главное меню



Рисунок 15 - Голова змеи

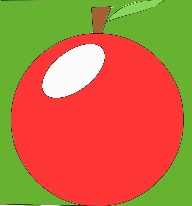


Рисунок 16 - Яблоко

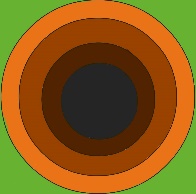


Рисунок 17 - Норка



Рисунок 18 - Мяч

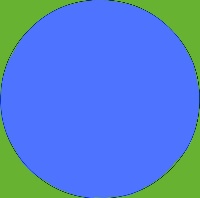


Рисунок 19 - Часть тела змеи

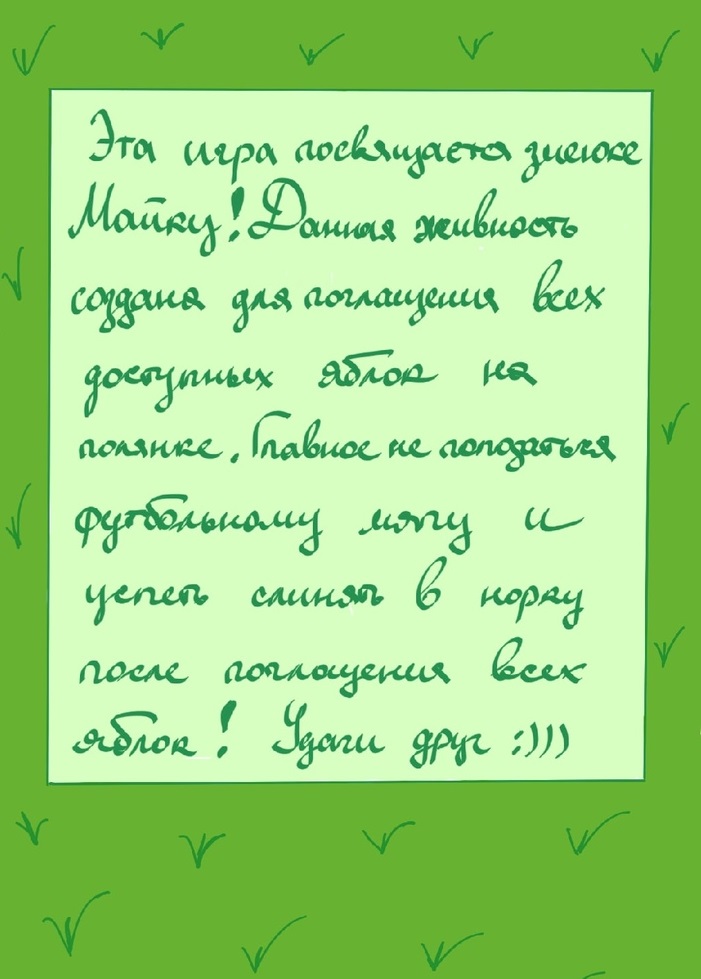


Рисунок 20 - Описание игры

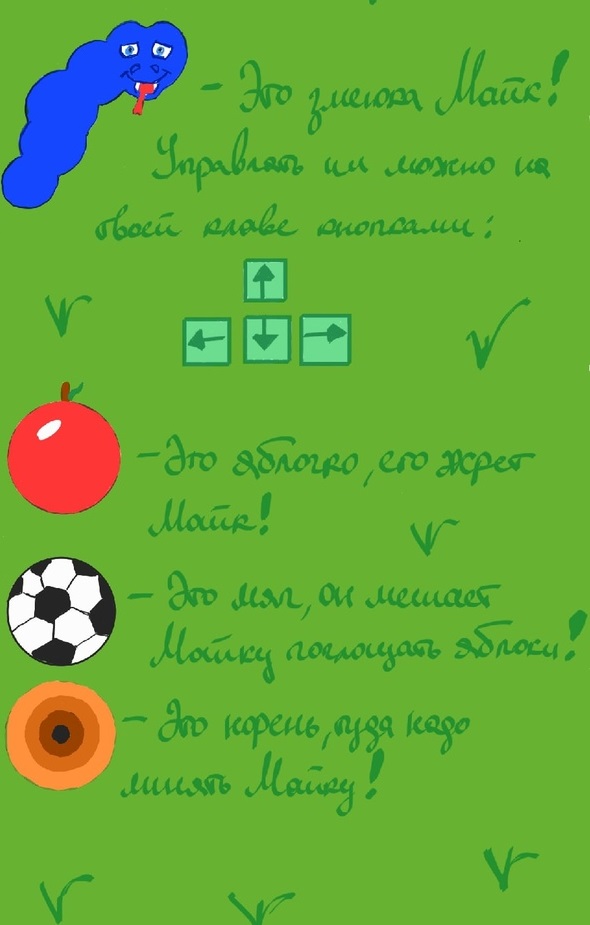


Рисунок 21 - Управление в игре

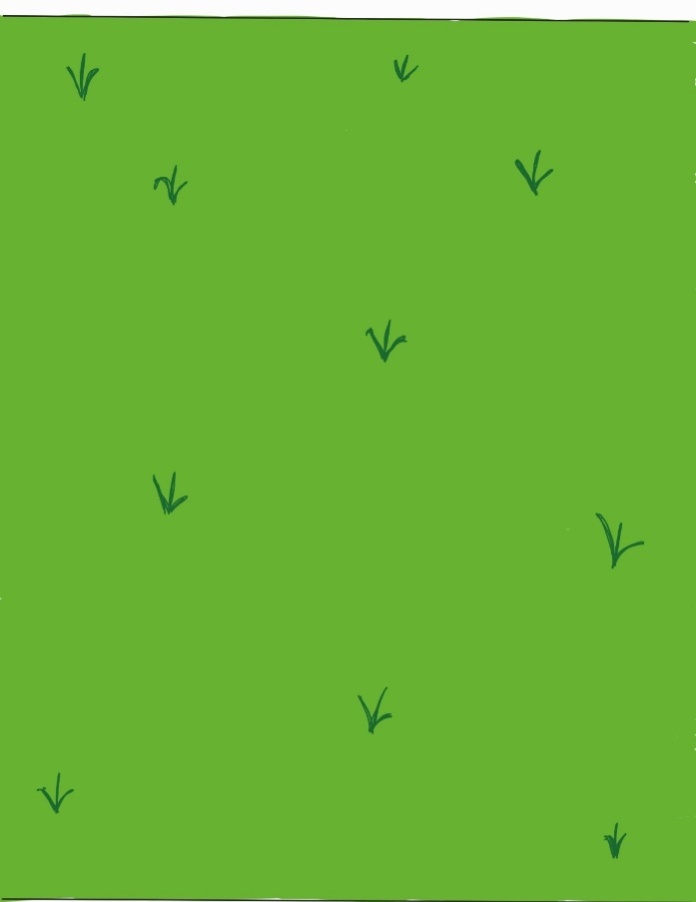


Рисунок 22 - Игровая полянка

**3.3. Реализация классов**

**3.3.1.Разработка класса загрузочного модуля**

В начале разработаем класс загрузочного окна с одним единственным методомload.

Итог: класс загрузочного окна.

classLoading

{

public:

voidload(QSplashScreen\* psplash); //загрузочнаязаставка

};

* + 1. **Разработка класса меню**

После того, как мы создали класс загрузки, приступим к разработке класса главного меню.

Из него мы будем переходить по различным окнам игры, поэтому создадим заранее классы, в которых описывается как управлять, о чем эта игра, уровень сложности, а также класс с музыкой и непосредственно с самой игрой.

При нажатии кнопки, например, играть, будем создавать объект класса «Snake» в методе нажатия соответствующей кнопки.Также с остальными перечисленными классами, которые связаны с главным меню кроме музыки.

* Класс «AboutWindow» предназначен для окна с описанием игры.
* Класс «ControlWindow» предназначен для окна с описанием управления в игре.
* Класс «LevelWindow» предназначен для окна с выбором уровня сложности.
* Класс «Music» предназначен для воспроизведения и остановки музыкальных файлов.

Итог: класс главного меню, наследуется от базового QMainWindow (главное окно приложения), а также связан при помощи композиции с классами ControlWindow, AboutWindow, LevelWindow, Music, Snake.

classMainWindow : public QMainWindow

{

public:

MainWindow(QWidget \*parent = nullptr);

~MainWindow();

private slots:

voidon\_Play\_clicked(); //игратьвигру

voidon\_Exit\_clicked(); //выходизигры

voidon\_Level\_clicked(); //уровеньсложности

void on\_About\_clicked(); //информация об игре

void on\_Control\_clicked(); //информация об управлении в игре

private:

Ui::MainWindow \*ui; //главноеокно

AboutWindow \*about; //окноописанияигры

ControlWindow \*control; //окноуправлениявигре

LevelWindow \*level; //окноуровнейсложности

Snake \*window; //окноигры

Music \*mus = new Music;//музыка

};

* + 1. **Разработка класса для описания игры**

Теперь опишем отдельно созданный класс «AboutWindow».

В нем в виде картинки будет отображаться вся нужная информация, в методе нажатия кнопки будем возвращаться в главное меню при помощи сигнала.

Для создания элементов в окне будем наследоваться от класса «Qwidget».

Итог: класс окна об игре, наследуется от QWidget, которыйявляется базовым для всех объектов пользовательского интерфейса.

classAboutWindow : public QWidget

{

public:

explicitAboutWindow(QWidget \*parent = nullptr);

~AboutWindow();

private slots:

voidon\_pushButton\_clicked(); //выходвглавноеменю

signals:

void firstWindow(); // Сигнал для открытия первого окна

private:

Ui::AboutWindow \*ui;

};

* + 1. **Разработка класса для описания управления в игре**

Опишем отдельно созданный класс «ControlWindow».

В нем в виде картинки будем отображать всю нужную информацию, в методе нажатия кнопки будем возвращаться в главное меню при помощи сигнала.

