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		М.Р.Назаров
	подпись, дата	инициалы, фамилия

Санкт-Петербург 2021

Оглавление

1. **Введение**

Информационная система – это взаимосвязанная совокупность средств, методов и персона ла, используемых для хранения, обработки и выдачи информации для достижения цели уп равления.

В современных условиях основным техническим средством обработки информации является персональный компьютер. Большинство современных информационных систем преобразуют не информацию, а данными, поэтому их называют системами обработки данных.

В моём случае, я разрабатываю автоматизированную информационную систему «Snake» на языке программирования C++ на базе IDE QT Creator. Эта простая программа является компьютерной игрой с главной целью и «фейлстейтом».

2. **Задание на курсовое проектирование**

Вариант 25:
Игра «Snake». Правила игры: Чтобы выиграть в Snake, вам нужно съесть все яблоки в комнате. С каждым съеденным яблоком вы становитесь длиннее. Если вы врежетесь в стену, вы умрете. Если вы врежетесь в себя, вы умрете. После того, как вы съели яблоко, новое яблоко появляется на экране. Если вы умираете, то ваш счет (количество съеденных яблок) появляется на экране. Игру можно перезапустить после смерти.

3. **Постановка задачи**

3.1. **Анализ предметной области**

Предметной областью курсового проекта является информационная система игры «Snake». Система должна реализовывать функции управления игровым персонажем «Змейкой». «Змейка» должна уметь есть яблоки, которые увеличивают общую длину персонажа. Новое яблоко должно появляться разу после того, как съедено предыдущее. Игры продолжается, пока не достигнут «фейлстэйт» (состояние поражения): «Змейка» сталкивается с собой или с границей экрана. При достижении «фейлстейта» должна быть возможность перезапустить игру.

3.2. **Формулировка технического задания**

В результате анализа предметной области можно сформулировать следующие функциональные требования:

Программа должна быть реализована на языке C++ в виде приложения с графическим интерфейсом. Должны быть реализованными следующие основные функции:

1. Отрисовка основных объектов игры на экране (игровое окно, Яблоко, Змейка (голова и тело), экран конца игры) и обновление экрана;
2. Управление игровым персонажем (Змейкой);
3. Перемещение игрового персонажа (Змейки);
4. Перемещение яблока;
5. Подсчет очков;
6. Увеличение длины игрового персонажа (Змейки);

7. Окончание игры при достижении фейлстейта (столкновение головы Змейки с ее телом, столкновение Змейки с границами окна) и вывод экрана конца игры с количеством очков игрока;
8. Перезапуск игры из экрана конца игры.

3.2.1 Отрисовка основных объектов игры на экране

Программа должна реализовывать следующие функции:

- Вывод окна приложения;
- Прорисовка изображения яблока по указанным координатам (относительно окна приложения);
- Прорисовка изображения головы по указанным координатам (относительно окна приложения);
- Прорисовка изображений тела по указанным координатам (относительно окна приложения);
- Обновление экрана приложения с определенной частотой.

3.2.2 Управление игровым персонажем (Змейкой)

Программа должна реализовывать следующие функции:

- Получение ввода пользователя через стрелки клавиатуры и пробел;
- Изменение направление движения Змейки в соответствии вводу пользователя через стрелки клавиатуры.

3.2.3 Перемещение игрового персонажа (Змейки)

Программа должна реализовывать следующие функции:

- Перемещение Змейки в зависимости от текущего направления движения.

3.2.4 Перемещение яблока

Программа должна реализовывать следующие функции:

- Перемещение яблока на экране при его столкновении со Змейкой.

3.2.5 Подсчет очков

Программа должна реализовывать следующие функции:

- Вычисление разницы между конечным числом сегментов тела и начальным;
- Вывод разницы на экран конца игры.

3.2.6 Увеличение длины игрового персонажа (Змейки)

Программа должна реализовывать следующие функции:

- Добавление еще одного сегмента тела к Змейке при столкновении Змейки и яблока.

3.2.7 Окончание игры при достижении фейлстейта и вывод экрана конца игры с количеством очков игрока

Программа должна реализовывать следующие функции:

- Если Змейка столкнулась сама с собой или с границей экрана, вывести экран конца игры с количеством полученных игроком очков.

3.2.8 Перезапуск игры из экрана конца игры.

Программа должна реализовывать следующие функции:

- Перезапуск игрового процесса, если игра уже окончена и игрок нажал пробел.

4. Проектирование классов

4.1. Разработанные классы

Классы сущностей:

- Apple – класс яблока;
- Head – класс головы Змейки;
- Point – класс точки (тела/сегмента Змейки).

Интерфейсные классы:

- DrawManager – класс отрисовки всех графических элементов и их обновления;
- GameOverScreen – класс экрана конца игры;

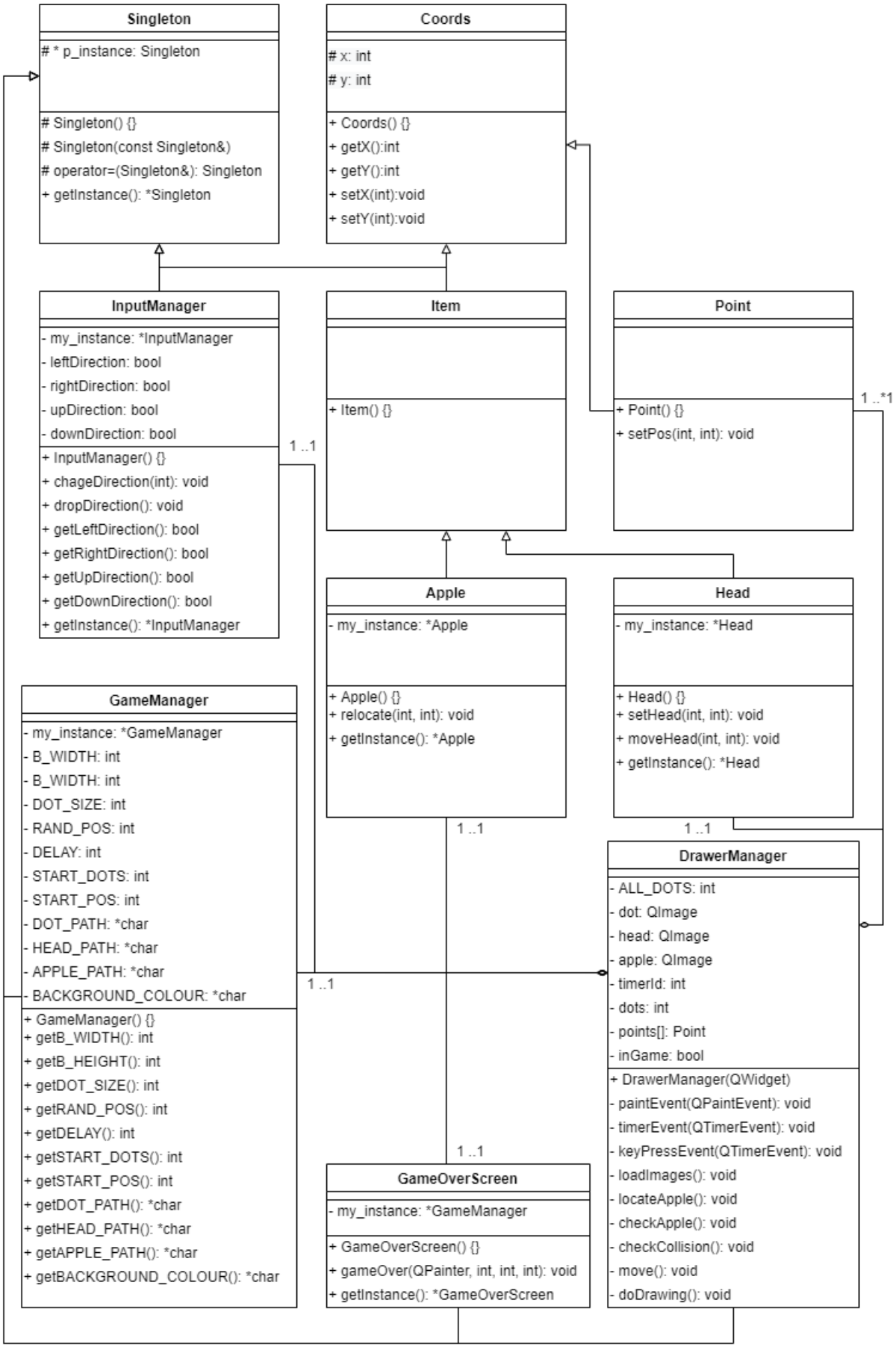
Классы управления:

