## Санкт-Петербургский государственный политехнический университет

Факультет технической кибернетики

Кафедра компьютерных систем и программных технологий

# Курсовая работа по курсу «Программирование»

# Тема работы: «Разработка приложения с графическим интерфейсом на языке C++ с использованием библиотеки QT»

Выполнила: Замотаева Юлия, гр. 1081/3

Преподаватель: доц. Пышкин Е.В.

Санкт - Петербург 2012

## 1. Образ и границы проекта

Курсовая работа посвящена разработке программного обеспечения для компьютерной игры «Змейка.

Snake (Питон, Удав, Змейка и др.) – компьютерная игра, возникшая в середине или в конце 1970‐х. В свое время игра стала очень популярной как среди детей и молодежи, так и среди взрослого населения всего мира.

В игре должны быть реализованы стандартные правила:

В игре участвует один игрок, который управляет существом, напоминающим змею, которое ползает по плоскости (как правило, ограниченной стенками), собирая еду (или другие предметы), увеличивающие количество очков, но избегая столкновения с собственным хвостом и краями игрового поля. Каждый раз, когда змея съедает кусок пищи, она становится длиннее, что постепенно усложняет игру. Игрок управляет направлением движения головы змеи (обычно 4 направления: вверх, вниз, влево, вправо), а хвост змеи движется следом. Игрок не может остановить движение змеи.

Поле при этом разбито на некоторое количество квадратов. Каждый из таких квадратов является отдельной частью поля, которая может быть либо стеной, либо обычным полем по которому можно ползать, либо едой.

Игра должна иметь счетчик набранных игроком очков, таблицу игроков, которая формируется в файле после окончания игры.



## 2. Техническое задание

Разработать данное приложение, представленное в предыдущем пункте на языке C++ с использованием библиотеки Qt для реализации графического интерфейса пользователя.

## Режимы работы приложения

Реализуется два режима работы этапа:

1. Консольное приложение без графического интерфейса. В этом режиме обеспечивается проверка бизнес-логики.
2. Интеграция кода программы для консоли с графической библиотекой Qt, и создание графического интерфейса.

## Требования к консольному интерфейсу

Главное меню консольного приложения:

1) Выбрать сценарий.

2) Описание игры (сценариев).

3) Выйти из игры.

## Основные сценарии функционирования программы

Основной сценарий функционирования консольного приложения состоит в следующем:

1. Выбрать сценарий: пользователь выбирает 1из трех предложенных сценариев, в каждом из которых уже задана определенная ситуация и он тем самым сможет наглядно увидеть, что происходит в самом корне программы (о самих сценариях подробно написано в следующем пункте).
2. Описание игры: здесь можно прочитать, что происходит в каждом конкретном сценарии:
3. В первом: поле: 6\*6, координаты змеи (1,2), длина хвоста -4, координаты яблока (3,0). На поле находится фрукт, змея движется, съедает его и увеличивает свою длину.
4. Во втором: поле 8\*8, координаты змеи (2,2), длина хвота-4. Змея движется вверх и врезается в стену.
5. В третьем: поле 5\*5, координаты змеи (1,1), длина хвоста-4. Змея поворачивается вправо и врезается в себя.
6. Выйти из игры.

## Требования к функциональности консольного приложения

1. Отображение пунктов меню (см. главное меню консольного приложения).
2. Наличие описанных раннее сценариев приложения (моделирование определенного поведения вызовами методов бизнес логики) .

## Требования к графическому пользовательскому интерфейсу

1. Начальный экран с пунктами меню в виде кнопок:
2. new Game;
3. continue last Game;
4. show Scores;
5. About;
6. Exit;
7. Регистрация игрока в определенном окне, после нажатия кнопки “new Game”.
8. Начало игры: экран с зелёным полем, на экране появляется змея начальной длины, которая начинает двигаться с момента нажатия клавиш, в случайной точке поля появляются фрукты (яблоки – разбрасываются в начале игры и остаются статичными, и груши – тоже разбрасываются, но исчезают через определенное время). Во время игры идет начисление очков за каждый съеденный фрукт.
9. Подсчет результатов после окончания игры.
10. Продолжение незаконченной игры.

