|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | | |  | | |  |
| МИНОБРНАУКИ РОССИИ | | | | | | |
| Федеральное государственное бюджетное образовательное учреждение  высшего образования  **«МИРЭА – Российский технологический университет»**  **РТУ МИРЭА** | | | | | | |
| Институт «Информационных технологий»  Кафедра «**Общей информатики**» | | | | | |
| **КУРСОВАЯ РАБОТА** | | | | | | |
| **по дисциплине «Объектно-ориентированное программирование»** | | | | | | |
|  | | | | | | |
| **Тема курсовой работы «**Компьютерная игра “Змейка“» | | | | | | |
| Студент группы ИНБО-01-18 | | | Косицын Артём Георгиевич | | | |
| Работа представлена к защите  «\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_ 2019 г. | | |  | *(подпись студента)* | | | |

|  |  |  |
| --- | --- | --- |
| Руководитель курсовой работы | | Карпов Дмитрий Анатольевич |
| «Работа допущена к защите» | |
| «\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_ 2019 г. |  |  |

Москва 2019

Оглавление

[Глава 1. Техническое задание 3](file:///C:\Users\User\Desktop\Курсовая%20работа%20-%20пример.docx#_Toc470682246)

1.1 [Введение](file:///C:\Users\User\Desktop\Курсовая%20работа%20-%20пример.docx#_Toc470682225)  [3](file:///C:\Users\User\Desktop\Курсовая%20работа%20-%20пример.docx#_Toc470682248)

[1.1.1. Наименование программы 4](file:///C:\Users\User\Desktop\Курсовая%20работа%20-%20пример.docx#_Toc470682248)

[1.1.2. Назначение и область применения 4](file:///C:\Users\User\Desktop\Курсовая%20работа%20-%20пример.docx#_Toc470682248)

1.2. Требования к программе или программному изделию [4](file:///C:\Users\User\Desktop\Курсовая%20работа%20-%20пример.docx#_Toc470682248)

[[1.2.1. Требования к функциональным характеристикам](file:///C:\Users\User\Desktop\Курсовая%20работа%20-%20пример.docx#_Toc470682229) 4](file:///C:\Users\User\Desktop\Курсовая%20работа%20-%20пример.docx#_Toc470682248)

[[1.2.2.Требования к надежности](file:///C:\Users\User\Desktop\Курсовая%20работа%20-%20пример.docx#_Toc470682230) 4](file:///C:\Users\User\Desktop\Курсовая%20работа%20-%20пример.docx#_Toc470682248)

[[1.3. [Условия эксплуатации](file:///C:\Users\User\Desktop\Курсовая%20работа%20-%20пример.docx#_Toc470682231)](file:///C:\Users\User\Desktop\Курсовая%20работа%20-%20пример.docx#_Toc470682239) 4](file:///C:\Users\User\Desktop\Курсовая%20работа%20-%20пример.docx#_Toc470682248)

[1.3.1. Требования к составу и параметрам технических средств 5](file:///C:\Users\User\Desktop\Курсовая%20работа%20-%20пример.docx#_Toc470682248)

[1.3.2. Требования к информационной и программной совместимости 5](file:///C:\Users\User\Desktop\Курсовая%20работа%20-%20пример.docx#_Toc470682248)

[[1.3.2.1. Требования к исходным кодам и языкам программирования](file:///C:\Users\User\Desktop\Курсовая%20работа%20-%20пример.docx#_Toc470682234) 5](file:///C:\Users\User\Desktop\Курсовая%20работа%20-%20пример.docx#_Toc470682248)

[[1.3.2.2.Требования к программным средствам, используемым программой](file:///C:\Users\User\Desktop\Курсовая%20работа%20-%20пример.docx#_Toc470682235) 5](file:///C:\Users\User\Desktop\Курсовая%20работа%20-%20пример.docx#_Toc470682248)

[[1.3.3. Специальные требования](file:///C:\Users\User\Desktop\Курсовая%20работа%20-%20пример.docx#_Toc470682239)  5](file:///C:\Users\User\Desktop\Курсовая%20работа%20-%20пример.docx#_Toc470682248)

[[1.4. Стадии и этапы разработки](file:///C:\Users\User\Desktop\Курсовая%20работа%20-%20пример.docx#_Toc470682239) 6](file:///C:\Users\User\Desktop\Курсовая%20работа%20-%20пример.docx#_Toc470682248)

[1.4.1. Стадии разработки 6](file:///C:\Users\User\Desktop\Курсовая%20работа%20-%20пример.docx#_Toc470682248)

[[1.5. Стадии и этапы разработки](file:///C:\Users\User\Desktop\Курсовая%20работа%20-%20пример.docx#_Toc470682239) 6](file:///C:\Users\User\Desktop\Курсовая%20работа%20-%20пример.docx#_Toc470682248)

[1.5.1. Стадии разработки 6](file:///C:\Users\User\Desktop\Курсовая%20работа%20-%20пример.docx#_Toc470682248)

[1.5.2. Этапы разработки 6](file:///C:\Users\User\Desktop\Курсовая%20работа%20-%20пример.docx#_Toc470682248)