Для создания элементов в окне будем наследоваться от класса «Qwidget».

Итог: класс окна об управлении в игре, наследуется от QWidget, которыйявляется базовым для всех объектов пользовательского интерфейса.

classControlWindow : public QWidget

{

public:

explicitControlWindow(QWidget \*parent = nullptr);

~ControlWindow();

signals:

voidfirstWindow(); //сигналнаоткрытиепервогоокна

private slots:

voidon\_pushButton\_clicked(); //выходвглавноеменю

private:

Ui::ControlWindow \*ui;

};

* + 1. **Разработка класса уровней сложности**

Опишем отдельно созданный класс «LevelWindow».

В нем при помощиradiobuttonпользователь будет выбирать уровень сложности.

В методах get\_speed\_snake() и get\_count\_apple() будем устанавливать количество яблок и скорость змеюки в зависимости от выбранного пользователем уровня сложности.

Для создания элементов в окне будем наследоваться от класса «Qwidget».

Для отправки значений скорости и количества яблок этот класс связан с классом «Snake».

Итог: класс окна выбора уровня сложности в игре, наследуется от QWidget, которыйявляется базовым для всех объектов пользовательского интерфейса, а также связан при помощи композиции с классомSnake.

classLevelWindow : public QWidget

{

public:

explicitLevelWindow(QWidget \*parent = nullptr);

~LevelWindow();

int speed\_snake;

int count\_apple;

intget\_speed\_snake(); //скоростьзмеюкиимячей

intget\_count\_apple(); //количествояблокдляпоедания

signals:

void firstWindow(); //сигнал на открытие первого окна

public slots:

voidon\_pushButton\_clicked(); //выходвглавноеменю

private:

Ui::LevelWindow \*ui;

};

* + 1. **Разработка класса музыки**

Создадим класс музыки, который будет содержать в себе методы воспроизведения музыки и ее остановки для разных исходов в игре (выигрыш, проигрыш, игровой процесс).

Этот класс связан с классами главного меню, окна выигрыша, окна проигрыша и игрой.

Итог: класс музыки в игре, наследуется от QWidget, которыйявляется базовым для всех объектов пользовательского интерфейса, а также связан при помощи композиции с классами MainWindow, Snake, winWindow, GameOverWindow.

classMusic : public QWidget

{

public:

QMediaPlayer \* music;

QMediaPlayer \* music2;

QMediaPlayer \* music3;

QMediaPlaylist \* repeat;

voidmain\_mus(); //загрузкамузыкивглавномменю

voidmain\_stop();//остановкамузыки

voidplay\_mus(); //загрузкамузыкидляигры

void play\_stop();//остановка музыки

void win\_mus(); //загрузка музыки при победе

void lose\_mus(); //загрузка музыки при проигрыше

};

* + 1. **Разработка класса выигрышного класса**

Создадим класс выигрышного окна.

Он нужен для показа пользователю того, что он выиграл.

У него есть метод выхода из этого окна по сигналу, а также он связан с классом «Music» для воспроизведение выигрышного сигнала и с классом «Snake», так как данный класс выигрышного окна вызывается из класса игры.

Итог: класс выигрышного окна в игре, наследуется от QWidget, которыйявляется базовым для всех объектов пользовательского интерфейса, а также связан при помощи композиции с классами Music, Snake.

classwinWindow : public QWidget

{

public:

explicitwinWindow(QWidget \*parent = nullptr);

~winWindow();

private slots:

voidon\_Exit\_clicked(); //выходизпроигрышногоокна

signals:

voidSnakeWindow(); //сигналнавозврат

private:

Ui::winWindow \*ui;

Music \*mus = new Music; //музыка

};

* + 1. **Разработка класса проигрышного окна**

Создадим класс проигрышного окна.

Он нужен для показа пользователю того, что он проиграл.

У него есть метод выхода из этого окна по сигналу, а также он связан с классом «Music» для воспроизведение проигрышного сигнала и с классом «Snake», так как данный класс вызывается из класса игры.

Итог: класс проигрышного окна в игре, наследуется от QWidget, которыйявляется базовым для всех объектов пользовательского интерфейса, а также связан при помощи композиции с классами Music, Snake.

classGameOverWindow : public QWidget

{

public:

explicitGameOverWindow(QWidget \*parent = nullptr);

~GameOverWindow();

private slots:

voidon\_Exit\_clicked(); //выходизокна

signals:

voidSnakeWindow(); //сигналнавозврат

private:

Ui::GameOverWindow \*ui;

Music \*mus = new Music;

};

* + 1. **Разработка класса изображений**

Создадим класс изображений «Images».

Там будем хранить все изображения для игры.

Класс содержит один единственный метод, в котором все изображения загружаются.

Также он связан с классом «Snake», так как данный класс вызывается из класса игры.

Итог: класс изображений в игре, наследуется от QWidget, которыйявляется базовым для всех объектов пользовательского интерфейса, а также связан при помощи композиции с классом Snake.

classImages : public QWidget

{

public:

QImagedot; //1 частьтелазмеиилияблоко

QImage head; //голова

QImage apple; //яблоко

QImage trava; //трава

QImage ball; //мяч

QImage nora; //норень

voidload\_all(); //загрузкаизображений

};

* + 1. **Разработка класса для хранения констант**

Создадим паттерн «Singleton» (одиночка) для хранения константных значений в игре: размер окна (ширина, высота), размеры объектов. Также он связан с классом «Snake», так как данный класс вызывается из класса игры.

Итог: класс константных значений в игре, который является паттерном одиночкой, связан при помощи композиции с классом Snake.

classSingleton

{

public:

static Singleton \*GetInstance();

intsize\_dot();//размеряблока / 1йчастизмеи

intwidth(); //ширинаполя

intheigth(); //высотаполя

int all\_dots();//максимальное количество возможных точек на доске (900\*900)/(30\*30)

int rand\_pos();//для вычисления случайной позиции яблока

private:

static Singleton \* instance;

};

* + 1. **Разработка класса игры**

Самый главный класс «Snake», он относится непосредственно к самой игре.

Здесь реализованы все методы управления змеюкой, установки объектов, движения, отрисовки, проверки на завершение игры и так далее.

Соответственно данный класс связан со многими другими классами: музыки, главного меню, уровня сложности, изображений, окна выигрыша, окна проигрыша, паттерна «одиночка», чтобы вызывать из них нужные методы, которые не прописаны здесь.

Итог: класс основной игры, наследуется от QMainWindow, а также связан при помощи композиции с классами Music, MainWindow, LevelWindow, Images, winWindow, GameOverWindow, Singleton.

classSnake : public QMainWindow

{

public:

explicitSnake(LevelWindow\*, QWidget \*parent = nullptr);

~Snake();

protected:

voidpaintEvent(QPaintEvent \*); //отрисовкаигровыхэлементов

voidtimerEvent(QTimerEvent \*); //таймервигре

voidkeyPressEvent(QKeyEvent \*); //управлениесклавиатуры

public slots:

intget\_apple(); //кол-вояблок

voidaddLevel(LevelWindow\* lvl); //выборуровнясложности

//установкаяблок

voidlocateApple();

voidnewlocateApple();

voidnewlocateApple2();

voidnewlocateApple3();

signals:

voidfirstWindow(); //сигналнавозвратвглавноеменю

private:

GameOverWindow \*over; //окно проигрыша

winWindow \*win; //окно победы

QTimer \*timer; //включение таймера

Music m; //загрузка музыки

Singleton s; //использование констант с помощью паттерна

Images img; //загрузка игровых изображений

LevelWindow \*l; //уровень сложности

Ui::Snake \*ui;

int timerId; //скорость передвижения объектов

int dots; //части змеи

bool draw = true; //отрисовка объектов, пока все яблоки не съедены

//координатыяблок

int apple\_x;

int apple\_y;

int apple2\_x;

int apple2\_y;

int apple3\_x;

int apple3\_y;

int apple4\_x;

int apple4\_y;

//координаты норки

int nora\_x;

int nora\_y;

//координатымячей

int ball\_x;

int ball\_y;

int ball2\_x;

int ball2\_y;

int ball3\_x;

int ball3\_y;

int ball4\_x;

int ball4\_y;

//координатыполя

intx[900];

int y[900];

//направления движения змеи/мячей

bool leftDirection;

bool rightDirection;

bool upDirection;

bool downDirection;

boolinGame; //поканеконецигры

//проверка на съеденное яблоко

voidcheckApple();

voidcheckBall();

voidcheckCollision();

voidNora(); //попаданиевнорку

//движениемячей

voidmove();

voidmove\_ball();

voidmove\_ball2();

voidmove\_ball3();

void move\_ball4();

void initGame(); //инициализация(начальная установка всех элементов)

void doDrawing();//отрисовка игровых объектов

void gameOver(); //конец игры

void locateNora(); //установка норки на поле

//установка мячей на поле

voidlocateBall();

voidlocateBall2();

voidlocateBall3();

voidlocateBall4();

};

**3.4. Архитектура программного модуля**

Сформулируемархитектурупрограммногомодуля.