- InputManager – класс регистрации ввода игрока;
- GameManager – класс параметров игрового процесса.

Служебные классы:

- Singleton – класс паттерна Singleton;
- Item – базовый класс Head и Apple;
- Coords – класс координат относительно экрана приложения.

4.2. Диаграмма классов



5. Разработка приложения

5.1. Разработка интерфейса приложения

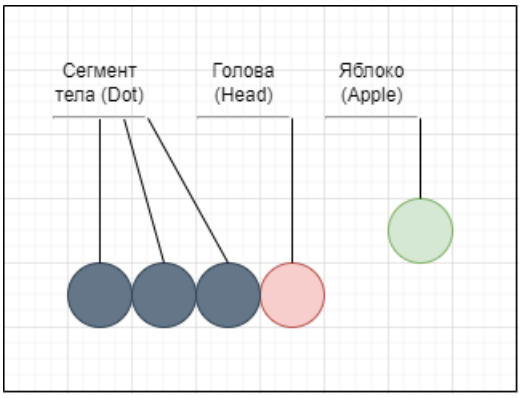


Рис. 5.1.1. Интерфейс игрового процесса

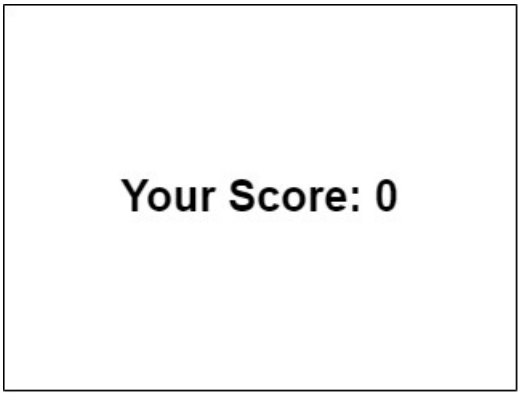


Рис. 5.1.2. Интерфейс экрана конца игры

5.2. Реализация классов

Реализация класса Apple:

```
#ifndef APPLE_H
#define APPLE_H

#include "item.h"

//Класс яблока. Отвечает за его расположение и релокацию
class Apple : public Item
{
public:
   //Станадартный пустой конструктор
   Apple() {};
   //Переместить яблоко в указанные координаты
   void relocate(int,int);
   //Получить ссылку на экземпляр яблока
   static Apple * getInstance() {
```

```
    if(!my_instance)
      my_instance = new Apple();
    return my_instance;
  }

private:
  //Ссылка на единственный экземляр яблока
  static Apple * my_instance;
};

#endif // APPLE_H
```

Реализация класса Head:

```
#ifndef HEAD_H
#define HEAD_H

#include "item.h"

//Класс головы змеи. Отвечает за расположение активного участка змеи
class Head : public Item
{
public:
  //Стандартный пустой конструктор
  Head(){}
  //Установить положение головы на указанные координаты x и y
  void setHead(int,int);
  //Сместить положение головы на указанные значения
  void moveHead(int,int);
  //Получить ссылку на экземпляр головы
  static Head * getInstance() {
    if(!my_instance)
      my_instance = new Head();
    return my_instance;
  }

private:
  //Ссылка на единственный экземляр головы
  static Head * my_instance;
};

#endif // HEAD_H
```

Реализация класса Point:

```
#ifndef POINT_H
#define POINT_H

#include "coords.h"


//Класс точки (тела змеи). Отвечает за расположение точки на экране
class Point : public Coords
```

```
{
public:
   //Стандартный пустой конструктор
   Point(){}
   //Установить положение точки по указанным координатам
   void setPos(int,int);
};

#endif // POINT_H
```

Реализация класса DrawManager:

```
#pragma once
#include <QWidget>
#include <QKeyEvent>
#include "head.h"
#include "apple.h"
#include "point.h"
#include "inputmanager.h"
#include "gameoverscreen.h"
#include "gamemanager.h"

//Класс отрисовки и контроля игры.
class DrawerManager : public QWidget, public Singleton {
public:
   //Конструктор
   DrawerManager(QWidget *parent = 0);

private:
   //Максимальное количество точек, доступных игроку
   static const int ALL_DOTS = 900;
   //Изображение точки (тела змеи)
   QImage dot;
   //Изображение головы змеи
   QImage head;
   //Изображение яблока
   QImage apple;

   //ID таймера
   int timerId;
   //Количество очков (точек тела змеи) у игрока
   int dots;

   //Массив точек тела змеи
   Point points[ALL_DOTS];
   //Идет ли еще игра?
   bool inGame;

   //Событие отрисовки
   void paintEvent(QPaintEvent *);
   //Событие таймера
   void timerEvent(QTimerEvent *);
   //Событие нажатия кнопки
   void keyPressEvent(QKeyEvent *);
   //Функция загрузки изображений из ресурсов
   void loadImages();
```

```
    //Функция старта игры
    void initGame();
    //Функция перемещения яблока
    void locateApple();
    //Функция проверки столкновения змейки с яблоком
    void checkApple();
    //Функция проверки столкновения змейки со стеной или собой
    void checkCollision();
    //Функция перемещения змейки
    void move();
    //Функция отрисовки змейки и яблока
    void doDrawing();
};
```