[[1.5.3. Содержание работ по этапам](file:///C:\Users\User\Desktop\Курсовая%20работа%20-%20пример.docx#_Toc470682242) 7](file:///C:\Users\User\Desktop\Курсовая%20работа%20-%20пример.docx#_Toc470682248)

[1.6. Порядок контроля и приемки 8](file:///C:\Users\User\Desktop\Курсовая%20работа%20-%20пример.docx#_Toc470682248)

[1.6.1. Виды испытаний 8](file:///C:\Users\User\Desktop\Курсовая%20работа%20-%20пример.docx#_Toc470682248)

[1.6.2. Общие требования к приемке работы 8](file:///C:\Users\User\Desktop\Курсовая%20работа%20-%20пример.docx#_Toc470682248)

[Глава 2. Проектирование и разработка программы 9](file:///C:\Users\User\Desktop\Курсовая%20работа%20-%20пример.docx#_Toc470682246)

[2.1. Проектирование программы 9](file:///C:\Users\User\Desktop\Курсовая%20работа%20-%20пример.docx#_Toc470682248)

[2.1.1. Функциональные требования к программе 9](file:///C:\Users\User\Desktop\Курсовая%20работа%20-%20пример.docx#_Toc470682248)

[2.1.2. Проектирование классов 9](file:///C:\Users\User\Desktop\Курсовая%20работа%20-%20пример.docx#_Toc470682248)

[2.2. Разработка программы 10](file:///C:\Users\User\Desktop\Курсовая%20работа%20-%20пример.docx#_Toc470682248)

[2.2.1. Описание проектного решения 10](file:///C:\Users\User\Desktop\Курсовая%20работа%20-%20пример.docx#_Toc470682248)

[2.2.2. Тестирование программы 10](file:///C:\Users\User\Desktop\Курсовая%20работа%20-%20пример.docx#_Toc470682248)

[Глава 3. Руководство пользователя 14](file:///C:\Users\User\Desktop\Курсовая%20работа%20-%20пример.docx#_Toc470682253)

[Заключение 17](file:///C:\Users\User\Desktop\Курсовая%20работа%20-%20пример.docx#_Toc470682254)

[Список источников 17](file:///C:\Users\User\Desktop\Курсовая%20работа%20-%20пример.docx#_Toc470682255)

[Приложение 1. Исходный код программы 18](file:///C:\Users\User\Desktop\Курсовая%20работа%20-%20пример.docx#_Toc470682256)

Приложение 2. [Ресурсы 29](file:///C:\Users\User\Desktop\Курсовая%20работа%20-%20пример.docx#_Toc470682256)

**Аннотация**

УДК 004.021

Косицын А.Г. Создание игры / Курсовой проект по дисциплине «Объектно-ориентированное программирование» профиля «Прикладная информатика» направления профессиональной подготовки бакалавриата 09.03.03 «Прикладная информатика» (2ой семестр) / руководитель асс. Карпов Д.А. / кафедра Общей информатики Института ИТ МИРЭА – с. 42, илл. 18, ист. 6, (в т.ч. 2 на англ. яз.).

Целью работы является создание компьютерной игры “Змейка“. Разработана программа, выполняющая эти требования**.**

**Введение**

В ходе данного проекта должны будут закрепиться знания по объективно-ориентированному программированию на языке программирования C++. Компьютерная игра “Змейка“ представляет собой специальную программу, с помощью которой вы сможете развлечь себя, и развить навыки внимательности и быстроты реакции.

# Глава 1. Техническое задание

## **Введение:**

### **Наименование программы**

Наименование программы: Компьютерная игра «Змейка».

* + 1. **Назначение и область применения**

Программа «Компьютерная игра “Змейка”» предназначена для развлекательных целей и развития внимательности и быстроты реакции.

* 1. **Требования к программе или программному изделию:**

**1.2.1 Требования к функциональным характеристикам**

Программа должна обеспечивать возможность выполнения перечисленных ниже функций:

1. Функции перемещения “Змейки” по игровому полю.
2. Функция обнаружения, совмещения “Змейки” с объектами в поле (далее еда).
3. Функции случайного добавления объектов на карту.
4. Функции увеличения размера “Змейки”.
   * 1. **Требования к надежности**

Надежное (устойчивое) функционирование программы должно быть обеспечено выполнением определённых действий, перечень которых приведен ниже:

1. Базовые знания с работой программного обеспечения;
2. Использованием лицензионного программного обеспечения;
3. Тестирование
   1. **Условия эксплуатации****:**
      1. **Требования к составу и параметрам технических средств**

В состав технических средств должен входить персональный компьютер включающий в себя:

1. Процессор не менее Pentium M;   
2. Оперативную память объемом, 1 Гигабайт, не менее;   
3. Операционную систему Windows, не ниже Windows 7;

* + 1. **Требования к информационной и программной совместимости:**
       1. **Требования к исходным кодам и языкам программирования:**

Исходные коды программы должны быть реализованы на языке C++. В качестве интегрированной среды разработки программы должна быть использована среда Qt creator 5.10.0. В качестве компилятора должен быть использован MinGW.

* + - 1. **Требования к программным средствам, используемым программой:**

Системные программные средства, используемые программой, должны быть представлены лицензионной локализованной версией операционной системы.