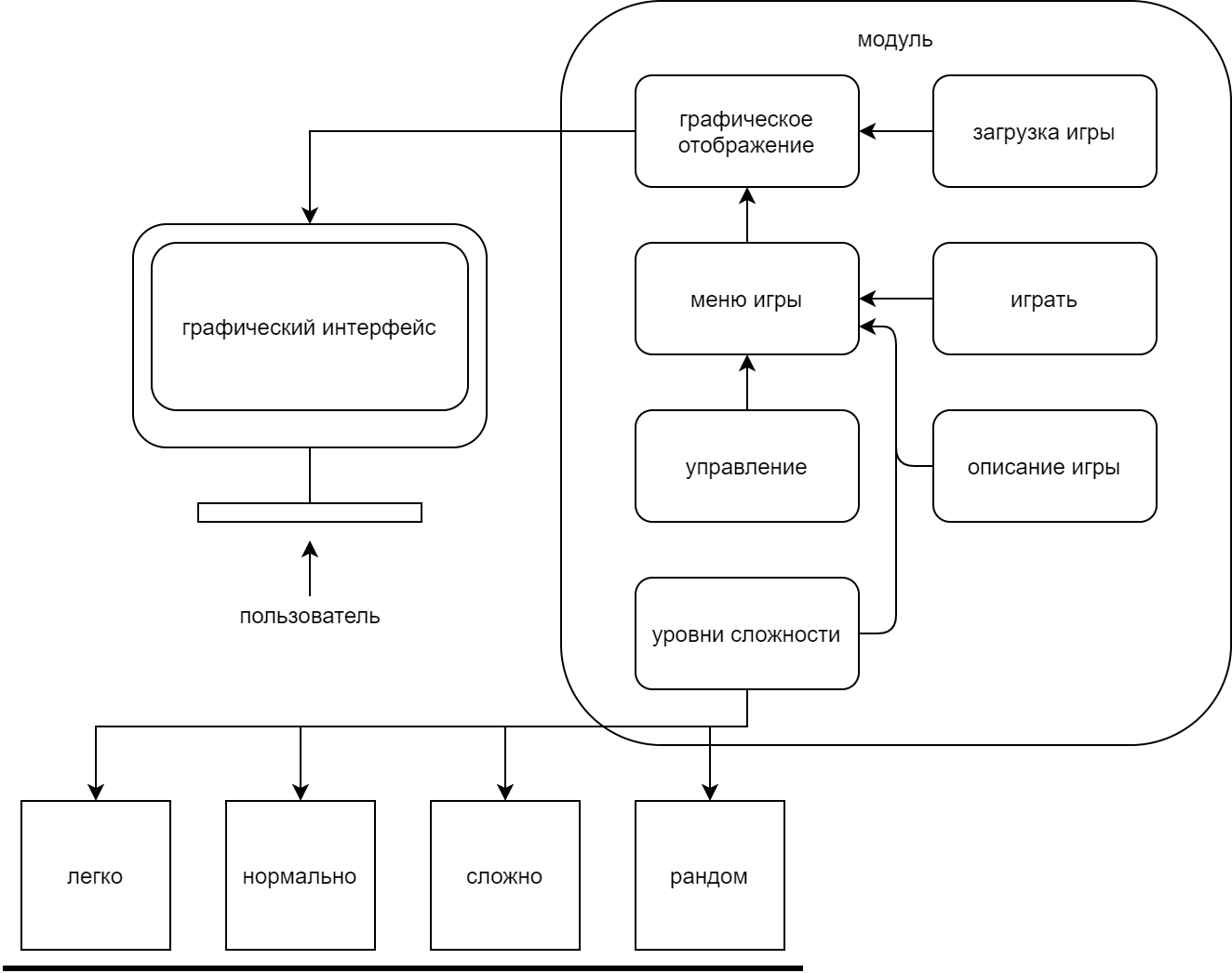


Рисунок 23 - Архитектура модуля

**3.5. Разработка тестового приложения.**

Определим алгоритмы методов, которые играют ключевую роль в игре «Змеюка Майк»:

1. Установка объектов на поле (яблоко, змеюка Майк, мяч, норка).

1.1. Яблоки ставятся через рандомный генератор.

Алгоритм:

*Устанавливить время, начать отсчет с текущего времени.*

*Отсчет у генератора установить в секундах.*

*Задать координате яблока по оси ОХ позицию через рандом.*

*Задать координате яблока по оси ОY позицию через рандом.*

1.2. Мячи ставятся на определенные позиции для того, чтобы они катались по полю в зависимости от направления движения Майка. Всего их 4.

Алгоритм:

*Задать координате мяча по оси ОХ позицию: координата поля по оси OX \* размер мяча.*

*Задать координате мяча по оси ОYпозицию: координата поля по оси OY \* размер мяча.*

1.3. Норка по условию задачи должна появиться в верху поля после того, как Майк съест все яблоки, поэтому для нее алгоритм точно такой же, как и для мяча.

1.4. Змеюку Майка мы будем ставить в начале игры в одно и тоже место.

Алгоритм:

*Задать начальный размер змеюки.*

*По всей длине змеюке:*

*Координате змеюки по оси ОХ задать положение: конст – часть тела \* конст.*

*Координате змеюки по оси OY задать положение: конст.*

1. Соответственно метод инициализации будет выглядеть следующим образом:

Алгоритм:

*Установить змеюки на поле.*

*Установить мячей на поле.*

*Установить яблок на поле.*

*Старт таймера, объекты движутся с заданной скоростью.*

1. Метод отрисовки объектов.

Алгоритм:

*Если игра не окончена:*

*Если яблоки закончились:*

*Остановить таймера.*

*Яблоки не рисуются.*

*Отрисовка норки с заданными координатами.*

*Установить норки на игровом поле.*

*Если яблоки не закончились:*

*Отрисовка яблок.*

*Отрисовка мячей.*

*По всей длине змеюки:*

*Если 1я часть тела змеюки:*

*Отрисовка головы змеюки.*

*Иначе:*

*Отрисовка туловища змеюки.*

*Иначе:*

*Закрыть игровое окно.*

1. Метод завершения игры.

Алгоритм:

*Вызов главного меню.*

*Открыть окно проигрыша/выигрыша.*

*Подписать окно и зафиксировать его размер.*

*Закрыть игрового окна.*

*Остановить воспроизведение музыки во время игры.*

*Включить музыку поражения/выигрыша.*

1. Метод проверки того, что яблоко съедено.

Алгоритм:

*Если не все яблоки съедены:*

*Если координата головы змеюки по оси ОХ и оси OY совпала с координатой яблока по оси ОХ и оси ОY:*

*Увеличить длину змеюки.*

*Уменьшить количество яблок.*

*Установить новое яблоко на поле.*

*Иначе:*

*Установить на поле норку.*

1. Метод проверки попадания головы змеюки в норку/мяч.

Алгоритм:

*Если координата головы змеюки по оси ОХ и оси OY совпала с координатой норки/мяча по оси ОХ и оси ОY:*

*Игра окончена.*

*Метод завершения игры.*

1. Метод перемещения мяча.

Рассмотрим перемещение одного мяча, для остальных алгоритм аналогичен, только меняется условие передвижения змеюки и координата мяча (Х/Y).

Алгоритм:

*Если змеюка движется вниз/влево и координата мяча по оси ОХ не превосходит границу поля:*

*Координата мяча по оси ОХ + размер мяча.*

*Если змеюка движется вверх/вправо и координата мяча по оси ОХ превосходит константную координату поля:*

*Координата мяча по оси ОХ - размер мяча.*

1. Метод перемещения змеюки Майка.

Алгоритм:

*Движение всей змеюки вслед за головой.Для всех частей тела, начиная с конца:*

*Координата по оси ОХ равна предыдущей.*

*Координата по оси ОYравна предыдущей.*

*Если влево:*

*Координата по оси ОХ – размер части тела.*

*Если вправо:*

*Координата по оси ОХ + размер части тела.*

*Если вверх:*

*Координата по оси ОY – размер части тела.*

*Если вниз:*

*Координата по оси ОY+ размер части тела.*

1. Метод проверки попадания змеюки в саму себя/выход за границы поля.

Алгоритм:

*Для всех частей змеюки:*

*Если змеюка длиннее 4 частей и 1я часть(голова) = другой части тела по координатам осей ОХ, ОY:*

*Игра окончена.*

*Метод завершения игры.*

*Если координата змеюки по оси ОY превышает высоту поля:*

*Игра окончена.*

*Метод завершения игры.*

*Если координата змеюки по оси ОХ превышает ширину поля:*

*Игра окончена.*

*Метод завершения игры.*

*Если координата змеюки по оси ОYменьше 0:*

*Игра окончена.*

*Метод завершения игры.*

*Если координата змеюки по оси ОХменьше 0:*

*Игра окончена.*

*Метод завершения игры.*

*Если не в игре:*

*Остановка таймера.*

1. Метод Таймера.

Алгоритм:

*Если игра идет:*

*Метод проверки на съеденное яблоко.*

*Метод проверки на попадание головы в мяч.*

*Метод проверки попадания змеюки в саму себя/выход за границы поля.*

*Метод перемещения змеюки.*

*Метод перемещения мячей.*

*Метод на проверку попадания в норку.*

*Перерисовка объектов.*

1. Метод обработки нажатия клавиш (стрелки, перемещение змеюки). Алгоритм:

*Если клавиша влево и змея не движется вправо:*

*Двигаться влево.*

*Если клавиша вправо и змея не движется влево:*

*Двигаться вправо.*

*Если клавиша вверх и змея не движется вниз:*

*Двигаться вверх.*

*Если клавиша вниз и змея не движется вверх:*

*Двигаться вниз.*

1. Метод загрузки игры. Перед включением игры происходит загрузка.

Алгоритм:

*Запустить таймер.*

*С 0 до 100%:*

*Если таймер истек:*

*Включить новый таймер.*

*Увеличить процент на 1.*

*Вывести сообщение о загрузке.*

Мы определили алгоритмы из основного класса с игрой, а также алгоритм загрузки игры. Остальные методы из меню были опущены, так как там все достаточно примитивно и одинаково реализовано.

1. **Тестирование**

В данном разделе мы покажем работоспособность игры в виде скриншотов.



Рисунок 24 - Загрузка игры

Для удобства пользователя загрузка происходит очень быстро, но стоит отметить, что она сделана искусственно для красоты, и не загружает ничего на самом деле.



Рисунок 25 - Главное меню игры

Меню содержит в себе несколько кнопок.