Реализация класса GameOverScreen:

```
#ifndef GAMEOVERSCREEN_H
#define GAMEOVERSCREEN_H
#include "QPainter"
#include "singleton.h"

//Класс экрана конца игры. Отвечает за отрисовку экрана со счетом
class GameOverScreen : public Singleton
{
public:
    //Стандартный пустой конструктор
    GameOverScreen(){}
    //Функция отрисовки экрана конца игры
    void gameOver(QPainter&, int, int, int);
    //Получить ссылку на экземпляр экрана
    static GameOverScreen * getInstance() {
        if(!my_instance)
            my_instance = new GameOverScreen();
        return my_instance;
    }

private:
    //Ссылка на единственный экземляр экрана
    static GameOverScreen * my_instance;
};
#endif // GAMEOVERSCREEN_H
```

Реализация класса InputManager:

```
#ifndef INPUTMANAGER_H
#define INPUTMANAGER_H
#include <QKeyEvent>
#include "singleton.h"
#include "QWidget"

//Класс менеджера ввода. Отвечает за направление движения змеи
class InputManager : public Singleton
{
public:
    //Стандартный пустой конструктор
    InputManager(){}
    //Изменить направление в зависимости от передаваемого ключа кнопки
```

```
   bool changeDirection(int);
   //Отчистить направление движения
   void dropDirection();
   //Получить значение левого направления
   bool getLeftDirection() const;
   //Получить значение правого направления
   bool getRightDirection() const;
   //Получить значение верхнего направления
   bool getUpDirection() const;
   //Получить значение нижнего направления
   bool getDownDirection() const;
   //Получить ссылку на экземпляр головы
   static InputManager * getInstance() {
      if(!my_instance)
         my_instance = new InputManager();
      return my_instance;
   }

private:
   //Движется ли змейка налево?
   bool leftDirection;
   //Движется ли змейка направо?
   bool rightDirection;
   //Движется ли змейка вверх?
   bool upDirection;
   //Движется ли змейка вниз?
   bool downDirection;
   //Ссылка на единственный экземляр менеджера ввода
   static InputManager * my_instance;
};

#endif // INPUTMANAGER_H
```

Реализация класса GameManager:

```
#ifndef GAMEMANAGER_H
#define GAMEMANAGER_H
#include "QString"
#include "singleton.h"

//Класс игрового менеджера. Отвечает за хранение изменяемых дизайнером значений
class GameManager : public Singleton
{
public:
   //Стандартный пустой конструктор
   GameManager(){}

   //Функция получения ширины окна
   static int getB_WIDTH();
   //Функция получения высоты окна
   static int getB_HEIGHT();
   //Функция получения размера точек
   static int getDOT_SIZE();
   //Функция получения ограничения случайной генерации
   static int getRAND_POS();
   //Функция получения задержки между кадрами
```

```
    static int getDELAY();
    //Функция получения начального количества точек у змейки
    static int getSTART_DOTS();
    //Функция получения пути до изображения точки (тела змеи)
    //Функция получения начального расположения змейки
    static int getSTART_POS();
    static const char *getDOT_PATH();
    //Функция получения пути до изображения головы
    static const char *getHEAD_PATH();
    //Функция получения пути до изображения яблока
    static const char *getAPPLE_PATH();
    //Функция получения цвета фона
    static const char *getBACKGROUND_COLOUR();

private:
    //Ширина окна
    static const int B_WIDTH = 300;
    //Высота окна
    static const int B_HEIGHT = 300;
    //Размер точки
    static const int DOT_SIZE = 10;
    //Ограничение для случайной генерации
    static const int RAND_POS = 29;
    //Задержка между кадрами. Отвечает за скорость игры
    static const int DELAY = 140;
    //Начальное количество точек у змейки
    static const int START_DOTS = 3;
    //Начальное расположение змейки
    static const int START_POS = 150;
    //Путь до изображения точки
    static constexpr const char* DOT_PATH = ":snake/images/dot.png";
    //Путь до изображения головы
    static constexpr const char* HEAD_PATH = ":snake/images/head.png";
    //Путь до изображения яблока
    static constexpr const char* APPLE_PATH = ":snake/images/apple.png";
    //Цвет фона
    static constexpr const char* BACKGROUND_COLOUR = "background-color:black;";
};

#endif // GAMEMANAGER_H
```

Реализация класса Singleton:

```
#ifndef SINGLETON_H
#define SINGLETON_H

//Базовый класс. Отвечает за создание и поддержание единтсвенного экземляра класса
class Singleton
{
public:
    //Получить ссылку на единственный экземляр класса
    static Singleton * getInstance() {
        if(!p_instance)
            p_instance = new Singleton();
        return p_instance;
    }
```

```
protected:
   //Статичная ссылка на экземпляр класса
   static Singleton * p_instance;
   //Стандартный пустой конструктор
   Singleton() {}
   //Конструктор с параметром
   Singleton( const Singleton& );
   //Переопределение оператора =
   Singleton& operator=( Singleton& );
};
#endif // SINGLETON_H
```

Реализация класса Coords:

```
#ifndef COORDS_H
#define COORDS_H

//Класс координат. Содержит координаты x и y относительно окна программы, методы их установки и получения
class Coords
{
public:
   //Станадартный пустой конструктор
   Coords(){}

   //Метод получения координаты x
   int getX() const;
   //Метод установки координаты x
   void setX(int value);

   //Метод получения координаты y
   int getY() const;
   //Метод установки координаты y
   void setY(int value);

protected:
   //Координата x
   int x;
   //Координата y
   int y;
};

#endif // COORDS_H
```

Реализация класса Item:

```
#ifndef ITEM_H
#define ITEM_H
#include "coords.h"
#include "singleton.h"

//Класс предмета. Отвечает за координаты предмета и то, что он находится в единственном экземпляре
class Item : public Singleton, public Coords
{
public:
   //стандартный пустой конструктор
   Item(){}
};
```

```
#endif // ITEM_H
```

5.3. Разработка тестового алгоритма

Рассмотрим алгоритм тестовых запусков игры.

5.3.1 Змейка сталкивается со стенкой, игрок перезапускает приложение

1. Игрок запускает приложение.
2. Приложение выводит на экран змейку и яблоко.
3. Игрок не вводит никаких значений, не нажимает никаких кнопок.
4. Змейка идет по прямой и сталкивается с границей экрана. Приложение прекращает выводить изображение Змейки и яблока. Приложение выводит сообщение «Your score: 0».
5. Игрок нажимает клавишу «Пробел».
6. Приложение выводит на экран змейку и яблоко.