* + 1. **Специальные требования**

Программа должна обеспечивать взаимодействие с пользователем посредством графического пользовательского интерфейса.

* 1. **Требования к программной документации**
     1. **Предварительный состав программной документации**

Состав программной документации должен включать в себя:

1. Техническое задание.
2. Программу и методики испытаний.
3. Руководство пользователя.
   1. **Стадии и этапы разработки**
      1. **Стадии разработки**

Разработка должна быть проведена в три стадии:

1. Разработка технического задания.
2. Рабочее проектирование.
3. Внедрение.
   * 1. **Этапы разработки**

На стадии разработки технического задания должен быть выполнен этап разработки, согласования и утверждения настоящего технического задания.

На стадии рабочего проектирования должны быть выполнены перечисленные ниже этапы работ:

1. Разработка программы.
2. Разработка программной документации.
3. Испытания программы.

На стадии внедрения должен быть выполнен этап разработки - подготовка и передача программы.

* + 1. **Содержание работ по этапам**

На этапе разработки технического задания должны быть выполнены перечисленные ниже работы:

1. Постановка задачи.
2. Определение и уточнение требований к техническим средствам. Определение требований к программе.
3. Определение стадий, этапов и сроков разработки программы и документации на неё.
4. Согласование и утверждение технического задания.

На этапе разработки программы должна быть выполнена работа по программированию (кодированию) и отладке программы.

На этапе разработки программной документации должна быть выполнена разработка программных документов в соответствии с требованиями к составу документации.

На этапе испытаний программы должны быть выполнены перечисленные ниже виды работ:

1. Разработка, согласование и утверждение программы и методики испытаний.
2. Проведение приемо-сдаточных испытаний.
3. Корректировка программы и программной документации по результатам испытаний.

На этапе подготовки и передачи программы должна быть выполнена работа по подготовке и передаче программы и программной документации в эксплуатацию.

* 1. **Порядок контроля и приемки**
     1. **Виды испытаний**

Демонстрация работы программы и ее функционала.

Тестирование различных сценариев работы (взаимодействие с объектами, увеличение размера “Змейки”, перемещение “Змейки” по игровому полю, случайное добавление объектов на карту).

* + 1. **Общие требования к приемке работы**

Программа должна соответствовать всем вышесказанным требованиям.

# Глава 2. Проектирование и разработка программы

## **Проектирование программы**

### **Функциональные требования к программе**

Функциональные требования к программе описаны в пункте № 1.2 Технического задания.

## **Проектирование классов**

Для реализации проекта будет использован язык программирования С++. Данный язык является одним из самых наиболее распространённых объектно-ориентированных языков программирования (ООП).

Целью данной работы является написание программы «компьютерная игра “Змейка“». Для удобства пользования этой программы будет разработан графический пользовательский интерфейс (англ. GUI). Разработанный GUI должен содержать визуализацию движений змейки по карте.

## **Разработка программы**

## **Описание проектного решения**

Общая UML диаграмма показана на рисунке 2.1:

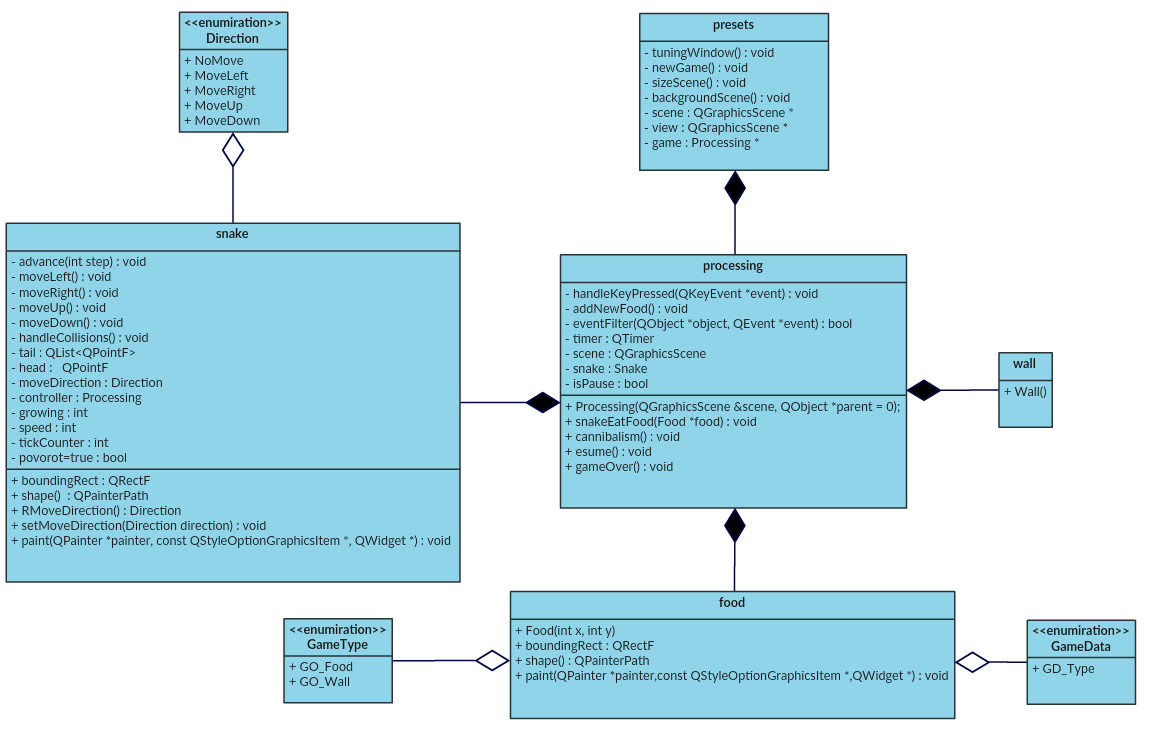


Рис. 2.1 Общая UML диаграмма

Класс **presets (**см. рис. 2.2) является основным классом программы «Компьютерная игра “Змейка“», который её запускает. В этом классе задаются такие функции, как **tuningWindow**(), которая задает стиль отображения виджета, **newGame**(), предлагающая игроку начать новую игру, **sizeScene**(), которая задает масштаб игрового поля и **backgroundScene**(), которая отвечает за графические параметры игрового поля.

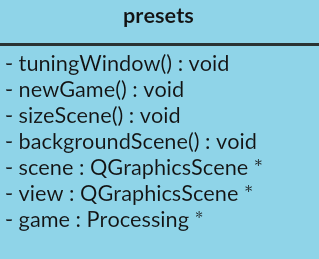


Рис. 2.2 UML диаграмма класса presets

Класс **processing (**см. рис. 2.3)является самым главным в программе, он контролирует все процессы в программе. Функции **handleKeyPressed**(QKeyEvent \*event) и ***eventFilter***(QObject \*object, QEvent \*event) отвечают за считывание и обработку нажатия на клавишу. Функция **addNewFood**() отвечает за добавление объекта “еда”. Функция **snakeEatFood**(Food \*food) отвечает за процессы связанные совмещения “Змейки” с объектом “еда”. Функция **cannibalism**() сигнализирует, о том что “Змейка” совместилась сама с собой . А такие функции **esume**(), **gameOver**() отвечают соответственно за возобновление и окончание игровой сессии в результате проигрыша.

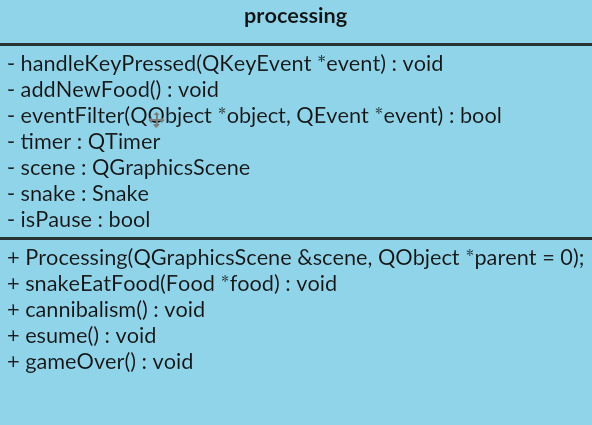


Рис. 2.3 UML диаграмма класса processing

В классе **Snake (**см. рис. 2.4) задается визуальная модель и поведение главного героя игры “Змейка”. За перемещение по игровому полю ответственны такие функции как: ***advance***(int step), **moveLeft**(), **moveRight**(), **moveUp**(), **moveDown**() и **setMoveDirection**(Direction direction). Функция ***boundingRect***() отвечает за изменение размеров “Змейки” при совмещении “Змейки” с объектом “еда”. Функция ***shape***() возвращает форму этого элемента в виде QPainterPath в локальных координатах. Функция ***paint***(QPainter \*painter, const QStyleOptionGraphicsItem \*, QWidget \*) отвечает за отображении змейки на игровом поле.

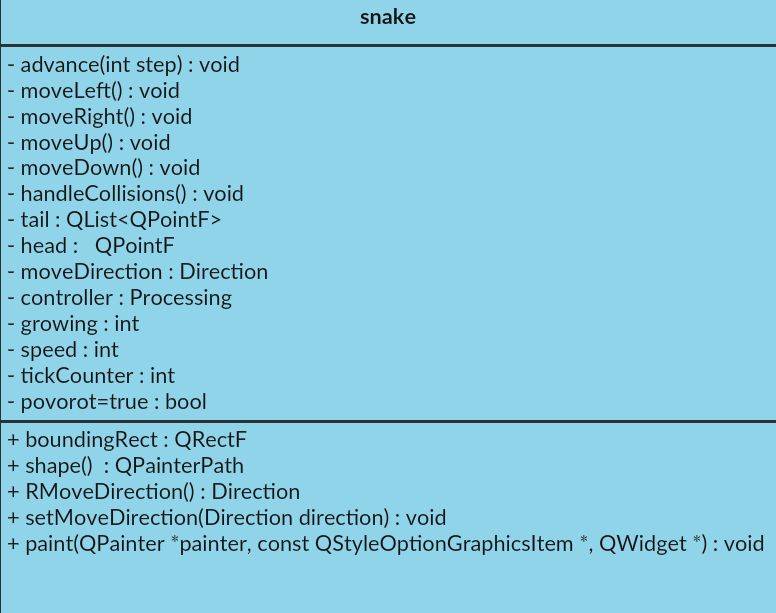


Рис. 2.4 UML диаграмма класса snake

Класс **Food (**см. рис. 2.5) - отвечает за объект “еда”. Функции **Food**(int x,int y), ***shape***() и ***boundingRect***() создают объект “еда”. Функция ***paint***(QPainter \*painter,const QStyleOptionGraphicsItem \*,QWidget \*) отвечает за отображении объекта “еда”.