* Нажав на «Играть», мы можем сразу начать игру на легком уровне сложности.
* Нажав на «Сложность», мы можем выбрать 1 из 4 уровней сложности.
* Нажав на «Как играть», мы можем посмотреть мини инструкцию того, как управлять Майком.
* Нажав на «Об игре», мы можем узнать, что нужно делать в игре.
* Нажав на «Выход», мы можем выйти из игры.

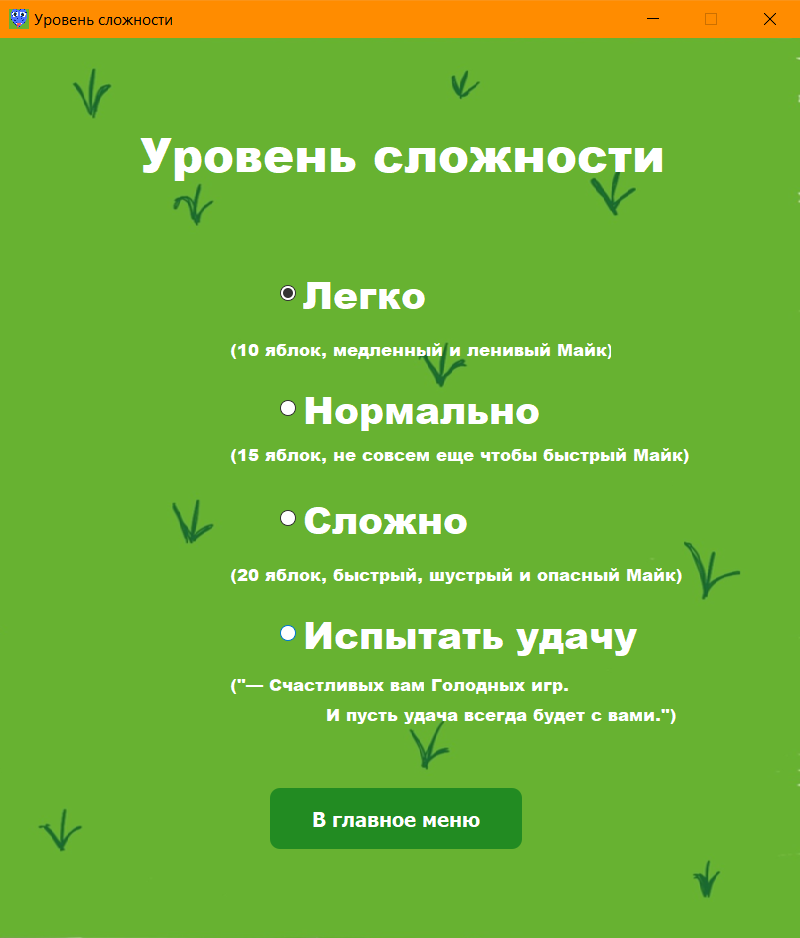


Рисунок 26 - Уровень сложности игры

Здесь мы можем выбрать нужный нам уровень сложности, также мы можем видеть то, чем отличаются предложенные уровни сложности друг от друга.

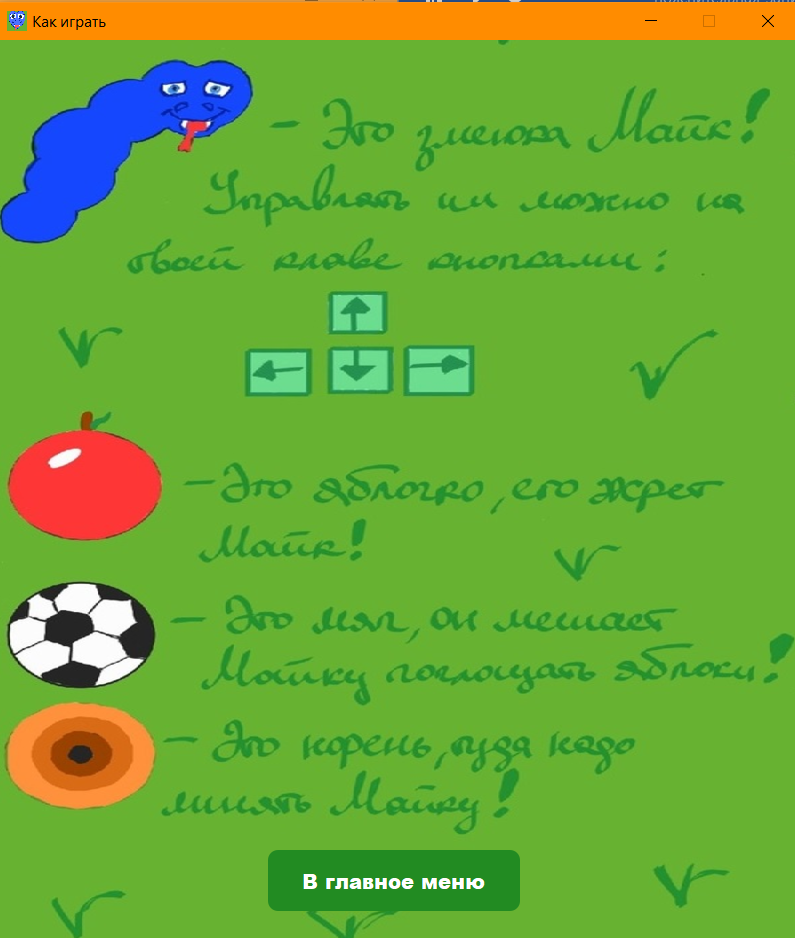


Рисунок 27 - Как играть в игру

В данном разделе меню мы видим всю информацию о действующих лицах/предметах, а также об управлении ими.

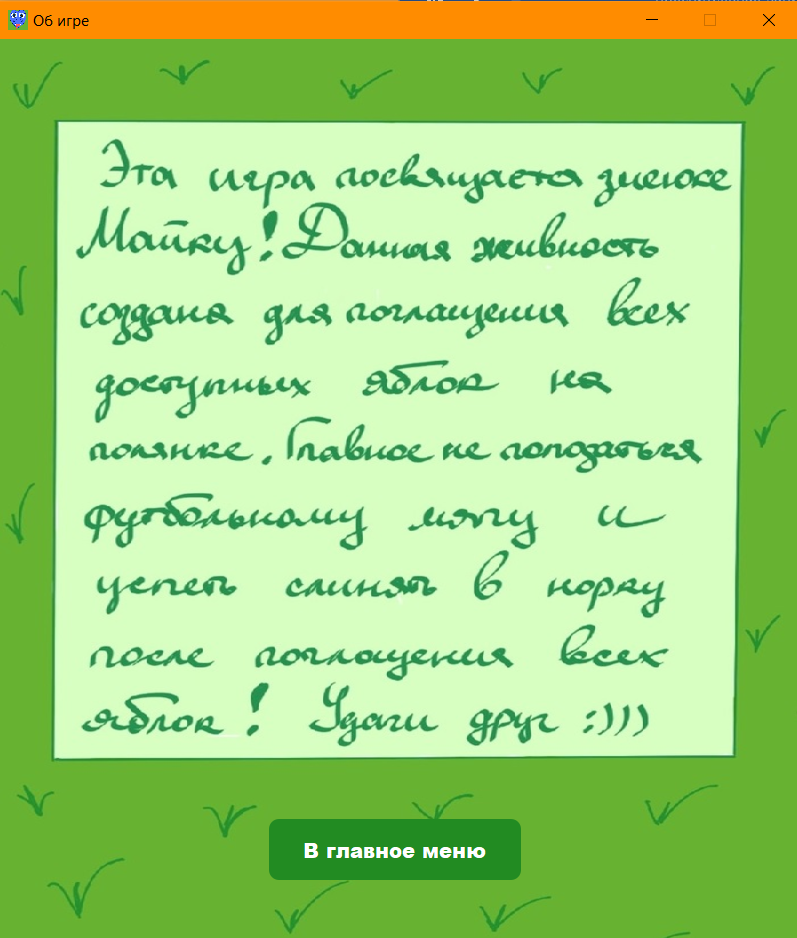


Рисунок 28 - Описание игры

Здесь мы можем увидеть описание процесса игры.



Рисунок 29 - Начальное положение объектов в игре

В начале игры мы будем всегда наблюдать это положение объектов. Для того, чтобы начать, нужно нажать на стрелочки на клавиатуре (управление змеюкой).



Рисунок 30 - Процесс игры

Здесь виден процесс игры, движение мячей зависит от направления движения змеюки. Яблоки меняют свое положение, если их вовремя не есть. Сверху показано количество оставшихся яблок.

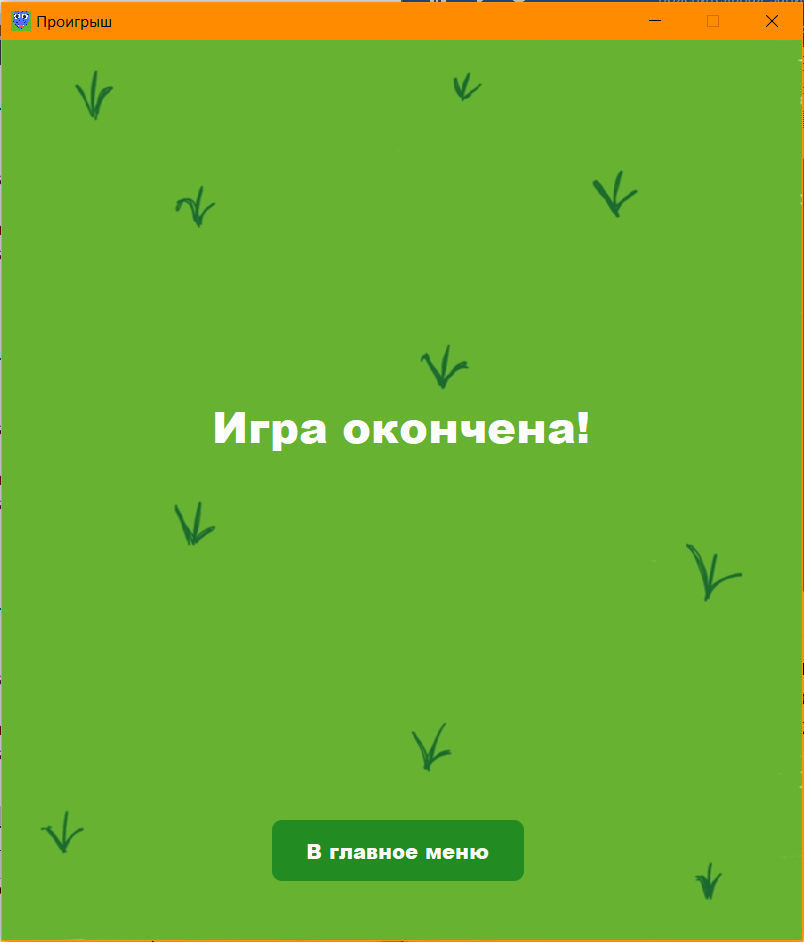


Рисунок 31 - Проигрыш в игре

Если Майк врезался в себя/мяч/стену, то игра заканчивается.