5.3.2 Змейка ест два яблока и сталкивается с собой

1. Игрок запускает приложение.
2. Приложение выводит на экран змейку и яблоко.
3. Игрок направляет Змейку при помощи клавиш «вверх», «вниз», «вправо», «влево» чтобы столкнуть Змейку с первым яблоком.
4. Длина Змейки увеличивается на один сегмент.
5. Игрок направляет Змейку при помощи клавиш «вверх», «вниз», «вправо», «влево» чтобы столкнуть Змейку со вторым яблоком.
6. Длина Змейки увеличивается на один сегмент.
7. Игрок направляет Змейку при помощи клавиш «вверх», «вниз», «вправо», «влево» чтобы столкнуть Змейку с сегментом ее тела.
8. Приложение прекращает выводить изображение Змейки и яблока. Приложение выводит сообщение «Your score: 2».

6. Тестирование

Змейка сталкивается со стенкой, игрок перезапускает приложение

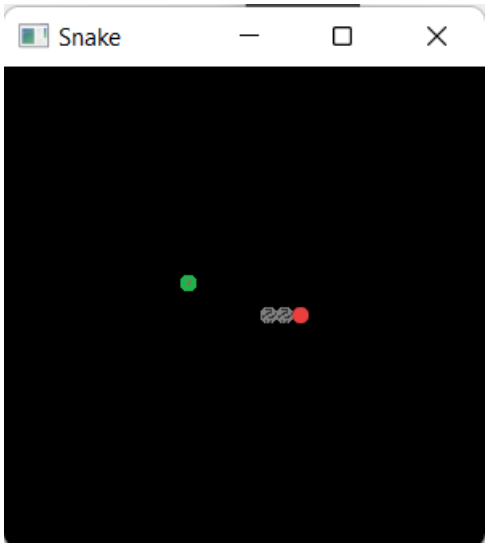


Рис. 6.1. Приложение выводит на экран змейку и яблоко.

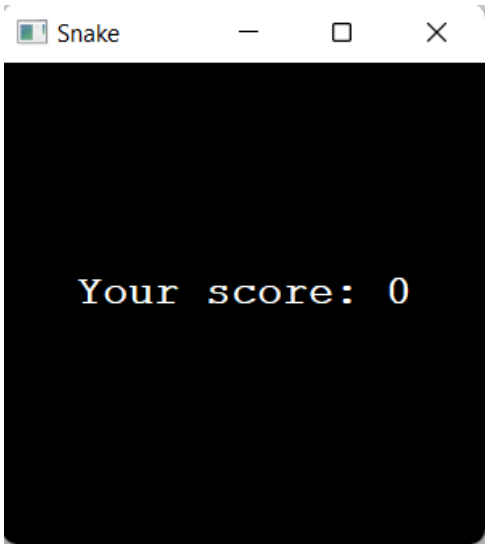


Рис. 6.2. Змейка идет по прямой и сталкивается с границей экрана. Приложение прекращает выводить изображение Змейки и яблока. Приложение выводит сообщение «Your score: 0»

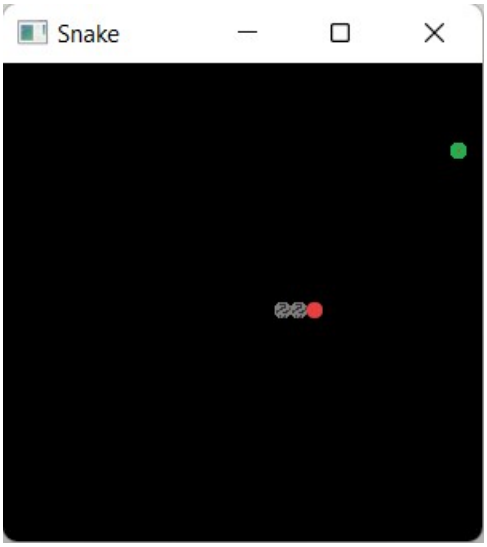


Рис. 6.3. Игрок нажимает клавишу «Пробел». Приложение выводит на экран змейку и яблоко.

Змейка ест два яблока и сталкивается с собой

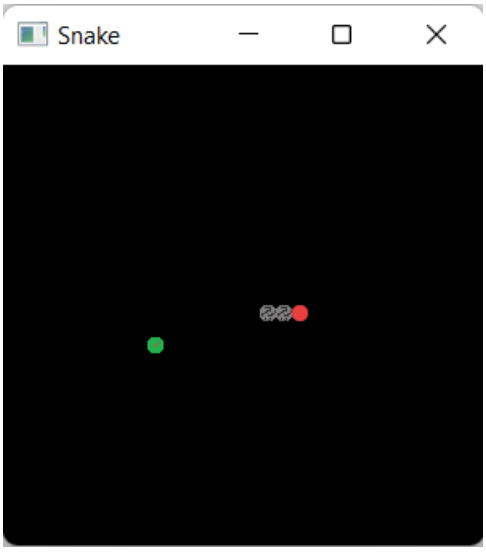


Рис. 6.4 Приложение выводит на экран змейку и яблоко.

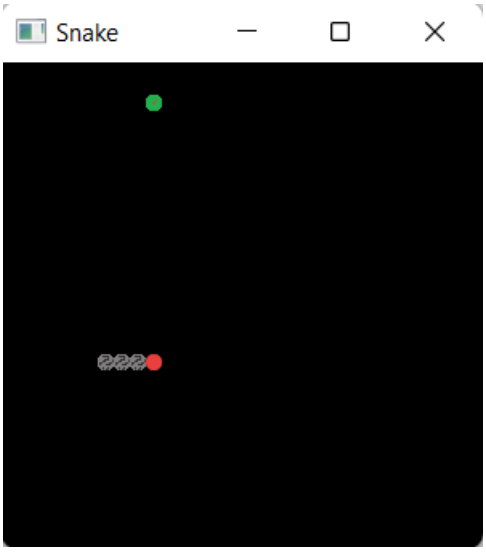


Рис. 6.5 Игрок направляет Змейку при помощи клавиш «вверх», «вниз», «вправо», «влево» чтобы столкнуть Змейку с первым яблоком. Длина Змейки увеличивается на один сегмент.

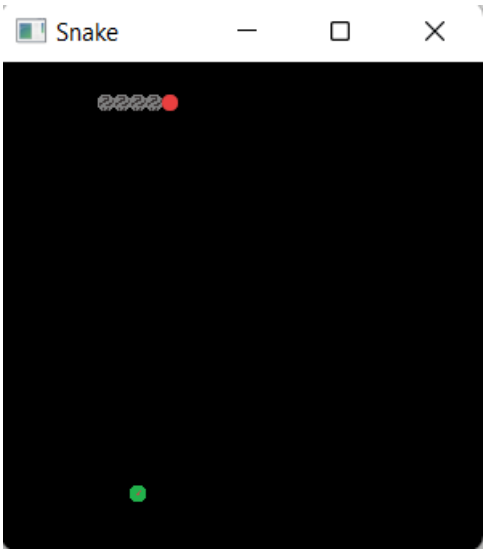


Рис. 6.6 Игрок направляет Змейку при помощи клавиш «вверх», «вниз», «вправо», «влево» чтобы столкнуть Змейку со вторым яблоком. Длина Змейки увеличивается на один сегмент.

