МИНОБРНАУКИ РОССИИ

Федеральное государственное автономное образовательное учреждение высшего образования

«Новосибирский национальный исследовательский государственный университет»

(Новосибирский государственный университет, НГУ)

**Структурное подразделение Новосибирского государственного университета -**

**Высший колледж информатики Университета**

Кафедра информатики

**КУРСОВОЙ ПРОЕКТ**

Основы алгоритмизации и программирования

Тема: Разработка компьютерной игры «Python»

Выполнил: *Колесова Г.И.*

Студент группы: *007сб2*

Преподаватель: *Абраменко К.О.*

Работа допущена к защите: 19.06.2021

Оценка: \_\_\_\_\_\_\_\_

Подпись: \_\_\_\_\_\_\_\_\_\_\_

**ОГЛАВЛЕНИЕ**

[ВВЕДЕНИЕ 3](#_Toc74943210)

[1 ОПИСАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ 4](#_Toc74943211)

[2 ОБЗОР СУЩЕСТВУЮЩИХ МЕТОДОВ РЕШЕНИЯ ЗАДАЧИ 5](#_Toc74943212)

[3 СЦЕНАРИЙ ИГРЫ, ЭЛЕМЕНТЫ УПРАВЛЕНИЯ 7](#_Toc74943213)

[3.1 Сценарий игры 7](#_Toc74943214)

[3.2 Элементы управления 7](#_Toc74943215)

[3.3 Функциональные возможности 7](#_Toc74943216)

[4 СТРУКТУРА ПРОГРАММЫ 8](#_Toc74943217)

[4.1 Блок-схема программы 8](#_Toc74943218)

[5 ЛИСТИНГ ПРОГРАММЫ 9](#_Toc74943219)

[5.1. GameForm.cpp 9](#_Toc74943220)

[5.2. GameForm.h 14](#_Toc74943221)

# **ВВЕДЕНИЕ**

Python (Snake, Удав, Змейка и др.) — компьютерная игра, впервые появившаяся в кнопочном телефоне Nokia 6110. Разработана финским разработчиком Танели Арманто и выпущена компанией Nokia.

Игрок управляет длинным, тонким существом, напоминающим змею, которое ползает по плоскости (как правило, ограниченной стенками), собирая еду (или другие предметы), избегая столкновения с собственным хвостом и краями игрового поля. Каждый раз, когда змея съедает кусок пищи, она становится длиннее, что постепенно усложняет игру

Цель данного курсового проекта – получить практические навыки по разработке игры на языке C++.

Для достижения поставленной цели необходимо решить следующие задачи:

1. Изучить функциональные возможности языка C++;
2. Определить, какая игра будет реализована на данном языке;
3. Обозначить правила игры, сюжет или сценарий;
4. Обозначить элементы управления игрой;
5. Спроектировать концепцию и алгоритмы, необходимые для работы игры;
6. Реализовать алгоритмы в среде Visual Studio 2019;
7. Организовать простой и удобный интерфейс;

# **1 ОПИСАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ**

Разрабатываемая мной игра «Python», является одним из многих вариантов реализации классической игры «Змейка».

Игра начинается с запуска игры. Над игровым полем перед пользователем представлены три кнопки:

1. Открытие меню;
2. «Справка» - вывод правил игры;
3. «Выход» - осуществляет выход из игры и закрытие формы;

Меню состоит из нескольких вложенных блоков, которые будут описаны более подробно ниже.

В процессе игры змейка ползает по полю и собирает генерируемые объекты, то есть, поедает их, за что начисляются очки. С увеличением счёта очков тело змейки увеличивается на одно значение.

# **2 ОБЗОР СУЩЕСТВУЮЩИХ МЕТОДОВ РЕШЕНИЯ ЗАДАЧИ**

В процессе разработки данной игры были использованы следующие технологии:

1. Управляемые классы, которые хранят в себе методы, обеспечивающие работу игры, конструкторы классов;
2. Цикл for, а также алгоритмическое ветвление с использованием операторов if, else if, else – case конструкций;
3. Стандартные контейнеры WinForms. Контейнеры в WinForms – это управляемые классы с внутренней реализованной логикой и методами для взаимодействия с ними;
4. Элементы WinForms. Специальные объекты в среде WinForms, позволяющие создавать в конструкторе форм приложения. К ним относятся PictureBox, Form, TextLabel, Button, CheckBoxGroup, Timer и другие;
5. Фрэймворк BunifuUI – позволяющий эффективно настраивать нужные параметры WinForms и добавляющий новые настроенные управляемые элементы для создания приложений. Фрэймворк был использован в игре с несколькими целями: обеспечить возможность передвижения нестандартных форм, создать красивый дизайн приложения, добавить некоторые новые элементы, которые не содержатся по-умолчанию в среде WinForms;
6. Таймеры от WinForms. Специальный элемент, который отсчитывает задаваемый пользователем интервал, по истечению которого будет вызвано событие. Такая конструкция удобна тем, что добавляет много возможностей в управление игрой. В моей игре Таймеры используются для обновления кадров игры, для задания времени жизни объекта, для задержки в спавна игровых объектов;
7. Методы работы с файлами. В языке есть встроенные возможности для открытия и закрытия файлов в потоке, что позволяет записывать в файл нужные данные, стирать из файла нужные данные, сохранять настройки игры и прочее;