Рисунок 32 - Победа в игре

Если Майк сожрал все яблоки и залез в норку, которая появляется вверху поля, то игра завершается победой.



Рисунок 33 - Появление норки

Если Майк все съел, то вот такая норка будет ждать его, но до нее нужно еще добраться!

В заключении тестирования нужно отметить, что во время всего игрового процесса присутствует музыкальное сопровождение. Скриншотами, как известно, это показать невозможно. Из недостатков я бы отметила то, что можно было бы предоставить пользователю выбирать цвет змеюки, фрукты, которые змеюка будет поедать, но это не было реализовано, так как лапки автора устали рисовать. Уровни сложности представлены неплохо, возможно можно было еще чего-нибудь понапридумывать, но автор действовал в рамках задания. Все элементы игры нарисованы автором, поэтому таких змеюк Вы нигде не встретите в своей жизни ☺.

Текст программы представлен в приложении 1.

# **Приложение 1**

*Текст программы*

Aboutwindow.h

#ifndefABOUTWINDOW\_H

#defineABOUTWINDOW\_H

#include<QWidget>

#include<QDir>

namespace**Ui**{

class**AboutWindow**;

}

class**AboutWindow**:publicQWidget

{

Q\_OBJECT

public:

explicit**AboutWindow**(QWidget\*parent=nullptr);

~***AboutWindow***();

privateslots:

void**on\_pushButton\_clicked**();//выходвглавноеменю

signals:

void**firstWindow**();//Сигналдляоткрытияпервогоокна

private:

Ui::AboutWindow\*ui;

};

#endif//ABOUTWINDOW\_H

Aboutwindow.cpp

#include"aboutwindow.h"

#include"ui\_aboutwindow.h"

AboutWindow::**AboutWindow**(QWidget\*parent):

QWidget(*parent*),

ui(newUi::AboutWindow)

{

ui->setupUi(this);

//установкафоновогоизображения

QPixmaptrava(QCoreApplication::applicationDirPath()+QDir::toNativeSeparators(QDir::separator())+"about.png");

trava=trava.scaled(this->size(),Qt::IgnoreAspectRatio);

QPalettepalette;

palette.setBrush(QPalette::Background,trava);

this->setPalette(palette);

}

AboutWindow::~***AboutWindow***()

{

deleteui;

}

voidAboutWindow::**on\_pushButton\_clicked**()

{

this->close();//Закрываемокно

emitfirstWindow();//Ивызываемсигналнаоткрытиеглавногоокна

}

Controlwindow.h

#ifndefCONTROLWINDOW\_H

#defineCONTROLWINDOW\_H

#include<QWidget>

#include<QDir>

namespace**Ui**{

class**ControlWindow**;

}

class**ControlWindow**:publicQWidget

{

Q\_OBJECT

public:

explicit**ControlWindow**(QWidget\*parent=nullptr);

~***ControlWindow***();

signals:

void**firstWindow**();//Сигналнаоткрытиепервогоокна

privateslots:

void**on\_pushButton\_clicked**();//выходвглавноеменю

private:

Ui::ControlWindow\*ui;

};

#endif//CONTROLWINDOW\_H

Controlwindow.cpp

#include"controlwindow.h"

#include"ui\_controlwindow.h"

ControlWindow::**ControlWindow**(QWidget\*parent):

QWidget(*parent*),

ui(newUi::ControlWindow)

{

ui->setupUi(this);

//установкафоновогоизображения

QPixmaptrava(QCoreApplication::applicationDirPath()+QDir::toNativeSeparators(QDir::separator())+"control.png");

trava=trava.scaled(this->size(),Qt::IgnoreAspectRatio);

QPalettepalette;

palette.setBrush(QPalette::Background,trava);

this->setPalette(palette);

}

ControlWindow::~***ControlWindow***()

{

deleteui;

}

voidControlWindow::**on\_pushButton\_clicked**()

{

this->close();//Закрываемокно

emitfirstWindow();//Ивызываемсигналнаоткрытиеглавногоокна

}

Gameoverwindow.h

#ifndefGAMEOVERWINDOW\_H

#defineGAMEOVERWINDOW\_H

#include<QWidget>

#include<QProcess>

#include<QApplication>

#include"music.h"

#include<QDir>

namespace**Ui**{

class**GameOverWindow**;

}

class**GameOverWindow**:publicQWidget

{

Q\_OBJECT

public:

explicit**GameOverWindow**(QWidget\*parent=nullptr);

~***GameOverWindow***();

privateslots:

void**on\_Exit\_clicked**();//выходизокна

signals:

void**SnakeWindow**();//сигналнавозврат

private:

Ui::GameOverWindow\*ui;

Music\*mus=newMusic;

};

#endif//GAMEOVERWINDOW\_H

Gameoverwindow.cpp

#include"gameoverwindow.h"

#include"ui\_gameoverwindow.h"

GameOverWindow::**GameOverWindow**(QWidget\*parent):

QWidget(*parent*),

ui(newUi::GameOverWindow)

{

ui->setupUi(this);

//установкафоновогоизображения

QPixmaptrava(QCoreApplication::applicationDirPath()+QDir::toNativeSeparators(QDir::separator())+"trava.png");

trava=trava.scaled(this->size(),Qt::IgnoreAspectRatio);

QPalettepalette;

palette.setBrush(QPalette::Background,trava);

this->setPalette(palette);

}

GameOverWindow::~***GameOverWindow***()

{

deleteui;

deletemus;

}

voidGameOverWindow::**on\_Exit\_clicked**()

{

this->close();

mus->main\_mus();//включениемузыкиглавногоменю

//QProcess::startDetached(QApplication::applicationFilePath(),QStringList(),QApplication::applicationDirPath());

emitSnakeWindow();//вызываемсигналнаоткрытиепредыдущегоокна

}

Images.h

#ifndefIMAGES\_H

#defineIMAGES\_H

#include<QWidget>

#include<QDir>

#include<QApplication>

class**Images**:publicQWidget

{

Q\_OBJECT

public:

QImagedot;//1частьтелазмеиилияблоко

QImagehead;//голова

QImageapple;//яблоко

QImagetrava;//трава

QImageball;//мяч

QImagenora;//норень

void**load\_all**();//загрузкаизображений

};

#endif//IMAGES\_H

Images.cpp

#include"images.h"

voidImages::**load\_all**(){

dot.load(QCoreApplication::applicationDirPath()+QDir::toNativeSeparators(QDir::separator())+"dot.png");

head.load(QCoreApplication::applicationDirPath()+QDir::toNativeSeparators(QDir::separator())+"head.png");

ball.load(QCoreApplication::applicationDirPath()+QDir::toNativeSeparators(QDir::separator())+"ball.png");

nora.load(QCoreApplication::applicationDirPath()+QDir::toNativeSeparators(QDir::separator())+"nora.png");

apple.load(QCoreApplication::applicationDirPath()+QDir::toNativeSeparators(QDir::separator())+"apple.png");

}

Levelwindow.h

#ifndefLEVELWINDOW\_H

#defineLEVELWINDOW\_H

#include<QWidget>

#include<QDir>

namespace**Ui**{

class**LevelWindow**;

}

class**LevelWindow**:publicQWidget

{

Q\_OBJECT

public:

explicit**LevelWindow**(QWidget\*parent=nullptr);

~***LevelWindow***();

intspeed\_snake;

intcount\_apple=10;

int**get\_speed\_snake**();//скоростьзмеюкиимячей

int**get\_count\_apple**();//количествояблокдляпоедания

signals:

void**firstWindow**();//сигналнаоткрытиепервогоокна

publicslots:

void**on\_pushButton\_clicked**();//выходвглавноеменю

private:

Ui::LevelWindow\*ui;

};

#endif//LEVELWINDOW\_H

Levelwindow.cpp

#include"levelwindow.h"

#include"ui\_levelwindow.h"

#include<QTime>

#include<QRandomGenerator>

LevelWindow::**LevelWindow**(QWidget\*parent):

QWidget(*parent*),

ui(newUi::LevelWindow)

{

ui->setupUi(this);

//установкафоновогоизображения

QPixmaptrava(QCoreApplication::applicationDirPath()+QDir::toNativeSeparators(QDir::separator())+"trava.png");

trava=trava.scaled(this->size(),Qt::IgnoreAspectRatio);

QPalettepalette;

palette.setBrush(QPalette::Background,trava);

this->setPalette(palette);

connect(ui->radioButton,SIGNAL(toggled()),this,SLOT(isChecked()));

connect(ui->radioButton\_2,SIGNAL(toggled()),this,SLOT(isChecked()));

connect(ui->radioButton\_3,SIGNAL(toggled()),this,SLOT(isChecked()));

connect(ui->radioButton\_4,SIGNAL(toggled()),this,SLOT(isChecked()));

ui->radioButton->setChecked(true);//изначальновыбранлегкийуровеньсложности

ui->radioButton\_2->setChecked(false);

ui->radioButton\_3->setChecked(false);

ui->radioButton\_4->setChecked(false);