Рис. 6.7 Игрок направляет Змейку при помощи клавиш «вверх», «вниз», «вправо», «влево» чтобы столкнуть Змейку с сегментом ее тела.

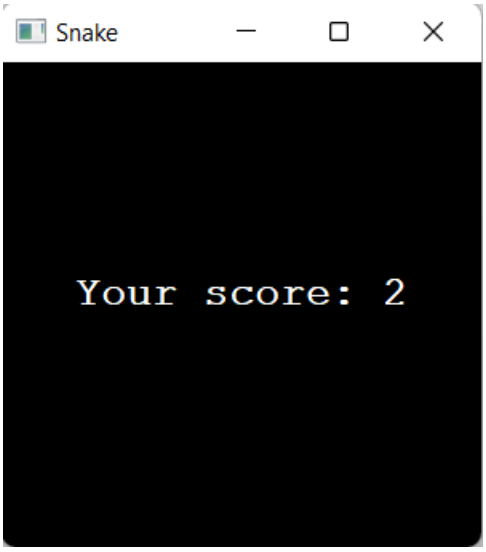


Рис. 6.8 Приложение прекращает выводить изображение Змейки и яблока. Приложение выводит сообщение «Your score: 2».

7. Заключение

Подводя итоги, можно выделить преимущества и недостатки разработанного приложения. К достоинствам можно отнести:

- Приложение предоставляет развлечение;
- Присутствуют основные элементы игры: Цель, Препятствие, Фейлстэйт, Победа;
- Существует класс для редактирования элементов игры (скорость, количество стартовых сегментов и др.);
- Присутствует соревновательный элемент в виде количества очков (подразумевает желание игрока получить наибольшее количество очков после каждой попытки).

К недостаткам можно отнести:

- Отсутствует графический интерфейс изменения параметров игры игровым дизайнером;
- Отсутствует графический интерфейс изменения параметров игры игроком;
- Слабое графическое представление игры;
- Слабые инструменты воздействия на мотивацию игрока;
- Малая вариативность и вызванная ею малое количество выбора игроком;
- Отсутствие таблицы счета игроков;
- Отсутствие системы сохранений.

8. Список использованных источников

1. Шлее М., Qt 5.10. Профессиональное программирование на C++. – СПб.: БХВ-Петербург, 2018. – 1072 с.
2. Qt Documentation:QDialog Class [Электронный ресурс]. – The Qt Company Ltd, 2021. – URL: https://doc.qt.io/qt-5/qdialog.html (дата обращения: 05.11.2021)
3. Е.О. Шумова, ОБЪЕКТНО-ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ Методические указания к выполнению курсового проекта. – ФГАОУ ВО СПбГУАП, 2021. – 17 с.

Приложение А

Singleton.h:

```
#ifndef SINGLETON_H

#define SINGLETON_H

//Базовый класс. Отвечает за создание и поддержание единтсвенного экземляра класса
class Singleton
{
public:
   //Получить ссылку на единственный экземляр класса
   static Singleton * getInstance() {
      if(!p_instance)
         p_instance = new Singleton();
      return p_instance;
   }
protected:
   //Статичная ссылка на экземпляр класса
   static Singleton * p_instance;
   //Стандартный пустой конструктор
   Singleton() {}
   //Конструктор с параметром
   Singleton( const Singleton& );
   //Переопределение оператора =
   Singleton& operator=( Singleton& );
};
#endif // SINGLETON_H
```

Singleton.cpp:

```
#include "singleton.h"


//Обнуляем ссылку на экземляр класса во избежание ошибок
Singleton* Singleton::p_instance = 0;
```

Coords.h:

```
#ifndef COORDS_H

#define COORDS_H

//Класс координат. Содержит координаты x и y относительно окна программы, методы их установки и
получения
class Coords
{
public:
   //Станадартный пустой конструктор
   Coords(){}

   //Метод получения координаты x
   int getX() const;
   //Метод установки координаты x
   void setX(int value);
```

```
    //Метод получения координаты y
    int getY() const;
    //Метод установки координаты y
    void setY(int value);

protected:
    //Координата x
    int x;
    //Координата y
    int y;
};

#endif // COORDS_H
```

Coords.cpp:

```
#include "coords.h"


//Получить x
//Не принимает значений
//Возвращает значение int
int Coords::getX() const
{
    return x;
}

//Установить x
//Принимает значение int
//Не возвращает значений
void Coords::setX(int value)
{
    x = value;
}

//Получить y
//Не принимает значений
//Возвращает значение int
int Coords::getY() const
{
    return y;
}

//Установить y
//Принимает значение int
//Не возвращает значений
void Coords::setY(int value)
{
    y = value;
}
```

Item.h:

```
#ifndef ITEM_H

#define ITEM_H
#include "coords.h"
#include "singleton.h"
```

```
//Класс предмета. Отвечает за координаты предмета и то, что он находится в единственном экземпляре
class Item : public Singleton, public Coords
{
public:
   //стандартный пустой конструктор
   Item(){}
};

#endif // ITEM_H
```

Item.cpp:

```
#include "item.h"
```

Apple.h:

```
#ifndef APPLE_H

#define APPLE_H

#include "item.h"

//Класс яблока. Отвечает за его расположение и релокацию
class Apple : public Item
{
public:
   //Станадартный пустой конструктор
   Apple() {};
   //Переместить яблоко в указанные координаты
   void relocate(int,int);
   //Получить ссылку на экземпляр яблока
   static Apple * getInstance() {
      if(!my_instance)
         my_instance = new Apple();
      return my_instance;
   }

private:
   //Ссылка на единственный экземляр яблока
   static Apple * my_instance;
};

#endif // APPLE_H
```

Apple.cpp:

```
#include "apple.h"


//Обнуляем ссылку на экземляр класса во избежание ошибок
Apple* Apple::my_instance = 0;

//Переместить яблоко в указанные координаты
//Принимает значения int
//Не возвращает значений
void Apple::relocate(int newx, int newy)
{
   setX(newx);
   setY(newy);
}
```

Point.h:

```
#ifndef POINT_H

#define POINT_H

#include "coords.h"


//Класс точки (тела змеи). Отвечает за расположение точки на экране
class Point : public Coords
{
public:
   //Стандартный пустой конструктор
   Point(){}
   //Установить положение точки по указанным координатам
   void setPos(int,int);
};

#endif // POINT_H
```