## Основной сценарий функционирования программы

Основной сценарий функционирования приложения состоит в следующем:

1. New Game.

Регистрация игрока: игрок должен ввести имя (ник), под которым он будет играть (ведение записи очков).

* Выбор начальных координат змеи случайным образом.
* Прорисовка начального положения.
* Генерация на поле яблок и груш случайным образом. Проверка: не появились ли фрукты на черве.
* Управление “Змейкой” с клавиатуры.
* Проверка: “не укусила” ли “змейка сама себя”, “не произошло ли врезание в стену” и если укусила или врезалась, то выводится информационное окно (информационная форма) и игра заканчивается, в противном случае переходим к следующему пункту.
* Проверка: “не съела” ли “змейка” “яблоко или грушу”, если съела, то наращиваем ей хвост и продолжаем движение, если нет, то ничего не наращиваем, и змея движется дальше. Игра продолжается до тех пор, пока змея не столкнется со стеной или не пересечет саму себя.

Формирование результатов: подсчет съеденных груш и яблок и перевод их в очки (1 яблоко- 10 очков, 1 груша-20), а также время игры. После окончания игры набранное число очков данным игроком сохраняется вместе с его именем в текстовый выходной файл. С последующим сохранением очков и имени следующего игрока, текстовый файл сортируется по возрастанию очков.

1. Continue last Game: загрузка предыдущей незаконченной игры (возможно в формате XML).
2. Show scores: просмотр статистики - подразумевает просмотр текстового файла, содержащего ник-нейм игрока и набранное количество очков.
3. About: Описание игры-общие правила игры.

## Форматы данных

В текстовом файле содержатся имена игроков с их результатами. Он обновляется (редактируется) после окончания игры и загружается при выборе необходимой кнопки меню.

## Обработка ошибок

Приложение должно обрабатывать следующие виды ошибок:

Консоль:

1. Если пользователь вводит что-то другое, отличное от номеров пунктов меню, то выведется сообщение об ошибке и просьба ввести один из предложенных пунктов.

Графика:

1) Вывести сообщение об ошибке, если игрок попытался остановить змею (нажал противоположную движению клавишу) – в том случае, если получится удовлетворительно поработать с динамикой и змея будет двигаться без остановок.

2) Ошибка загрузки входного файла с очками игроков.

## 3. Проектирование системной архитектуры

### Классы консольного приложения

В консольном приложении будут реализованы следующие классы:

1. class Field {

private:

int\*\* field;

int width, height;

public:

Field (int W, int H);

Field ();

~Field ();

bool set\_field (int x, int y, int new\_value); //для изменения ячейки поля

int get\_field (int x, int y); //получить ячейку

int get\_width ();

bool set\_width (int new\_width);

int get\_height();

bool set\_height (int new\_height);

bool createField ();

void init ();

bool resize(int size1, int size2);

};

Класс Field отвечает за представление игрового поля. На данном этапе возвращает конкретные символы, определенные нахождением тех или иных объектов на нем.

1. class Fruct { //единичный фрукт, абстрактый базовый класс

private:

int x, y;

public:

Fruct (int X, int Y);

~Fruct();

int getx();

int gety();

bool setx (int new\_x);

bool sety (int new\_y);

virtual int fructenumerate ( )=0;

};

Класс Fruct отвечает за представление единичного фрукта, является абстрактным базовым классом. От него наследуются следующие классы: Яблоко, Груша.

1. class Apple: public Fruct { //производный класс

public:

Apple (int X, int Y);

~Apple();

int fructenumerate ();

};

1. class Pear: public Fruct { //производный класс

public:

int time;

Pear (int t, int X, int Y);

~Pear();

int fructenumerate ();

};

1. class Fructs { //Фрукты, состоящие из фруктов

private:

int numberfructs; //число фруктов

public:

std::vector<Fruct\*>Fruits; //в public для сохранения свойств (нужно получать размер и т.п., а не только сам элемент)

Fructs (int len);

Fructs();

bool set\_numberfructs (int new\_numberfructs);

int get\_numberfructs ();

void add (int size1,int size2); //случайное добавление фруктов

void addApple(Apple \*new\_apple);//вручную добавляем

void addPear(Pear \*new\_pear);

bool deleteFruct(int number);

};

Класс Fructs отвечает за представления фруктов(мн.ч). Включает в себя динамический массив и различные методы.