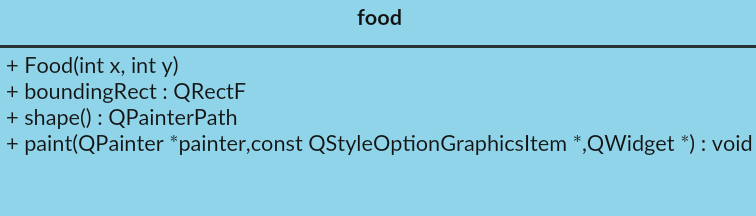


Рис. 2.5 UML диаграмма класса food

Класс **Wall (**см. рис. 2.6) предназначен для отображения стены на игровом поле, за что и отвечает функция **Wall**()**.** Мы придерживаемся одной из концепций компьютерной игры “Змейка“ именно поэтому в нашем случае стены играют роль телепорта, перемещающего змейку на противоположный конец карты, при контакте со стеной.

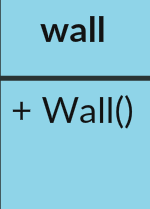


Рис. 2.6 UML диаграмма класса wall

## **Тестирование программы**

Тестирование игровой программы производилось путем проверки выполнения корректности работы программы. Перемещение “Змейки” по игровому полю, обработка и фильтрация нажатий на клавиши, совмещение “Змейки” с объектами в поле, увеличение размера “Змейки”, обработка совмещения “Змейки” самой себя и функция взаимодействия “Змейки” со стеной производится корректно и без сбоев.

# Глава 3. Руководство пользователя

Запуск программы осуществляется открытием файла Game Snake.exe. Файл с приложением находится в папке Game Snake exe для открытия файла наведите курсором на файл Game Snake.exe и кликните по нему 2 раза. Для того что бы начать играть нужно нажать на одну из клавиш управления курсором. Управление “Змейкой” осуществляется нажатием на клавиши управления курсором. Цель игры “съесть” как можно больше еды и не врезаться в самого себя. Для завершения игры нажмите на крестик, указанный на рисунке 2.7 стрелкой.

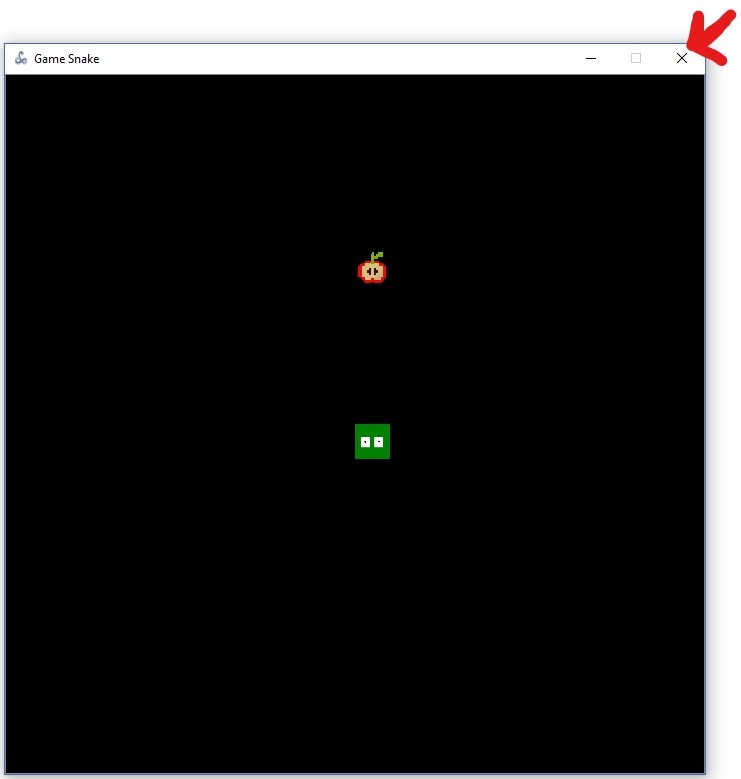


Рис. 2.7 Завершение программы

# Заключение

# В результате выполнения курсовой работы была полностью реализована программа «Компьютерная игра “Змейка“», используя объектно-ориентированное программирование. Написание программы способствовало закреплению теоретического материала на практических занятиях. В ходе выполнения курсовой работы были более подробно изучены базовые элементы управления, их свойства и методы.