}

LevelWindow::~***LevelWindow***()

{

deleteui;

}

//вглавноеменю

voidLevelWindow::**on\_pushButton\_clicked**()

{

this->count\_apple=get\_count\_apple();//выбранноекол-вояблок

this->speed\_snake=get\_speed\_snake();//выбраннаяскорость

this->close();//Закрываемокно

emitfirstWindow();//Ивызываемсигналнаоткрытиеглавногоокна

}

intLevelWindow::**get\_speed\_snake**()

{

QTimetime=QTime::currentTime();

QRandomGeneratorqrand((uint)time.msec());

if(ui->radioButton->isChecked()){//выборскорости

speed\_snake=120+(qrand()%1);

returnspeed\_snake;

}

if(ui->radioButton\_2->isChecked()){

speed\_snake=100+(qrand()%1);

returnspeed\_snake;

}

if(ui->radioButton\_3->isChecked()){

speed\_snake=80+(qrand()%1);

returnspeed\_snake;

}

if(ui->radioButton\_4->isChecked()){

speed\_snake=120+(qrand()%(120-100)+1);

returnspeed\_snake;

}

speed\_snake=0;

returnspeed\_snake;

}

intLevelWindow::**get\_count\_apple**()

{

QTimetime=QTime::currentTime();

QRandomGeneratorqrand((uint)time.msec());

if(ui->radioButton->isChecked()){//выборкол-ваяблок

count\_apple=10+(qrand()%1);

returncount\_apple;

}

if(ui->radioButton\_2->isChecked()){

count\_apple=15+(qrand()%1);

returncount\_apple;

}

if(ui->radioButton\_3->isChecked()){

count\_apple=20+(qrand()%1);

returncount\_apple;

}

if(ui->radioButton\_4->isChecked()){

count\_apple=10+(qrand()%(20-10)+1);

returncount\_apple;

}

returncount\_apple;

}

Loading.h

#ifndefLOADING\_H

#defineLOADING\_H

#include<QApplication>

#include<QLabel>

#include<QSplashScreen>

#include<QPainter>

#include<QTime>

staticconstintLOAD\_TIME\_MSEC=5\*1000;

staticconstintPROGRESS\_X\_PX=272;

staticconstintPROGRESS\_Y\_PX=333;

staticconstintPROGRESS\_WIDTH\_PX=310;

staticconstintPROGRESS\_HEIGHT\_PX=28;

class**Loading**

{

public:

void**load**(QSplashScreen\*psplash);//загрузочнаязаставка

};

#endif//LOADING\_H

Loading.cpp

#include"loading.h"

voidLoading::**load**(QSplashScreen\*psplash)

{

QTimetime;

time.start();

//загрузкаигры

for(inti=0;i<100;){

if(time.elapsed()>5){

time.start();

++i;

}

//сообщениеозагрузкеиегосвойства

psplash->showMessage("Loadingmodules:"

+QString::number(i)+"%",

Qt::AlignRight|Qt::AlignBottom,

Qt::black);

}

}

Mainwindow.h

#ifndefMAINWINDOW\_H

#defineMAINWINDOW\_H

#include<QMainWindow>

#include"snake.h"

#include"aboutwindow.h"

#include"controlwindow.h"

#include"levelwindow.h"

#include"music.h"

#include<QDir>

QT\_BEGIN\_NAMESPACE

namespace**Ui**{class**MainWindow**;}

QT\_END\_NAMESPACE

class**MainWindow**:publicQMainWindow

{

Q\_OBJECT

public:

**MainWindow**(QWidget\*parent=nullptr);

~***MainWindow***();

privateslots:

void**on\_Play\_clicked**();//игратьвигру

void**on\_pushButton\_clicked**();//выходизигры

void**on\_pushButton\_2\_clicked**();//уровеньсложности

void**on\_About\_clicked**();//информацияобигре

void**on\_Control\_clicked**();//информацияобуправлениивигре

private:

Ui::MainWindow\*ui;//главноеокно

AboutWindow\*about;//окноописанияигры

ControlWindow\*control;//окноуправлениявигре

LevelWindow\*level;//окноуровнейсложности

Snake\*window;//окноигры

Music\*mus=newMusic;//музыка

};

#endif//MAINWINDOW\_H

Mainwindow.cpp

#include"mainwindow.h"

#include"ui\_mainwindow.h"

MainWindow::**MainWindow**(QWidget\*parent)

:QMainWindow(*parent*)

,ui(newUi::MainWindow)

{

ui->setupUi(this);

mus->main\_mus();//музыкаглавногоменю

//установкафоновогоизображения

QPixmaptrava(QCoreApplication::applicationDirPath()+QDir::toNativeSeparators(QDir::separator())+"cover.png");

trava=trava.scaled(this->size(),Qt::IgnoreAspectRatio);

QPalettepalette;

palette.setBrush(QPalette::Background,trava);

this->setPalette(palette);

//окноописанияигры

about=newAboutWindow();

connect(about,&AboutWindow::firstWindow,this,&MainWindow::show);

//окноуправлениявигре

control=newControlWindow();

connect(control,&ControlWindow::firstWindow,this,&MainWindow::show);

//окновыборауровнясложности

level=newLevelWindow();

connect(level,&LevelWindow::firstWindow,this,&MainWindow::show);

}

MainWindow::~***MainWindow***()

{

deleteui;

deletemus;

deleteabout;

deletelevel;

deletecontrol;

deletewindow;

}

//игратьвзмеюку

voidMainWindow::**on\_Play\_clicked**()

{

//окноигры

window=newSnake(*level*);

connect(window,&Snake::firstWindow,this,&MainWindow::show);

window->setWindowTitle("Змеюка");//установканазванияокна

window->setFixedSize(900,900);//фиксированныйразмерокна

window->show();//открытиеокнаигры

this->close();//закрытиеокнаменю

}

//выходизигры

voidMainWindow::**on\_pushButton\_clicked**()

{

this->close();

}

//выборсложности

voidMainWindow::**on\_pushButton\_2\_clicked**()

{

level->setFixedSize(800,900);

level->show();

level->setWindowTitle("Уровеньсложности");

this->close();

}

//обигре

voidMainWindow::**on\_About\_clicked**()

{

about->setFixedSize(800,900);

about->show();

about->setWindowTitle("Обигре");

this->close();

}

//управлениевигре

voidMainWindow::**on\_Control\_clicked**()

{

control->setFixedSize(800,900);

control->show();

control->setWindowTitle("Какиграть");

this->close();

}

Music.h

#ifndefMUSIC\_H

#defineMUSIC\_H

#include<QWidget>

#include<QMediaPlayer>

#include<QMediaPlaylist>

#include<QDir>

#include<QApplication>

class**Music**:publicQWidget

{

Q\_OBJECT

public:

QMediaPlayer\*music;

QMediaPlayer\*music2;

QMediaPlayer\*music3;

QMediaPlaylist\*repeat;

void**main\_mus**();//загрузкамузыкивглавномменю

void**main\_stop**();//остановкамузыки

void**play\_mus**();//загрузкамузыкидляигры

void**play\_stop**();//остановкамузыки

void**win\_mus**();//загрузкамузыкиприпобеде

void**lose\_mus**();//загрузкамузыкиприпроигрыше

};

#endif//MUSIC\_H

Music.cpp

#include"music.h"

voidMusic::**main\_mus**(){

music=newQMediaPlayer();

music->setMedia(QUrl::fromLocalFile(QCoreApplication::applicationDirPath()+QDir::toNativeSeparators(QDir::separator())+"main.mp3"));//чтениемуз.файла

music->setVolume(100);//установкагромкости

music->play();//включениемузыки

repeat=newQMediaPlaylist();//повтормузыкальнойкомпозиции

repeat->setPlaybackMode(QMediaPlaylist::CurrentItemInLoop);

}

voidMusic::**main\_stop**(){

music->stop();//остановкавоспроизведениямуз.файла

}

voidMusic::**play\_mus**(){

music2=newQMediaPlayer();

music2->setMedia(QUrl::fromLocalFile(QCoreApplication::applicationDirPath()+QDir::toNativeSeparators(QDir::separator())+"playing.mp3"));

music2->setVolume(100);

music2->play();

repeat=newQMediaPlaylist();

repeat->setPlaybackMode(QMediaPlaylist::Loop);

}

voidMusic::**play\_stop**(){

music2->stop();

}

voidMusic::**win\_mus**(){

music=newQMediaPlayer();

music->setMedia(QUrl::fromLocalFile(QCoreApplication::applicationDirPath()+QDir::toNativeSeparators(QDir::separator())+"win.mp3"));

music->setVolume(100);

music->play();

repeat=newQMediaPlaylist();

repeat->setPlaybackMode(QMediaPlaylist::Loop);

}

voidMusic::**lose\_mus**(){

music3=newQMediaPlayer();

music3->setMedia(QUrl::fromLocalFile(QCoreApplication::applicationDirPath()+QDir::toNativeSeparators(QDir::separator())+"lose.mp3"));

music3->setVolume(100);

music3->play();

repeat=newQMediaPlaylist();

repeat->setPlaybackMode(QMediaPlaylist::Loop);

}

Singleton.h

#ifndefSINGLETON\_H

#defineSINGLETON\_H

class**Singleton**

{

public:

staticSingleton\***GetInstance**();

int**size\_dot**();//размеряблока/1йчастизмеи

int**width**();//ширинаполя

int**heigth**();//высотаполя

int**all\_dots**();//максимальноеколичествовозможныхточекнадоске(900\*900)/(30\*30)

int**rand\_pos**();//длявычисленияслучайнойпозициияблока

private:

staticSingleton\*instance;

};

#endif//SINGLETON\_H

Singleton.cpp

#include"singleton.h"