Point.cpp:

```
#include "point.h"


//Установить положение точки по указанным координатам
//Принимает значения int
//Не возвращает значений
void Point::setPos(int newx, int newy)
{
   setX(newx);
   setY(newy);
}
```

Head.h:

```
#ifndef HEAD_H

#define HEAD_H

#include "item.h"

//Класс головы змеи. Отвечает за расположение активного участка змеи
class Head : public Item
{
public:
   //Стандартный пустой конструктор
   Head(){}
   //Установить положение головы на указанные координаты x и y
   void setHead(int,int);
   //Сместить положение головы на указанные значения
   void moveHead(int,int);
   //Получить ссылку на экземпляр головы
   static Head * getInstance() {
      if(!my_instance)
         my_instance = new Head();
      return my_instance;
```

```
    }

private:
    //Ссылка на единственный экземляр головы
    static Head * my_instance;
};

#endif // HEAD_H
```

Head.cpp:

```
#include "head.h"


//Обнуляем ссылку на экземляр класса во избежание ошибок
Head* Head::my_instance = 0;

//Установить положение головы на указанные координаты x и y
//Принимает значения int
//Не возвращает значений
void Head::setHead(int newx, int newy)
{
    setX(newx);
    setY(newy);
}
//Сместить положение головы на указанные значения
//Принимает значения int
//Не возвращает значений
void Head::moveHead(int offsetx, int offsety)
{
    setX(getX()-offsetx);
    setY(getY()-offsety);
}
```

DrawManager.h:

```
#pragma once

#include <QWidget>
#include <QKeyEvent>
#include "head.h"
#include "apple.h"
#include "point.h"
#include "inputmanager.h"
#include "gameoverscreen.h"
#include "gamemanager.h"

//Класс отрисовки и контроля игры.
class DrawerManager : public QWidget, public Singleton {
public:
    //Конструктор
    DrawerManager(QWidget *parent = 0);

private:
    //Максимальное количество точек, доступных игроку
    static const int ALL_DOTS = 900;
    //Изображение точки (тела змеи)
    QImage dot;
    //Изображение головы змеи
```

```
    QImage head;
    //Изображение яблока
    QImage apple;

    //ID таймера
    int timerId;
    //Количество очков (точек тела змеи) у игрока
    int dots;

    //Массив точек тела змеи
    Point points[ALL_DOTS];
    //Идет ли еще игра?
    bool inGame;

    //Событие отрисовки
    void paintEvent(QPaintEvent *);
    //Событие таймера
    void timerEvent(QTimerEvent *);
    //Событие нажатия кнопки
    void keyPressEvent(QKeyEvent *);
    //Функция загрузки изображений из ресурсов
    void loadImages();
    //Функция старта игры
    void initGame();
    //Функция перемещения яблока
    void locateApple();
    //Функция проверки столкновения змейки с яблоком
    void checkApple();
    //Функция проверки столкновения змейки со стеной или собой
    void checkCollision();
    //Функция перемещения змейки
    void move();
    //Функция отрисовки змейки и яблока
    void doDrawing();
};
```