# Список источников

# 1) http://www.quizful.net/post/enum-types-c

# 2) https://doc.qt.io/qt-5/qtimer.html

# 3) https://doc.qt.io/qt-5/qgraphicsscene.html

# 4) https://doc.qt.io/qt-5/qgraphicsview.html

# 5) http://blog.kislenko.net/show.php?id=1302

# 6) http://doc.crossplatform.ru/qt/4.7.x/demos-declarative-snake.html

# Приложение 1. Исходный код программы

**food.h**

#ifndef FOOD\_H

#define FOOD\_H

#include <QGraphicsItem>

class Food : public QGraphicsItem

{

public:

Food(int x, int y); // Эта функция отвечает за установка координат еды

QRectF ***boundingRect***() const; // Эта функция определяет внешние границы элемента и определяется нуждается ли элемент в перерисовке

QPainterPath ***shape***() const; // Эта функция возвращает форму этого элемента в виде QPainterPath в локальных координатах.

void ***paint***(QPainter \*painter,const QStyleOptionGraphicsItem \*,QWidget \*); // Эта функция отвечает за отрисовку еды на поле

};

#endif // FOOD\_H

**presets.h**

#ifndef PRESETS\_H

#define PRESETS\_H

#include <QMainWindow>

class QGraphicsScene;

class QGraphicsView;

class Processing;

class Presets : public QMainWindow

{

Q\_OBJECT

public:

Presets (QWidget \*parent = 0);

~***Presets***();

private slots:

void **tuningWindow**(); // Эта функция задает стиль отображения виджета

void **newGame**(); // Эта функция отвечает за создание новой игры

private:

void **sizeScene**(); // Эта функция отвечает за масштаб игрового поля

void **backgroundScene**(); // Эта функция отвечает за графические параметры игрового поля

// Установка стартовых параметров

QGraphicsScene \*scene;

QGraphicsView \*view;

Processing \*game;

};

#endif // PRESETS\_H

# processing.h

#ifndef PROCESSING\_H

#define PROCESSING\_H

#include <QObject>

#include <QTimer>

class QGraphicsScene;

class QKeyEvent;

class Snake;

class Food;

class Wall;

class Processing : public QObject

{

Q\_OBJECT

public:

Processing(QGraphicsScene &scene, QObject \*parent = 0);

void **snakeEatFood**(Food \*food); // Эта функция отвечает за поедание еды

void **cannibalism**(); // Эта функция отвечает за вызов окна "Конец игры" при поедание змейки самой себя

public slots:

void **esume**(); // Эта функция отвечает за продолжение игры

void **gameOver**(); // Эта функция отвечает за создание экрана "Конец игры"

private:

~***Processing***();

void **handleKeyPressed**(QKeyEvent \*event); // Эта функция отвечает за обработку нажатий клавиш

void **addNewFood**(); // Эта функция отвечает за добавление еды на игровое поле

bool ***eventFilter***(QObject \*object, QEvent \*event); // Эта функция за считывание нажатия на клавишу

// Установка стартовых параметров

QTimer timer;

QGraphicsScene &scene;

Snake \*snake;

bool Pause;

};

#endif // PROCESSING\_H

# snake.h

#ifndef SNAKE\_H

#define SNAKE\_H

#include <QGraphicsItem>

#include <QRectF>

class Processing;

class Snake : public QGraphicsItem

{

public:

// перечисление для направления змейки

enum Direction

{

NoMove,

MoveLeft,

MoveRight,

MoveUp,

MoveDown

};

Snake(Processing & controller);

QRectF ***boundingRect***() const; // Эта функция отвечает за увеличение змейки

QPainterPath ***shape***() const; // Эта функция возвращает форму этого элемента в виде QPainterPath в локальных координатах.

Direction **RMoveDirection**(); // Эта функция возвращает направление змейки

void **setMoveDirection**(Direction direction); // Эта функция устанавливает направление змейки

void ***paint***(QPainter \*painter, const QStyleOptionGraphicsItem \*,

QWidget \*); // Эта функция отвечает за отрисовку змейки на поле

private:

void ***advance***(int step); // Эта функция отвечает за обновлении положение змейки на игровом поле

void **moveLeft**(); // Эта функция отвечает за поворот змейки в левую сторону

void **moveRight**(); // Эта функция отвечает за поворот змейки в праву сторону

void **moveUp**(); // Эта функция отвечает за поворот змейки в верхнюю сторону

void **moveDown**(); // Эта функция отвечает за поворот змейки в нижнюю сторону

void **handleCollisions**(); // Эта функция отвечает за обработку столкновения змейки с самой собой и едой

// Установка стартовых параметров

QList<QPointF> tail;

QPointF head;

Direction moveDirection;

Processing &controller;

int growing;

int speed;

int tickCounter;

bool povorot=true;

};

#endif // SNAKE\_H

# variable.h

#ifndef VARIABLE\_H

#define VARIABLE\_H

// Константы для игровых объектов

enum GameData

{

GD\_Type

};

enum GameType

{

GO\_Food,

GO\_Wall

};

#endif // VARIABLE\_H

# wall.h

#ifndef WALL\_H

#define WALL\_H

#include <QGraphicsItem>

class Wall : public QGraphicsItem

{

public:

Wall(); // Эта функция отвечает за существование "стены" на игровом поле

};

#endif // WALL\_H

# food.cpp

#include <QPainter>

#include "food.h"

#include "variable.h"

Food::**Food**(int x,int y) // Эта функция отвечает за установка координат еды

{

setPos(x,y);

setData(GD\_Type,GO\_Food);

}

void Food::***paint***(QPainter \*painter,const QStyleOptionGraphicsItem \*,QWidget \*) // Эта функция отвечает за отрисовку еды на поле

{

painter->save();

painter->drawPixmap(0,0,10,10,QPixmap(":/images/fruit.png"));

painter->restore();

}

QPainterPath Food::***shape***() const // Эта функция возвращает форму этого элемента в виде QPainterPath в локальных координатах.