1. class Game { //единственный консольный класс

private:

int time;

int cycle\_snake;

float score;

public:

Game ();

~Game();

Scenario1 \*scenario1;

Scenario2 \*scenario2;

Scenario3 \*scenario3;

Scenario \*scenario0;

int gettime();

bool settime(int newtime);

int getcycle\_snake();

bool setcycle\_snake(int newcycle\_snake);

void update();

void printField(Scenario\*currentscenario);

void updateField(Scenario\*currentscenario);

static char intToChar (int code); //служебная функция для норм.отрисовки поля

static void printToFile(int cycle,Scenario\*currentscenario);

};

Класс Game отвечает за сам игровой процесс. (В консоли это предполагает запуск набора сценариев).

1. class Scenario

{

public:

Field \*scenarioField;

Zmeika \*scenarioZmeika;

Fructs \*scenarioFructs;

Scenario(void);

~Scenario(void);

void virtual init ()=0;

bool virtual run(int cycle\_snake)=0;

};

Класс Scenario обеспечивает наличие разных сценариев. Базовый класс, от которого наследуются следующие сценарные классы:

1. class Scenario1: public Scenario

{

public:

Scenario1(void);

~Scenario1(void);

void init ();

bool run(int cycle\_snake);

};

1. class Scenario2: public Scenario

{

public:

Scenario2(void);

~Scenario2(void);

void init ();

bool run(int cycle\_snake);

};

1. class Scenario3: public Scenario

{

public:

Scenario3(void);

~Scenario3(void);

void init ();

bool run(int cycle\_snake);

};

1. class Zmeika { //состоит из головы и хвоста

private:

int headx, heady;

int napx, napy;

bool alive;

int dir;

public:

Zmeika();

Zmeika(int X, int Y, int Napx, int Napy, int LTail);

Zmeika(int X, int Y, int new\_direction, int LTail);

~Zmeika();

bool getAlive();

void setAlive(bool newalive);

bool setdir(int newdir);

int getdir();

int get\_headx();

int get\_heady();

bool set\_headx (int new\_x);

bool set\_heady (int new\_y);

int get\_napx();

int get\_napy();

bool set\_napx (int new\_napx);

bool set\_napy (int new\_napy);

std::vector<ZmElement\*>Tail2;

ZmElement\* getElementTail2(int Number);

int enumerateHead ();

int enumerateTail ();

bool rotateUp();

bool rotateDown();

bool rotateLeft();

bool rotateRight();

bool rotateClockwise(); //по часовой стрелке

bool rotateAClockwise(); //против

bool move();

void shift();

bool checkEatFruct(Fructs \*allFructs,Field \*fullfield);

bool checkWall(Field \*fullfield);

bool checkPosition(int X,int Y);

bool checkSelfEating();

bool addTail(Field \*fullfield);

}

Класс отвечает за представление всей змейки в целом.

1. class ZmElement {

private:

int x, y;

int napx, napy;

int dir;

public:

ZmElement ();

ZmElement (int X, int Y);

~ZmElement();

int getx();

int gety();

bool setx (int new\_x);

bool sety (int new\_y);

int getnapx();

int getnapy();

bool set\_napx(int new\_napx );

bool set\_napy(int new\_napy);

};

Класс отвечает за представление элементов, из которых состоит змея (то есть, не включая головы).

1. class Test {

bool fructsCreator (Fructs\*fruits) ; //создан ли, ск элементов

bool zmeikaCreator (Zmeika\*zmeia) ; // змейка, хвост

bool fieldCreator (Field\*field) ;//поле

};

Тестовый класс (создавался для отладки приложения).