# **3 СЦЕНАРИЙ ИГРЫ, ЭЛЕМЕНТЫ УПРАВЛЕНИЯ**

## **3.1 Сценарий игры**

Змеек осталось совсем мало. Лишь остатки былого клана бродят по миру в поисках пропитания. Один из выживших – Питон, заточенный в пустыне. Его окружает ограда, столкнувшись с которой он не может совладать со своей скоростью и начинает метаться из стороны в сторону, больше и больше набирая скорость. Злополучная клетка находится посередине пустыни и главная опасность для Питона – жажда.

Нужно помочь Питону не умереть от обезвоживания. Соберите как можно больше воды для Питона.

## **Элементы управления**

Стрелки вправо, влево, вверх, вниз – управление движением вправо, влево, вверх и вниз, соответственно. Для работы с меню используется левая кнопка мыши.

## **3.3 Функциональные возможности**

Мы можем изменять положение змейки путём управления с клавиатуры, можем генерировать игровые объекты (включая змейку) в рандомных позициях.

В игре можно замораживать текущие состояния объектов, увеличивать скорость змейки и игры в целом.

# **4 СТРУКТУРА ПРОГРАММЫ**

## **4.1 Блок-схема программы**

На рисунке 1 изображена программа в виде схемы, которая отображает игровой процесс. На схеме не показано меню.

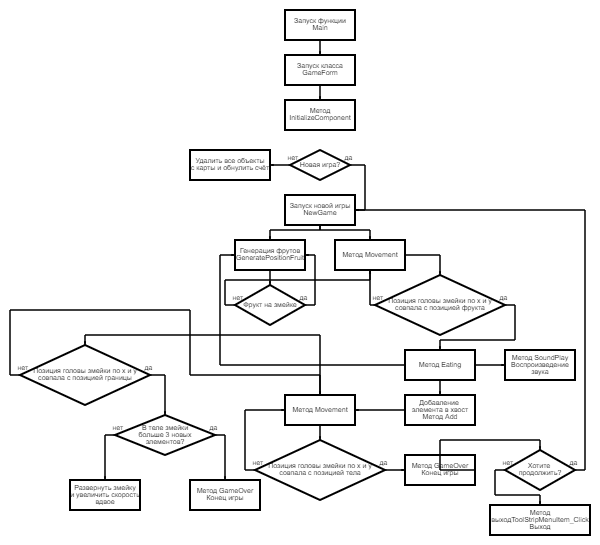


Рисунок 1 – блок-схема программы

# **5 ЛИСТИНГ ПРОГРАММЫ**

## **5.1. GameForm.cpp**

#include <iostream>

#include <Windows.h>

#include <vector>

#include "GameForm.h"

using namespace System;

using namespace System::Windows::Forms;

[STAThreadAttribute]

void Main(array<String^>^ args)

{ // Базовые настройки для запуска формы

Application::EnableVisualStyles();

Application::SetCompatibleTextRenderingDefault(false);

// Запуск указанной формы

Python::GameForm form;

Application::Run(% form);

}

// Структура, которую удобно использовать в 2д пространстве

struct Vector2

{

int X, Y;

};

Vector2 direction; // Направление змейки

Vector2 positionFruit; // Позиция фрукта

Vector2 gameArea; // Игровая зона

// Конструктор формы

Python::GameForm::GameForm(void)

{

InitializeComponent();

// Задаётся размер игровой зоны

gameArea.X = 500;

gameArea.Y = 500;

firstLaunch = true; // Первый запуск

NewGame(); // Запуск новой игры

}

void Python::GameForm::GeneratePositionFruit()

{

// Генерация случайной позиции фрукта

Random^ rand = gcnew Random();

positionFruit.X = rand->Next(17, gameArea.X); // 17 - смещение относительно боковых границ формы

positionFruit.Y = rand->Next(150, gameArea.Y); // 150 - смещение относительно верхней границы формы

// Проверка, чтобы фрукты не генерировались на теле змейки

for (int i = 0; i < score; i++) // Проход по всем элементам змейки

{

if (positionFruit.X == Serpens[i]->Location.X &&

positionFruit.Y == Serpens[i]->Location.Y) // Если при создании фрукта местоположение совпало со змейкой