{

QPainterPath p;

p.addEllipse(QPointF(5,5),3,3);

return p;

}

QRectF Food::***boundingRect***() const // Эта функция определяет внешние границы элемента и определяется нуждается ли элемент в перерисовке

{

return QRectF(-10,-10,20,20);

}

}

# main.cpp

#include <QApplication>

#include "presets.h"

int main(int argc, char \*argv[])

{

QApplication a(argc, argv);

Presets w;

w.show();

return a.exec();

}

# presets.cpp

#include <QGraphicsView>

#include <QIcon>

#include <QTimer>

#include "processing.h"

#include "presets.h"

Presets::**Presets**(QWidget \*parent) :QMainWindow(parent),

// Установка стартовых параметров

scene(new QGraphicsScene(this)),

view(new QGraphicsView(scene, this)),

game(new Processing(\*scene, this))

{

// Установка параметров

setCentralWidget(view);

setFixedSize(700,700);

setWindowIcon(QIcon(":/images/logo snake.jpg"));

sizeScene();

backgroundScene();

QTimer::singleShot(0,this,SLOT(tuningWindow()));

}

Presets::~***Presets***()

{

}

void Presets::**sizeScene**() // Эта функция отвечает за масштаб игрового поля

{

scene->setSceneRect(-100, -100, 200, 200);

}

void Presets::**tuningWindow**() // Эта функция задает стиль отображения виджета

{

view->fitInView(scene->sceneRect(),Qt::IgnoreAspectRatio);

}

void Presets::**backgroundScene**() // Эта функция отвечает за графические параметры игрового поля

{

QPixmap bg(600, 600);

QPainter p(&bg);

p.setBrush(QBrush(Qt::black));

p.drawRect(0, 0, 600, 600);

view->setBackgroundBrush(bg);

}

void Presets::**newGame**() // Эта функция отвечает за создание новой игры

{

QTimer::singleShot(0, game, SLOT(gameOver()));

}

}

# processing.cpp

#include <QEvent>

#include <QGraphicsScene>

#include <QKeyEvent>

#include <QMessageBox>

#include "processing.h"

#include "food.h"

#include "snake.h"

Processing::**Processing**(QGraphicsScene &scene, QObject \*parent) :

// Установка стартовых параметров

QObject(parent),

scene(scene),

snake(new Snake(\*this)),

Pause(false)

{

//Установка параметров

timer.start(1000/33);

Food \*a1 = new Food(0, -50);

scene.addItem(a1);

scene.addItem(snake);

scene.installEventFilter(this);

esume();

}

Processing::~***Processing***()

{

}

void Processing::**snakeEatFood**(Food \*food) //Эта функция отвечает за поедание еды

{

scene.removeItem(food);

addNewFood();

}

void Processing::**cannibalism**() // Эта функция отвечает за вызов окна "Конец игры" при поедание змейки самой себя

{

QTimer::singleShot(0, this, SLOT(gameOver()));

}

void Processing::**addNewFood**() // Эта функция отвечает за добавление еды на игровое поле

{

int x, y;

do

{

x = (int)(qrand() % 200) / 10 - 10;

y = (int)(qrand() % 200) / 10 - 10;

x \*= 10;

y \*= 10;

}

while (snake->*shape*().contains(snake->mapFromScene(QPointF(x + 5, y + 5))));

Food \*food = new Food(x, y);

scene.addItem(food);

}

void Processing::**handleKeyPressed**(QKeyEvent \*event) // Эта функция отвечает за обработку нажатий клавиш

{

if (!Pause)

switch (event->key())

{

case Qt::Key\_Left:

snake->setMoveDirection(Snake::MoveLeft);

break;

case Qt::Key\_Right:

snake->setMoveDirection(Snake::MoveRight);

break;

case Qt::Key\_Up:

snake->setMoveDirection(Snake::MoveUp);

break;

case Qt::Key\_Down:

snake->setMoveDirection(Snake::MoveDown);

break;

}

else

esume();

}

bool Processing::***eventFilter***(QObject \*object, QEvent \*event) // Эта функция за считывание нажатия на клавишу

{

if (event->type() == QEvent::KeyPress)

{

handleKeyPressed((QKeyEvent \*)event);

return true;

}

else

{

return QObject::*eventFilter*(object, event);

}

}

void Processing::**gameOver**() //Эта функция отвечает за создание экрана "Конец игры"

{

disconnect(&timer, SIGNAL(timeout()), &scene, SLOT(advance()));

if (QMessageBox::Yes == QMessageBox::information(

NULL,

tr("Конец игры"), tr("Хотите начать заново ?"),

QMessageBox::Yes|QMessageBox::No,

QMessageBox::Yes))