1. class HighScores

{

public:

HighScores();

virtual ~HighScores();

std::vector<char\*>Names;

std::vector<int>Scores;

bool addNewScore(char \* new\_name,int score);

bool saveToFile();

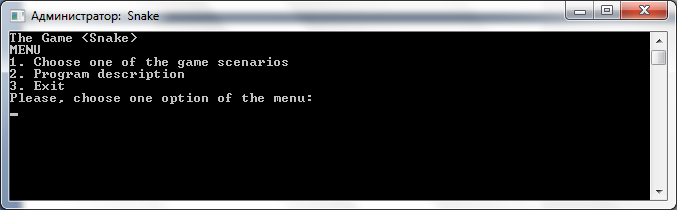
bool loadFromFile();

};

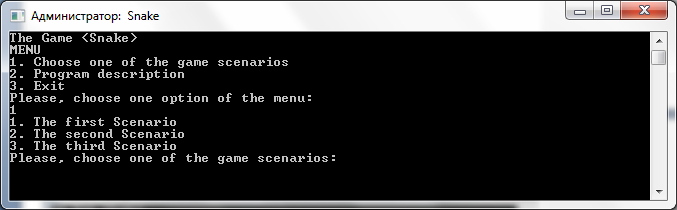
Класс для ведения учетной записи игрока. Понадобится для графического интерфейса.

Процесс тестирования приложения:

1. Была обнаружена следующая недоработка: при введении пользователем не того, что требовалось бы ввести (не пункт меню, а нечто другое) приложение ничего не сообщало, поэтому обработка этого недочета была исправлена (см.обработка ошибок).
2. Запускаем сценарии:

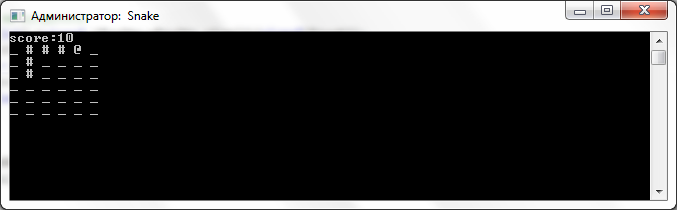


Выбираем 1 пункт меню:



И снова 1:





Видим, что змея съела фрукт, и произошло увеличение длины хвоста, программа работает верно. Остальные сценарии также были запущены, и все работало исправно. Также проводились изменения в сценариях, например, в первом в самом коде создавалось поле другого размера, выводилось не яблоко, а груша (в принципе в данной версии консоли принципиально они не отличаются), у змеи были другие координаты, и все также работало верно.

### Классы графического интерфейса

К вышеперечисленным классам добавляются:

class AboutDialog: public QDialog

{

Q\_OBJECT

QLabel \*label;

protected:

virtual void *paintEvent*(QPaintEvent \*);

public:

AboutDialog(QWidget \*parent);

~*AboutDialog*();

};

Данный класс отвечает за вывод окна с описанием приложения.

class FinalResults : public QDialog

{

Q\_OBJECT

public:

FinalResults(int newscore,QWidget \*parent);

protected:

virtual void *paintEvent*(QPaintEvent \*);

QLineEdit \*Editname;

QPushButton \*ButtonCommit;

public slots:

void oncommitname();

public:

bool SaveToFile();

int score;

};

class ScoreDialog : public QDialog

{

Q\_OBJECT

protected:

virtual void *paintEvent*(QPaintEvent \*);

public:

bool LoadFromFile();

ScoreDialog(QWidget \*parent);

~*ScoreDialog*();

HighScores Records;

QLabel \*text;

vector <QString>all;

};

Данные классы отвечают за представление результатов игры отдельном окне.

class Game : public QMainWindow

{

Q\_OBJECT

public:

Field \* Gfield;

Zmeika \*Gzmeika;

Fructs \*Gfructs;

public:

virtual void *keyPressEvent*(QKeyEvent \* ev);

void *paintEvent*(QPaintEvent\*);

void update();

void initgame();

void addfructs(int amount);

void updatefield();

bool checkPositionForFruct(int X,int Y);

private:

bool Pause;

public:

Game();

~Game() {}

};

Класс отвечает за представление игры в целом.

class MyDialog: public QDialog

{

Q\_OBJECT

protected:

virtual void *paintEvent*(QPaintEvent \*);

virtual void *keyPressEvent*(QKeyEvent \* ev);

public:

MyDialog(QWidget \*parent);

~*MyDialog*();

Game \* newGame;

bool state1;

bool state2;

};

Класс отвечает за представление игры в отдельном окне.

class Widget : public QWidget

{

Q\_OBJECT

QPushButton\* button1;

QPushButton\* button2;

QPushButton\* button3;

QPushButton\* button4;

public:

MyDialog\* Gdialog;

ScoreDialog\* Sdialog;

AboutDialog\* Abdialog;

public:

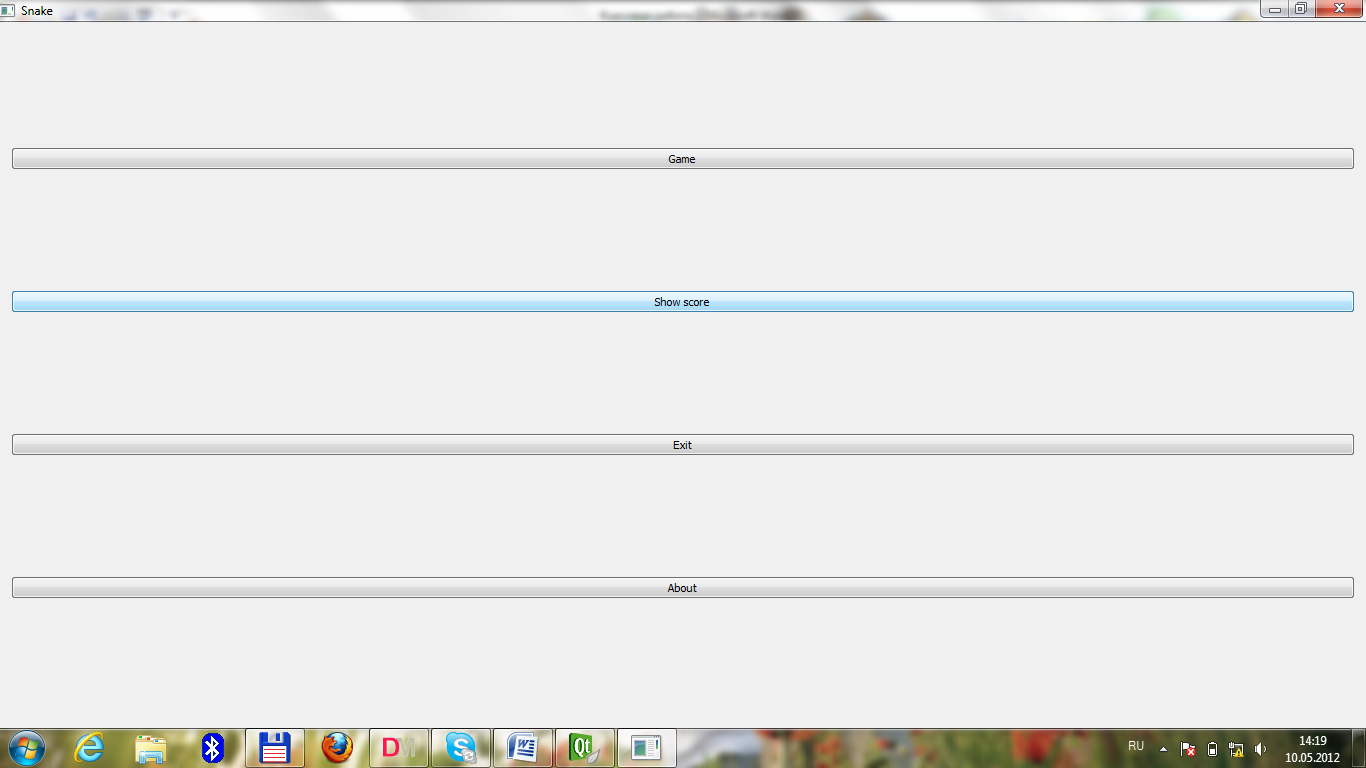
Widget(QWidget \*parent = 0);