DrawManager.cpp:

```
#include <QPainter>

#include <QRandomGenerator>
#include "drawermanager.h"

//Конструктор класса. Создает экземляр класса на основе базового класса QWidget и инициализирует
игру
//Принимает ссылку на QWidget
//Не возвращает значений
DrawerManager::DrawerManager(QWidget *parent) : QWidget(parent) {
    //Установить цвет фона
    setStyleSheet(GameManager::getBACKGROUND_COLOUR());
    //Изменить размер окна
    resize(GameManager::getB_WIDTH(), GameManager::getB_HEIGHT());
    //Загрузить изображения
    loadImages();
    //Инициализировать начало игры
    initGame();
```

```
}

//Событие отрисовки
//Принимает ссылку на QPainterEvent
//Не возвращает значений
void DrawerManager::paintEvent(QPaintEvent *e) {
   //Убрать неиспользуемые предупреждения
   Q_UNUSED(e);
   //Начать отрисовку объектов
   doDrawing();
}

//Событие таймера. Срабатывает каждый кадр, количетсво вызовов зависит от задержки, указанной в
GameManager.h
//Принимает ссылку на QTimerEvent
//Не возвращает значений
void DrawerManager::timerEvent(QTimerEvent *e) {
   //Убрать неиспользуемые предупреждения
   Q_UNUSED(e);

   //Если игра все еще идет
   if (inGame) {
      //Проверить, не съела ли змейка яблоко
      checkApple();
      //Проверить, не врезалась ли змейка в стену или в себя
      checkCollision();
      //Переместить змейку внутри окна
      move();
   }

   //отрисовать экран заново
   repaint();
}

//Событие нажатия кнопки. Срабатывает при нажатии кнопки
//Принимает ссылку на QKeyEvent
//Не возвращает значений
void DrawerManager::keyPressEvent(QKeyEvent *e) {

   //Получить ключ нажатой кнопки
   int key = e->key();

   //Если нажат пробел и игра закончена
   if(InputManager::getInstance()->changeDirection(key) && !inGame)
   {
      //то начать игру заново
      initGame();
   }

   //Вызвать событие нажатия кнопки в    QWidget
   QWidget::keyPressEvent(e);
}

//Функция загрузки изображений из ресурсов
//Не принимает значений
//Не возвращает значений
void DrawerManager::loadImages() {
```

```
    //Подгрузить изображения из указанных в GameManager.h путей
    dot.load(GameManager::getDOT_PATH());
    head.load(GameManager::getHEAD_PATH());
    apple.load(GameManager::getAPPLE_PATH());
}
//Функция старта игры
//Не принимает значений
//Не возвращает значений
void DrawerManager::initGame() {

    //Сделать игру активной
    inGame = true;
    //Обнулить направление движения змейки
    InputManager::getInstance()->dropDirection();

    //Сделать количество точек у игрока начальным
    dots = GameManager::getSTART_DOTS();

    //Установить начальные позиции змейки
    for (int z = 0; z < dots; z++) {
        if (z==0)
        {
            Head::getInstance()->setHead(GameManager::getSTART_POS(),GameManager::getSTART_POS());
        }
        points[z].setPos(GameManager::getSTART_POS() - z * 10, GameManager::getSTART_POS());
    }

    //переместить яблоко
    locateApple();

    //Создать таймер
    timerId = startTimer(GameManager::getDELAY());
}

//Функция перемещения яблока
//Не принимает значений
//Не возвращает значений
void DrawerManager::locateApple() {

    //Сгенерировать случайные координаты x и y
        int rx = QRandomGenerator::global()->generate() % GameManager::getRAND_POS() *
GameManager::getDOT_SIZE();
        int ry = QRandomGenerator::global()->generate() % GameManager::getRAND_POS() *
GameManager::getDOT_SIZE();
    //Переместить яблоко на случайные координаты
    Apple::getInstance()->relocate(rx,ry);
}

//Функция проверки столкновения змейки с яблоком
//Не принимает значений
//Не возвращает значений
void DrawerManager::checkApple() {

    //Если координаты головы и яблока совпадают
     if (((Head::getInstance()->getX()) == Apple::getInstance()->getX()) && (Head::getInstance()-
>getY()) == Apple::getInstance()->getY()) {
        //то добавить змейке точку
```

```
      dots++;
      //и переместить яблоко
      locateApple();
   }

   //Если количество точек змейки больше или равно максимальному
   if (dots >= ALL_DOTS)
   {
      //то завершить игру
      inGame = false;
   }
}

//Функция проверки столкновения змейки со стеной или собой
//Не принимает значений
//Не возвращает значений
void DrawerManager::checkCollision() {

   //Проверить каждую точку змейки
   for (int z = dots; z > 0; z--) {

      //Если координаты головы и точки совпадают
            if ((z > GameManager::getSTART_DOTS() + 1) && (Head::getInstance()->getX() ==
points[z].getX()) && (Head::getInstance()->getY() == points[z].getY())) {
        //то закончить игру
        inGame = false;
      }
   }

   //Если координаты головы вышли за рамки экрана
   if (Head::getInstance()->getY() >= GameManager::getB_HEIGHT() ||
        Head::getInstance()->getY() < 0 ||
        Head::getInstance()->getX() >= GameManager::getB_WIDTH()||
        Head::getInstance()->getX() < 0) {
      //то закончить игру
      inGame = false;
   }

   //Если игра закончена
   if(!inGame) {
      //то уничтожить таймер
      killTimer(timerId);
   }
}

//Функция перемещения змейки
void DrawerManager::move() {
   //Создать значения для смещения змейки
   int offsetX = 0;
   int offsetY = 0;
   //Переместить каждую точку на координату следующей точки
   for (int z = dots; z > 0; z--) {
      if(z == 1)
      {
         points[z].setPos(Head::getInstance()->getX(),Head::getInstance()->getY());
```

```
      }
      else
      {
         points[z].setPos(points[(z-1)].getX(), points[(z-1)].getY());
      }

   }

   //Если текущее направление - влево
   if (InputManager::getInstance()->getLeftDirection()) {
      //то добавить смещение влево
      offsetX += GameManager::getDOT_SIZE();
   }
   //Если текущее направление - вправо
   if (InputManager::getInstance()->getRightDirection()) {
      //то добавить смещение вправо
      offsetX -= GameManager::getDOT_SIZE();
   }
   //Если текущее направление - вверх
   if (InputManager::getInstance()->getUpDirection()) {
      //то добавить смещение вверх
      offsetY += GameManager::getDOT_SIZE();
   }
   //Если текущее направление - вниз
   if (InputManager::getInstance()->getDownDirection()) {
      //то добавить смещение вниз
      offsetY -= GameManager::getDOT_SIZE();
   }

   //Переместить голову на указанное смещение
   Head::getInstance()->moveHead(offsetX,offsetY);
}

//Функция отрисовки змейки и яблока
//Не принимает значений
//Не возвращает значений
void DrawerManager::doDrawing() {

   //Создать объект QPainter
   QPainter qp(this);

   //Если игра все еще идет
   if (inGame) {
      //то отрисовать яблоко
      qp.drawImage(Apple::getInstance()->getX(), Apple::getInstance()->getY(), apple);

      //и голову с точками
      for (int z = 0; z < dots; z++) {
         if (z == 0) {
            qp.drawImage(Head::getInstance()->getX(), Head::getInstance()->getY(), head);
         } else {
            qp.drawImage(points[z].getX(), points[z].getY(), dot);
         }
      }
   //иначе
```

```
    } else {
        //вывести экран конца игры
                    GameOverScreen::getInstance()->gameOver(qp,     height(),     width(),(dots     -
GameManager::getSTART_DOTS()));
    }
}
```

GameOverScreen.h:

```
#ifndef GAMEOVERSCREEN_H

#define GAMEOVERSCREEN_H
#include "QPainter"
#include "singleton.h"

//Класс экрана конца игры. Отвечает за отрисовку экрана со счетом
class GameOverScreen : public Singleton
{
public:
    //Стандартный пустой конструктор
    GameOverScreen(){}
    //Функция отрисовки экрана конца игры
    void gameOver(QPainter&, int, int, int);
    //Получить ссылку на экземпляр экрана
    static GameOverScreen * getInstance() {
        if(!my_instance)
            my_instance = new GameOverScreen();
        return my_instance;
    }

private:
    //Ссылка на единственный экземляр экрана
    static GameOverScreen * my_instance;
};
#endif // GAMEOVERSCREEN_H
```

GameOverScreen.cpp:

```
#include "gameoverscreen.h"


//Обнуляем ссылку на экземляр класса во избежание ошибок
GameOverScreen* GameOverScreen::my_instance = 0;

//Функция отрисовки экрана конца игры
//Получает адрес Qpainter и значения int
//Не возвращает значений
void GameOverScreen::gameOver(QPainter &qp, int height, int width, int points)
{
    //Создать сообщение
    QString message = "Your score: " + QString::number(points);
    //Создать шрифт
    QFont font("Courier", 15, QFont::DemiBold);
    //Создать метрики измерения шрифта
    QFontMetrics fm(font);
    //Узнать ширину сообщения
    int textWidth = fm.horizontalAdvance(message);

    //Установить белый цвет шрифта
```

```
    qp.setPen(QColor(Qt::white));
    //Установить ранее созданный шрифт
    qp.setFont(font);

    //Переместить указатель на середину экрана
    qp.translate(QPoint(width/2, height/2));
    //Написать сообщение
    qp.drawText(-textWidth/2, 0, message);
}
```

InputManager.h:

```
#ifndef INPUTMANAGER_H

#define INPUTMANAGER_H
#include <QKeyEvent>
#include "singleton.h"
#include "QWidget"

//Класс менеджера ввода. Отвечает за направление движения змеи
class InputManager : public Singleton
{
public:
    //Стандартный пустой конструктор
    InputManager(){}
    //Изменить направление в зависимости от передаваемого ключа кнопки
    bool changeDirection(int);
    //Отчистить направление движения
    void dropDirection();
    //Получить значение левого направления
    bool getLeftDirection() const;
    //Получить значение правого направления
    bool getRightDirection() const;
    //Получить значение верхнего направления
    bool getUpDirection() const;
    //Получить значение нижнего направления
    bool getDownDirection() const;
    //Получить ссылку на экземпляр головы
    static InputManager * getInstance() {
        if(!my_instance)
            my_instance = new InputManager();
        return my_instance;
    }

private:
    //Движется ли змейка налево?
    bool leftDirection;
    //Движется ли змейка направо?
    bool rightDirection;
    //Движется ли змейка вверх?
    bool upDirection;
    //Движется ли змейка вниз?
    bool downDirection;
    //Ссылка на единственный экземляр менеджера ввода
    static InputManager * my_instance;
};
```

```
#endif // INPUTMANAGER_H
```