{

connect(&timer, SIGNAL(timeout()),

&scene, SLOT(advance()));

scene.clear();

snake = new Snake(\*this);

scene.addItem(snake);

addNewFood();

}

else

{

exit(0);

}

}

void Processing::**esume**() //Эта функция отвечает за продолжение игры

{

connect(&timer, SIGNAL(timeout()),

&scene, SLOT(advance()));

Pause = false;

}

# snake.cpp

#include <QPainter>

#include "snake.h"

#include "processing.h"

#include "food.h"

#include "variable.h"

Snake::**Snake**(Processing &controller) :

// Установка стартовых параметров

head(0,0),

growing(2),

speed(4),

moveDirection(NoMove),

controller(controller)

{

}

QRectF Snake::***boundingRect***() const // Эта функция отвечает за увелечение змейки

{

int minX = head.x();

int minY = head.y();

int maxX = head.x();

int maxY = head.y();

foreach (QPointF p, tail) // foreach несете функцию перебора всех элементов контейнера

{

maxX = p.x() > maxX ? p.x() : maxX;

maxY = p.y() > maxY ? p.y() : maxY;

minX = p.x() < minX ? p.x() : minX;

minY = p.y() < minY ? p.y() : minY;

}

QPointF tl = mapFromScene(QPointF(minX, minY));

QPointF br = mapFromScene(QPointF(maxX, maxY));

QRectF bound = QRectF(tl.x(),tl.y(),br.x() - tl.x() + 10,br.y() - tl.y() + 10);

return bound;

}

QPainterPath Snake::***shape***() const // Эта функция возвращает форму этого элемента в виде QPainterPath в локальных координатах.

{

QPainterPath path;

path.setFillRule(Qt::WindingFill);

path.addRect(QRectF(0,0,10,10));

foreach (QPointF p, tail)

{

QPointF itemp = mapFromScene(p);

path.addRect(QRectF(itemp.x(), itemp.y(),10,10));

}

return path;

}

void Snake::***paint***(QPainter \*painter, const QStyleOptionGraphicsItem \*, QWidget \*) // Эта функция отвечает за отрисовку змейки на поле

{

painter->save();

painter->fillPath(*shape*(),Qt::darkGreen);

if (povorot==true)

{

painter->drawPixmap(0,0,10,10,QPixmap(":/images/0 grade.png"));

}

else

{

painter->drawPixmap(0,0,10,10,QPixmap(":/images/180 grade.png"));

}

painter->restore();

}

void Snake::**setMoveDirection**(Direction direction) // Эта функция устанавливает направление змейки

{

if (moveDirection == MoveLeft && direction == MoveRight)

return;

if (moveDirection == MoveRight && direction == MoveLeft)

return;

if (moveDirection == MoveUp && direction == MoveDown)

return;

if (moveDirection == MoveDown && direction == MoveUp)

return;

moveDirection = direction;

}

Snake::Direction Snake::**RMoveDirection**() // Эта функция возвращает направление змейки

{

return moveDirection;

}

void Snake::**moveLeft**() // Эта функция отвечает за поворот змейки в левую сторону

{

head.rx() -= 10;

if (head.rx() < -100)

{

head.rx() = 90;

}

povorot=false;

}

void Snake::**moveRight**() // Эта функция отвечает за поворот змейки в праву сторону

{

head.rx() += 10;

if (head.rx() >= 100)

{

head.rx() = -100;

}

povorot=false;

}

void Snake::**moveUp**() // Эта функция отвечает за поворот змейки в верхнюю сторону

{

head.ry()-=10;

if (head.ry()<-100)

{

head.ry()=90;

}

povorot=true;

}

void Snake::**moveDown**() // Эта функция отвечает за поворот змейки в нижнюю сторону

{

head.ry() += 10;

if (head.ry() >= 100)

{

head.ry() = -100;

}

povorot=true;

}

void Snake::***advance***(int step) // Эта функция отвечает за обновлении положение змейки на игровом поле

{

if (!step)

{

return;

}

if (tickCounter++ % speed != 0)

{

return;

}

if (moveDirection == NoMove)

{

return;

}

if (growing > 0)

{

QPointF tailPoint = head;

tail << tailPoint;

growing -= 1;

} else

{

tail.removeFirst();

tail << head;

}

switch (moveDirection)

{

case MoveLeft:

moveLeft();

break;

case MoveRight:

moveRight();

break;

case MoveUp:

moveUp();

break;

case MoveDown:

moveDown();

break;

}

setPos(head);

handleCollisions();

}

void Snake::**handleCollisions**() // Эта функция отвечает за обработку столкновения змейки с самой собой и едой

{

QList<QGraphicsItem \*> collisions = collidingItems();

foreach (QGraphicsItem \*collidingItem, collisions)

{

if (collidingItem->data(GD\_Type) == GO\_Food)

{

controller.snakeEatFood((Food \*)collidingItem);

growing += 1;

}

}

if (tail.contains(head))

{

controller.cannibalism();

}

}

# wall.cpp

#include "wall.h"

Wall::**Wall**()

{

}

**Приложение 2. Ресурсы**