~*Widget*();

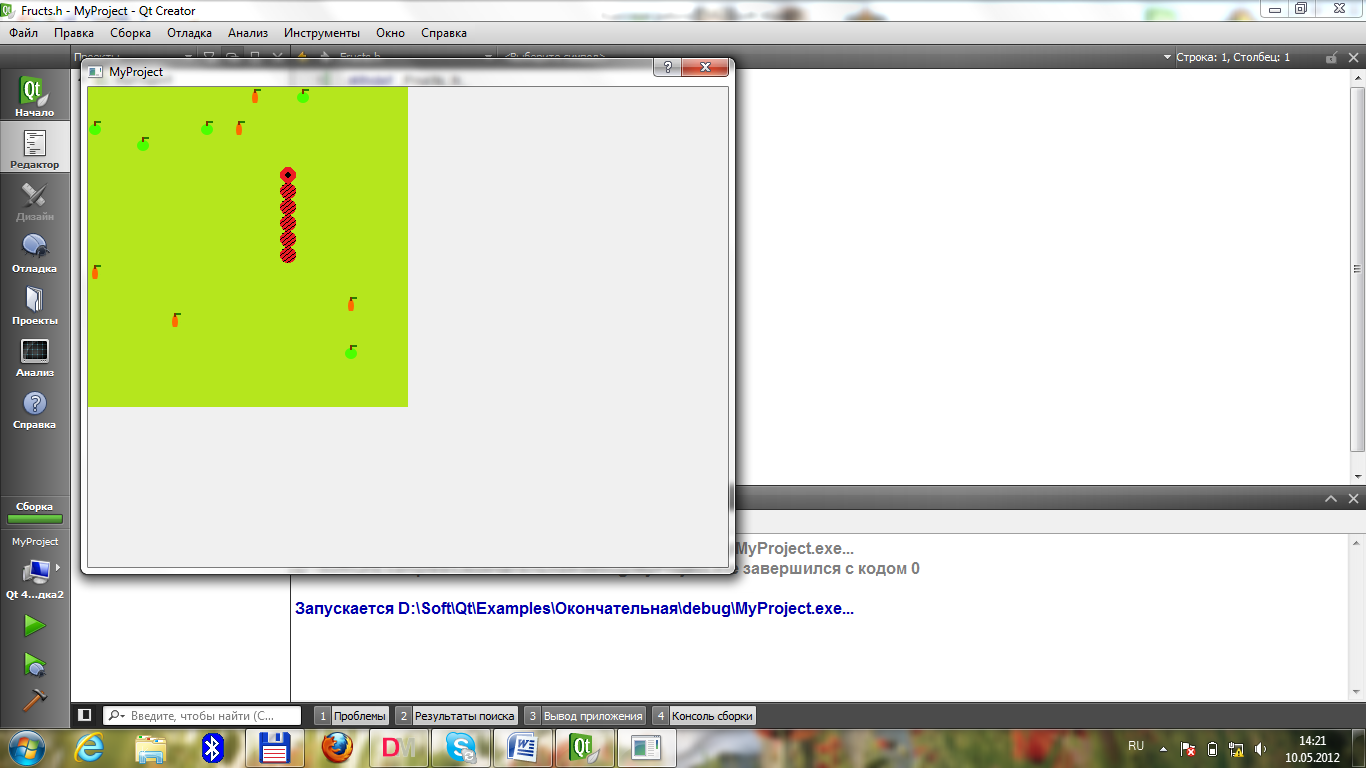
Данный класс отвечает за создание меню приложения.