InputManager.cpp:

```
#include "inputmanager.h"


//Обнуляем ссылку на экземляр класса во избежание ошибок
InputManager* InputManager::my_instance = 0;

//Изменить направление в зависимости от передаваемого ключа кнопки. Возвращает true, если нажат Пробел (Space) и false в ином случае
//Принимает значение int
//Возвращает значение bool
bool InputManager::changeDirection(int key) {
   //Если нажата клавша влево и не выбрано направление вправо
   if ((key == Qt::Key_Left) && (!rightDirection)) {
      //Сделать текущим направлением "влево"
      leftDirection = true;
      upDirection = false;
      downDirection = false;
   }
   //Если нажата клавша вправо и не выбрано направление влево
   if ((key == Qt::Key_Right) && (!leftDirection)) {
      //Сделать текущим направлением "вправо"
      rightDirection = true;
      upDirection = false;
      downDirection = false;
   }
   //Если нажата клавша вверх и не выбрано направление вниз
   if ((key == Qt::Key_Up) && (!downDirection)) {
      //Сделать текущим направлением "вверх"
      upDirection = true;
      rightDirection = false;
      leftDirection = false;
   }
   //Если нажата клавша вниз и не выбрано направление вверх
   if ((key == Qt::Key_Down) && (!upDirection)) {
      //Сделать текущим направлением "вниз"
      downDirection = true;
      rightDirection = false;
      leftDirection = false;
   }

   //Возвращает true, если нажат Пробел (Space) и false в ином случае
   if(key == Qt::Key_Any)
   {
      return true;
   }
   else
   {
      return false;
   }
}
//Отчистить направление движения
```

```
//Не принимает значений
//Не возвращает значений
void InputManager::dropDirection()
{
   leftDirection = false;
   rightDirection = true;
   upDirection = false;
   downDirection = false;

}
//Получить значение левого направления
//Не принимает значений
//Возвращает значение bool
bool InputManager::getLeftDirection() const
{
   return leftDirection;
}

//Получить значение правого направления
//Не принимает значений
//Возвращает значение bool
bool InputManager::getRightDirection() const
{
   return rightDirection;
}

//Получить значение верхнего направления
//Не принимает значений
//Возвращает значение bool
bool InputManager::getUpDirection() const
{
   return upDirection;
}

//Получить значение нижнего направления
//Не принимает значений
//Возвращает значение bool
bool InputManager::getDownDirection() const
{
   return downDirection;
}
```

GameManager.h:

```
#ifndef GAMEMANAGER_H

#define GAMEMANAGER_H
#include "QString"
#include "singleton.h"

//Класс игрового менеджера. Отвечает за хранение изменяемых дизайнером значений
class GameManager : public Singleton
{
public:
   //Стандартный пустой конструктор
   GameManager(){}

   //Функция получения ширины окна
   static int getB_WIDTH();
   //Функция получения высоты окна
   static int getB_HEIGHT();
```

```
    //Функция получения размера точек
    static int getDOT_SIZE();
    //Функция получения ограничения случайной генерации
    static int getRAND_POS();
    //Функция получения задержки между кадрами
    static int getDELAY();
    //Функция получения начального количества точек у змейки
    static int getSTART_DOTS();
    //Функция получения пути до изображения точки (тела змеи)
    //Функция получения начального расположения змейки
    static int getSTART_POS();
    static const char *getDOT_PATH();
    //Функция получения пути до изображения головы
    static const char *getHEAD_PATH();
    //Функция получения пути до изображения яблока
    static const char *getAPPLE_PATH();
    //Функция получения цвета фона
    static const char *getBACKGROUND_COLOUR();

private:
    //Ширина окна
    static const int B_WIDTH = 300;
    //Высота окна
    static const int B_HEIGHT = 300;
    //Размер точки
    static const int DOT_SIZE = 10;
    //Ограничение для случайной генерации
    static const int RAND_POS = 29;
    //Задержка между кадрами. Отвечает за скорость игры
    static const int DELAY = 140;
    //Начальное количество точек у змейки
    static const int START_DOTS = 3;
    //Начальное расположение змейки
    static const int START_POS = 150;
    //Путь до изображения точки
    static constexpr const char* DOT_PATH = ":snake/images/dot.png";
    //Путь до изображения головы
    static constexpr const char* HEAD_PATH = ":snake/images/head.png";
    //Путь до изображения яблока
    static constexpr const char* APPLE_PATH = ":snake/images/apple.png";
    //Цвет фона
    static constexpr const char* BACKGROUND_COLOUR = "background-color:black;";
};

#endif // GAMEMANAGER_H
```

GameManager.cpp:

```
#include "gamemanager.h"


//Функция получения ширины окна
//Не принимает значений
//Возвращает значение int
int GameManager::getB_WIDTH()
{
```

```
    return B_WIDTH;
}

//Функция получения высоты окна
//Не принимает значений
//Возвращает значение int
int GameManager::getB_HEIGHT()
{
    return B_HEIGHT;
}

//Функция получения размера точек
//Не принимает значений
//Возвращает значение int
int GameManager::getDOT_SIZE()
{
    return DOT_SIZE;
}

//Функция получения ограничения случайной генерации
//Не принимает значений
//Возвращает значение int
int GameManager::getRAND_POS()
{
    return RAND_POS;
}

//Функция получения задержки между кадрами
//Не принимает значений
//Возвращает значение int
int GameManager::getDELAY()
{
    return DELAY;
}

//Функция получения начального количества точек у змейки
//Не принимает значений
//Возвращает значение int
int GameManager::getSTART_DOTS()
{
    return START_DOTS;
}

//Функция получения начального расположения змейки
//Не принимает значений
//Возвращает значение int
int GameManager::getSTART_POS()
{
    return START_POS;
}

//Функция получения пути до изображения точки (тела змеи)
//Не принимает значений
//Возвращает значение char*
const char *GameManager::getDOT_PATH()
{
    return DOT_PATH;
}

//Функция получения пути до изображения головы
//Не принимает значений
//Возвращает значение char*
```

```
const char *GameManager::getHEAD_PATH()
{
   return HEAD_PATH;
}

//Функция получения пути до изображения яблока
//Не принимает значений
//Возвращает значение char*
const char *GameManager::getAPPLE_PATH()
{
   return APPLE_PATH;
}

//Функция получения цвета фона
//Не принимает значений
//Возвращает значение char*
const char *GameManager::getBACKGROUND_COLOUR()
{
   return BACKGROUND_COLOUR;
}
```