Singleton\*Singleton::instance=0;

Singleton\*Singleton::**GetInstance**(){

if(instance==0)

instance=newSingleton;

returninstance;

}

//размеригровогоэлемена

intSingleton::**size\_dot**(){

return30;

}

//ширинаигровогополя

intSingleton::**width**(){

return900;

}

//высотаигровогополя

intSingleton::**heigth**(){

return900;

}

//максимальноекол-воэлементовнаполе

intSingleton::**all\_dots**(){

return900;

}

//позицияэлементадлярандома

intSingleton::**rand\_pos**(){

return29;

}

Winwindow.h

#ifndefWINWINDOW\_H

#defineWINWINDOW\_H

#include<QWidget>

#include<QProcess>

#include<QApplication>

#include<QDir>

#include"music.h"

namespace**Ui**{

class**winWindow**;

}

class**winWindow**:publicQWidget

{

Q\_OBJECT

public:

explicit**winWindow**(QWidget\*parent=nullptr);

~***winWindow***();

privateslots:

void**on\_Exit\_clicked**();//выходизпроигрышногоокна

signals:

void**SnakeWindow**();//сигналнавозврат

private:

Ui::winWindow\*ui;

Music\*mus=newMusic;//музыка

};

#endif//WINWINDOW\_H

Winwindow.cpp

#include"winwindow.h"

#include"ui\_winwindow.h"

winWindow::**winWindow**(QWidget\*parent):

QWidget(*parent*),

ui(newUi::winWindow)

{

ui->setupUi(this);

//установкафоновогоизображения

QPixmaptrava(QCoreApplication::applicationDirPath()+QDir::toNativeSeparators(QDir::separator())+"trava.png");

trava=trava.scaled(this->size(),Qt::IgnoreAspectRatio);

QPalettepalette;

palette.setBrush(QPalette::Background,trava);

this->setPalette(palette);

}

winWindow::~***winWindow***()

{

deleteui;

deletemus;

}

voidwinWindow::**on\_Exit\_clicked**()

{

this->close();//закрываемокно

mus->main\_mus();//воспроизведениемузыкиглавногоменю

//QProcess::startDetached(QApplication::applicationFilePath(),QStringList(),QApplication::applicationDirPath());

emitSnakeWindow();//сигналнавозвратвпредыдущееокно

}

Snake.h

#ifndefSNAKE\_H

#defineSNAKE\_H

#include<QMainWindow>

#include<QKeyEvent>

#include<QThread>

#include<QCoreApplication>

#include<QPainter>

#include<QTime>

#include<QDir>

#include<QTimer>

#include<QRandomGenerator>

#include"levelwindow.h"

#include"gameoverwindow.h"

#include"winwindow.h"

#include"music.h"

#include"images.h"

#include"singleton.h"

namespace**Ui**{

class**Snake**;

}

class**Snake**:publicQMainWindow

{

Q\_OBJECT

public:

explicit**Snake**(LevelWindow\*,QWidget\*parent=nullptr);

~***Snake***();

protected:

void***paintEvent***(QPaintEvent\*);//отрисовкаигровыхэлементов

void***timerEvent***(QTimerEvent\*);//таймервигре

void***keyPressEvent***(QKeyEvent\*);//управлениесклавиатуры

publicslots:

int**get\_apple**();//кол-вояблок

void**addLevel**(LevelWindow\*lvl);//выборуровнясложности

//установкаяблок

void**locateApple**();

void**newlocateApple**();

void**newlocateApple2**();

void**newlocateApple3**();

signals:

void**firstWindow**();//сигналнавозвратвглавноеменю

private:

GameOverWindow\*over;//окнопроигрыша

winWindow\*win;//окнопобеды

QTimer\*timer;//включениетаймера

Musicm;//загрузкамузыки

Singletons;//использованиеконстантспомощьюпаттерна

Imagesimg;//загрузкаигровыхизображений

LevelWindow\*l;//уровеньсложности

Ui::Snake\*ui;

inttimerId;//скоростьпередвиженияобъектов

intdots;//частизмеи

booldraw=true;//отрисовкаобъектов,покавсеяблокинесъедены

//координатыяблок

intapple\_x;

intapple\_y;

intapple2\_x;

intapple2\_y;

intapple3\_x;

intapple3\_y;

intapple4\_x;

intapple4\_y;

//координатынорки

intnora\_x;

intnora\_y;

//координатымячей

intball\_x;

intball\_y;

intball2\_x;

intball2\_y;

intball3\_x;

intball3\_y;

intball4\_x;

intball4\_y;

//координатыполя

intx[900];

inty[900];

//направлениядвижениязмеи/мячей

boolleftDirection;

boolrightDirection;

boolupDirection;

booldownDirection;

boolinGame;//поканеконецигры

//проверканасъеденноеяблоко

void**checkApple**();

void**checkBall**();

void**checkCollision**();

void**Nora**();//попаданиевнорку

//движениемячей

void**move**();

void**move\_ball**();

void**move\_ball2**();

void**move\_ball3**();

void**move\_ball4**();

void**initGame**();//инициализация(начальнаяустановкавсехэлементов)

void**doDrawing**();//отрисовкаигровыхобъектов

void**gameOver**();//конецигры

void**locateNora**();//установканоркинаполе

//установкамячейнаполе

void**locateBall**();

void**locateBall2**();

void**locateBall3**();

void**locateBall4**();

};

#endif//SNAKE\_H

Snake.cpp

#include"snake.h"

#include"ui\_snake.h"

#include"mainwindow.h"

Snake::**Snake**(LevelWindow\*lvl,QWidget\*parent):

QMainWindow(*parent*),

ui(newUi::Snake)

{

ui->setupUi(this);

this->addLevel(*lvl*);//выбранныйуровеньсложности

//установкафоновогоизображения

QPixmaptrava(QCoreApplication::applicationDirPath()+QDir::toNativeSeparators(QDir::separator())+"trava.png");

trava=trava.scaled(this->size(),Qt::IgnoreAspectRatio);

QPalettepalette;

palette.setBrush(QPalette::Background,trava);

this->setPalette(palette);

//окнопроигрыша

over=newGameOverWindow();

connect(over,&GameOverWindow::SnakeWindow,this,&Snake::show);

//окновыигрыша

win=newwinWindow();

connect(win,&winWindow::SnakeWindow,this,&Snake::show);

//напрвлениядвижения

leftDirection=false;

rightDirection=false;

upDirection=false;

downDirection=false;

inGame=true;//игранеокончена

resize(s.GetInstance()->width(),s.GetInstance()->heigth());//установкарразмераигровогополя

img.load\_all();//загрузкаигровыхизображений

initGame();//инициализая,установкананачальныепозицииэлементов

timer=newQTimer(this);//таймердляобновленияпозицийяблок

connect(timer,SIGNAL(timeout()),this,SLOT(newlocateApple()));

connect(timer,SIGNAL(timeout()),this,SLOT(newlocateApple2()));

connect(timer,SIGNAL(timeout()),this,SLOT(newlocateApple3()));

connect(timer,SIGNAL(timeout()),this,SLOT(locateApple()));

timer->start(5000);//сменапозицийкаждые5сек

ui->label\_2->setText(QString::number(get\_apple()));//оставшеесякол-вояблок

m.play\_mus();//воспроизведениемузыкивовремяигры

}

//уровеньсложности

voidSnake::**addLevel**(LevelWindow\*lvl){

this->l=lvl;

}

Snake::~***Snake***()

{

deleteui;

deleteover;

deletewin;

deletel;

}

voidSnake::**initGame**(){

dots=3;//изначальныйразмерзмеюки

for(intz=0;z<dots;z++){//позицияначальнаязмеюки

x[z]=150-z\*60;

y[z]=150;

}

//установкимячейияблоквначалеигры

locateBall();

locateBall2();

locateBall3();

locateBall4();

locateApple();

newlocateApple();

newlocateApple2();

newlocateApple3();

timerId=startTimer(l->get\_speed\_snake());//скоростьобъектов

}

voidSnake::***paintEvent***(QPaintEvent\*e){

Q\_UNUSED(e);

doDrawing();//отрисовка

}

intSnake::**get\_apple**(){

returnl->count\_apple;//кол-вояблок

}

voidSnake::**doDrawing**(){

QPainterqp(this);

if(inGame){//покаигранеокончена

if(get\_apple()==0){//еслияблокизакончились

timer->stop();//остановкатаймера

draw=false;//яблокинерисуются

qp.drawImage(nora\_x,nora\_y,img.nora);//отрисовканорки

locateNora();//установканоркинаигровомполе

}

if(draw==true){//покаяблокинесъедены,отрисовка

qp.drawImage(apple\_x,apple\_y,img.apple);

qp.drawImage(apple2\_x,apple2\_y,img.apple);

qp.drawImage(apple3\_x,apple3\_y,img.apple);

qp.drawImage(apple4\_x,apple4\_y,img.apple);

}

//отрисовкамячей

qp.drawImage(ball\_x,ball\_y,img.ball);

qp.drawImage(ball2\_x,ball2\_y,img.ball);

qp.drawImage(ball3\_x,ball3\_y,img.ball);

qp.drawImage(ball4\_x,ball4\_y,img.ball);

//отрисовказмеюки

for(intz=0;z<dots;z++){

if(z==0){//вначалеотрисовкаголовы

qp.drawImage(x[z],y[z],img.head);

}else{//потомтуловища

qp.drawImage(x[z],y[z],img.dot);