Процесс тестирования приложения:

1. Открываем меню:



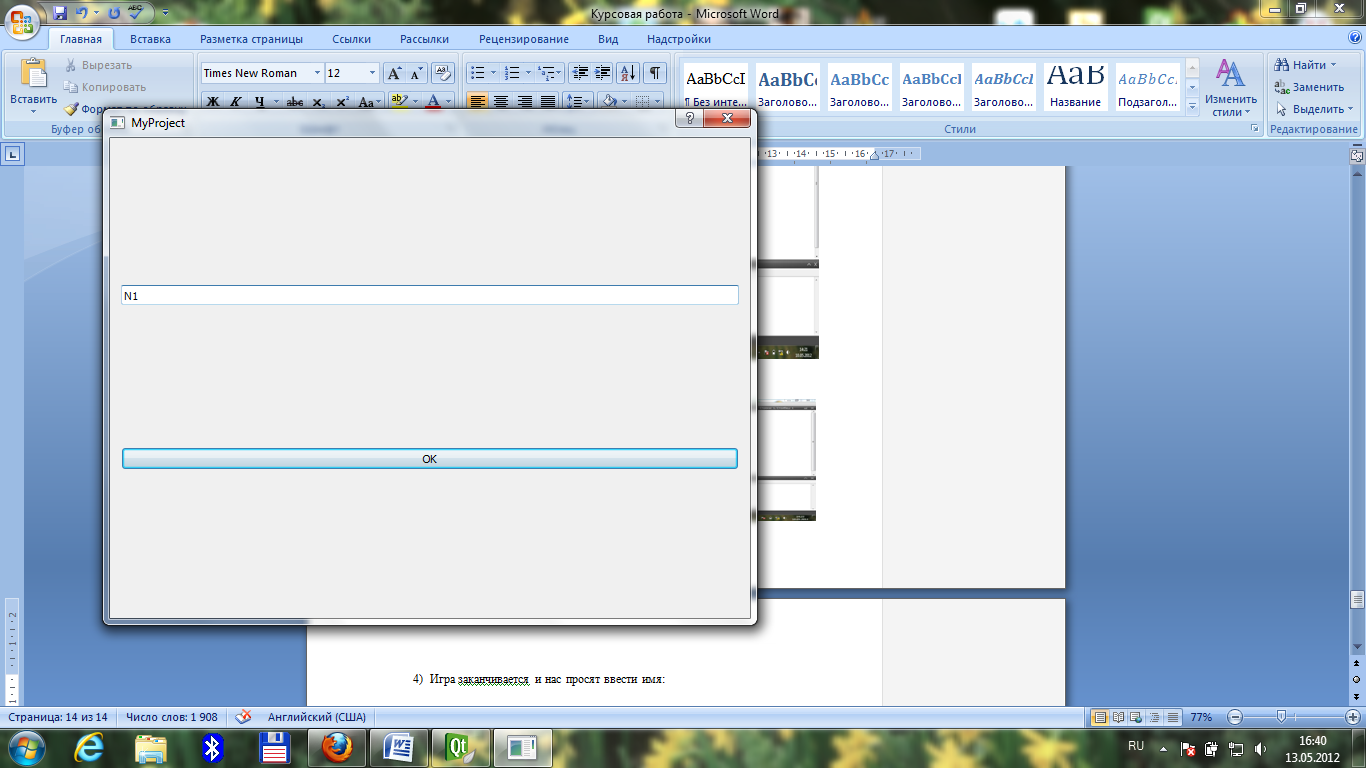
1. Выбираем пункт «Game», запускается игра, видим движущуюся змейку.



1. Съедаем яблоко и врезаемся в стенку:



1. Игра заканчивается, и нас просят ввести имя:



1. Потом можем посмотреть свои результаты.

Видим, что процесс теста прошел достаточно удачно. Однако, при проведении многих тестов обнаружились следующие недостатки: управление с помощью клавиш немного виснет (не успевает за змейкой), также при съедании змейкой много фруктов, хвост наращивается немного не так, как нужно. Это будет обязательно доработано и исправлено в дальнейшем.

В целом, считаю, что задача о создании игры «Змейка» мною выполнена.