}

}

}else{

this->close();//закрытиеигровогоокна

}

}

//конецигры

voidSnake::**gameOver**(){

emitfirstWindow();//вызовглавногоменю

over->setFixedSize(800,900);//установкаразмераокна

over->show();//открытияокнапроигрыша

over->setWindowTitle("Проигрыш");

this->close();//закрытиеигровогоокна

m.play\_stop();//остановкавоспроизведенияигровоймузыки

m.lose\_mus();//включениямузыкиприпоражении

}

voidSnake::**checkApple**(){

if(draw==true){//еслиневсеяблокисъедены

if((x[0]==apple\_x)&&(y[0]==apple\_y)){//еслиМайксъеляблоко

dots++;//увеличениедлиныМайка

l->count\_apple--;//уменьшениекол-ваяблок

ui->label\_2->setText(QString::number(get\_apple()));//отображениекол\_ваяблок

locateApple();//установкановогояблока

}

if((x[0]==apple2\_x)&&(y[0]==apple2\_y)){

dots++;

l->count\_apple--;

ui->label\_2->setText(QString::number(get\_apple()));

newlocateApple();

}

if((x[0]==apple3\_x)&&(y[0]==apple3\_y)){

dots++;

l->count\_apple--;

ui->label\_2->setText(QString::number(get\_apple()));

newlocateApple2();

}

if((x[0]==apple4\_x)&&(y[0]==apple4\_y)){

dots++;

l->count\_apple--;

ui->label\_2->setText(QString::number(get\_apple()));

newlocateApple3();

}

}else{

Nora();//иначеустановканаполенорки

}

}

voidSnake::**checkBall**(){

if((x[0]==ball\_x)&&(y[0]==ball\_y)){//есливголовуМайкаприлетелмяч

inGame=false;//конецигры

gameOver();//открытиеокнапроигрыша

}

if((x[0]==ball2\_x)&&(y[0]==ball2\_y)){

inGame=false;

gameOver();

}

if((x[0]==ball3\_x)&&(y[0]==ball3\_y)){

inGame=false;

gameOver();

}

if((x[0]==ball4\_x)&&(y[0]==ball4\_y)){

inGame=false;

gameOver();

}

}

voidSnake::**Nora**(){

if((x[0]==nora\_x)&&(y[0]==nora\_y)){//еслимайкзалезвнорку

inGame=false;//конецигры

emitfirstWindow();//открытиеокнаглавногоменю

this->close();//закрытиеигровогоокна

win->setFixedSize(800,900);//фиксированныйразмерокна

win->show();//открытиепобедногоокна

win->setWindowTitle("Победа");

m.win\_mus();//включениемузыкипобеды

m.play\_stop();//выключениеигровоймузыки

}

}

voidSnake::**move\_ball**(){

//еслизмеюкаползетвлевоиливниз

if((downDirection||leftDirection)&&ball\_x<s.GetInstance()->width()-390){

ball\_x+=s.GetInstance()->size\_dot();//мячдвижетсявправо

}

//еслизмеюкаползетвправоиливверх

if((upDirection||rightDirection)&&ball\_x>300){

ball\_x-=s.GetInstance()->size\_dot();//мячдвижетсявлево

}

}

voidSnake::**move\_ball2**(){

if((downDirection||leftDirection)&&ball2\_x<s.GetInstance()->width()-390){

ball2\_x+=s.GetInstance()->size\_dot();

}

if((upDirection||rightDirection)&&ball2\_x>300){

ball2\_x-=s.GetInstance()->size\_dot();

}

}

voidSnake::**move\_ball3**(){

if((downDirection||leftDirection)&&ball3\_y>0){

ball3\_y-=s.GetInstance()->size\_dot();

}

if((upDirection||rightDirection)&&ball3\_y<s.GetInstance()->heigth()-30){

ball3\_y+=s.GetInstance()->size\_dot();

}

}

voidSnake::**move\_ball4**(){

if((downDirection||leftDirection)&&ball4\_y>0){

ball4\_y-=s.GetInstance()->size\_dot();

}

if((upDirection||rightDirection)&&ball4\_y<s.GetInstance()->heigth()-30){

ball4\_y+=s.GetInstance()->size\_dot();

}

}

voidSnake::**move**(){

//движениевсейзмеюкивследзаголовой

for(intz=dots;z>0;z--){

x[z]=x[(z-1)];

y[z]=y[(z-1)];

}

//влево

if(leftDirection){

x[0]-=s.GetInstance()->size\_dot();

}

//вправо

if(rightDirection){

x[0]+=s.GetInstance()->size\_dot();

}

//вверх

if(upDirection){

y[0]-=s.GetInstance()->size\_dot();

}

//вниз

if(downDirection){

y[0]+=s.GetInstance()->size\_dot();

}

}

voidSnake::**checkCollision**(){

//еслизмеюкаврезаласьсамавсебя,токонецигры

for(intz=dots;z>0;z--){

if((z>4)&&(x[0]==x[z])&&(y[0]==y[z])){

inGame=false;

gameOver();

}

}

//еслизмеюкавышлазапределыигровогополя,токонецигры

if(y[0]>=s.GetInstance()->heigth()){

inGame=false;

gameOver();

}

if(y[0]<0){

inGame=false;

gameOver();

}

if(x[0]>=s.GetInstance()->width()){

inGame=false;

gameOver();

}

if(x[0]<0){

inGame=false;

gameOver();

}

//есликонецигры,тоостановкадвиженияобъектов

if(!inGame){

killTimer(timerId);

}

}

voidSnake::**locateNora**(){

//установканоркинаигровомполе

nora\_x=(10\*s.GetInstance()->size\_dot());

nora\_y=(1\*s.GetInstance()->size\_dot());

}

//установкаяблокчерезрандомнаигровомполе

voidSnake::**locateApple**(){

QTimetime=QTime::currentTime();

QRandomGeneratorqrand((uint)time.msec());

intr=qrand()%s.GetInstance()->rand\_pos();

apple\_x=(r\*s.GetInstance()->size\_dot());

r=qrand()%s.GetInstance()->rand\_pos();

apple\_y=(r\*s.GetInstance()->size\_dot());

}

voidSnake::**newlocateApple**(){

QTimetime=QTime::currentTime();

QRandomGeneratorqrand((uint)time.msec());

intr=qrand()%s.GetInstance()->rand\_pos()-3;

apple2\_x=(r\*s.GetInstance()->size\_dot());

r=qrand()%s.GetInstance()->rand\_pos()+6;

apple2\_y=(r\*s.GetInstance()->size\_dot());

}

voidSnake::**newlocateApple2**(){

QTimetime=QTime::currentTime();

QRandomGeneratorqrand((uint)time.msec());

intr=qrand()%s.GetInstance()->rand\_pos()-5;

apple3\_x=(r\*s.GetInstance()->size\_dot());

r=qrand()%s.GetInstance()->rand\_pos()-6;

apple3\_y=(r\*s.GetInstance()->size\_dot());

}

voidSnake::**newlocateApple3**(){

QTimetime=QTime::currentTime();

QRandomGeneratorqrand((uint)time.msec());

intr=qrand()%s.GetInstance()->rand\_pos()+6;

apple4\_x=(r\*s.GetInstance()->size\_dot());

r=qrand()%s.GetInstance()->rand\_pos()+4;

apple4\_y=(r\*s.GetInstance()->size\_dot());

}

//установкамячейнаигровомполе

voidSnake::**locateBall**(){

ball\_x=(10\*s.GetInstance()->size\_dot());

ball\_y=(10\*s.GetInstance()->size\_dot());

}

voidSnake::**locateBall2**(){

ball2\_x=(10\*s.GetInstance()->size\_dot());

ball2\_y=(20\*s.GetInstance()->size\_dot());

}

voidSnake::**locateBall3**(){

ball3\_x=(9\*s.GetInstance()->size\_dot());

ball3\_y=(10\*s.GetInstance()->size\_dot());

}

voidSnake::**locateBall4**(){

ball4\_x=(18\*s.GetInstance()->size\_dot());

ball4\_y=(10\*s.GetInstance()->size\_dot());

}

voidSnake::***timerEvent***(QTimerEvent\*e){

Q\_UNUSED(e);

if(inGame){//покаигранеокончена

//проверканасъеденныеяблоки

checkApple();

checkBall();

checkCollision();

//движениямячей

move();

move\_ball();

move\_ball2();

move\_ball3();

move\_ball4();

//проверканапопаданиевнорку

Nora();

}

repaint();//перерисовкаобъектовпридвижении

}

voidSnake::***keyPressEvent***(QKeyEvent\*e){

intkey=e->key();

//движениевлево,еслизмеюканедвижетсявправопонажатиюнаклаве<-

if((key==Qt::Key\_Left)&&(!rightDirection)){

leftDirection=true;

upDirection=false;

downDirection=false;

}

//движениевправо,еслизмеюканедвижетсявлевопонажатиюнаклаве->

if((key==Qt::Key\_Right)&&(!leftDirection)){

rightDirection=true;

upDirection=false;

downDirection=false;

}

//движениевверх,еслизмеюканедвижетсявниз

if((key==Qt::Key\_Up)&&(!downDirection)){

upDirection=true;

rightDirection=false;

leftDirection=false;

/\*img.head=img.head.transformed([](QPointcenter){

QMatrixmatrix;

matrix.translate(center.x(),center.y());

matrix.rotate(-180);

returnmatrix;

}(img.head.rect().center()));\*/

}

//движениевниз,еслизмеюканедвижетсявверх

if((key==Qt::Key\_Down)&&(!upDirection)){

downDirection=true;

rightDirection=false;

leftDirection=false;

}

QWidget::keyPressEvent(*e*);

}

Main.cpp

#include"mainwindow.h"

#include<QApplication>

#include"snake.h"

#include"loading.h"

intmain(intargc,char\*argv[])

{

QApplicationa(*argc*,*argv*);

//загрузочнаязаставка

Loadingl;

QSplashScreensplash(QPixmap(QCoreApplication::applicationDirPath()+QDir::toNativeSeparators(QDir::separator())+"splash.png"));

splash.show();

l.load(*&splash*);

//менюигры

MainWindoww;

w.setWindowTitle("Главноеменю");

w.setFixedSize(800,900);

w.show();

splash.close();

returna.exec();

}