{

GeneratePositionFruit(); // Фрукт пересоздаётся

}

}

// Преобразование текущей позиции фрукта, относительно шага змейки

int tempX = positionFruit.X % step;

positionFruit.X -= tempX;

int tempY = positionFruit.Y % step;

positionFruit.Y -= tempY;

// Присваивание фрукту позиции

fruit->Location = Point(positionFruit.X, positionFruit.Y);

// Добавление фрукта на карту

this->Controls->Add(fruit);

}

void Python::GameForm::Eating()

{

// Проверка позиции головы змеи и позиции фрукта

if (Serpens[0]->Location.X == positionFruit.X && Serpens[0]->Location.Y == positionFruit.Y) // Если голова совпадает по позиции с фруктом

{

labelScore->Text = "Счёт: " + ++score; // Увеличивается счёт

// Добавление нового элемента в хвост змейки

Serpens[score] = gcnew PictureBox(); // Инициализация нового элемента

Serpens[score]->Location = Point(Serpens[score - 1]->Location.X + step \* direction.X, Serpens[score - 1]->Location.Y - step \* direction.Y); // задать позицию относительно позиции предыдущего элменета + шаг и направление

Serpens[score]->BackColor = Color::Red; // цвет

Serpens[score]->Width = step; // ширина

Serpens[score]->Height = step; // высота

this->Controls->Add(Serpens[score]); // добавление нового элемента хвоста на поле

SoundPlay(); // Звук питья

GeneratePositionFruit(); // Сгенерировать новый фрукт (воду)

}

}

void Python::GameForm::Movement()

{

// Двигать каждый элемент змейки

for (int i = score; i >= 1; i--) // проход от последнего элемента к предпоследнему с начала

{

Serpens[i]->Location = Serpens[i - 1]->Location; // Каждый элемент змейки = позиции предыдущего

}

Serpens[0]->Location = Point(Serpens[0]->Location.X + direction.X \* step, Serpens[0]->Location.Y + direction.Y \* step); // Голова змейки + новая позиция относительно старой + направление \* на шаг

}

void Python::GameForm::EatYourself()

{

// проход по змейке

for (int i = 1; i < score; i++)

{

if (Serpens[0]->Location == Serpens[i]->Location) // Если голова попала локацией с хвостом

{

GameOver(); // игра окончена

}

}

}

void Python::GameForm::GameOver()

{

play = true;

die = true;

labelGameOver->Visible = true;

}

void Python::GameForm::NewGame()

{

// если игра не первая, с карты удаляются все объекты

if (!firstLaunch)

{

this->Controls->Remove(fruit);

for (int i = 0; i <= score; i++)

{

this->Controls->Remove(Serpens[i]);

}

score = 0;

}

else

{

firstLaunch = false;

}

// Инициализация змейки

Serpens = gcnew array <PictureBox^, 1>(400);

Serpens[0] = gcnew PictureBox();

Serpens[0]->Location = Point(200, 200);

Serpens[0]->BackColor = Color::DarkRed;

Serpens[0]->Width = step;

Serpens[0]->Height = step;

score = 0;

this->Controls->Add(Serpens[0]);

// Инициализация фрукта

fruit = gcnew PictureBox();

fruit->BackColor = Color::SkyBlue;

fruit->Width = step;

fruit->Height = step;

GeneratePositionFruit();

// задать интервал обновления и запустить таймер для обновления

timer->Interval = updateInterval;

timer->Start();

// задать направление при запуске

direction.X = 1;

direction.Y = 0;

play = true;

die = false;

labelScore->Text = "Счёт: 0";

labelGameOver->Visible = false;

groupBoxSettings->Visible = false;

}

void Python::GameForm::CheckBorders()

{

if (Serpens[0]->Location.X >= RightBorder->Location.X || Serpens[0]->Location.X <= LeftBorder->Location.X) // Если змейка врезалась в бортик

{

direction.X \*= -2; // Развернуть в обратном направлении и вдвое увеличить скорость

}

if (Serpens[0]->Location.Y <= UpperBorder->Location.Y || Serpens[0]->Location.Y >= LowerBorder->Location.Y) // Если змейка врезалась в бортик

{

direction.Y \*= -2; // Развернуть в обратном направлении и вдвое увеличить скорость

}

}

void Python::GameForm::SoundPlay()

{

soundEat = gcnew System::Media::SoundPlayer("..\\Resources\\3.wav");

soundEat->Play();

}

System::Void Python::GameForm::выходToolStripMenuItem\_Click(System::Object^ sender, System::EventArgs^ e)

{

Application::Exit();

}

System::Void Python::GameForm::справкаToolStripMenuItem\_Click(System::Object^ sender, System::EventArgs^ e)

{

MessageBox::Show("Правила игры в Питона:\n1. Для управления движением Питона используйте соответствующие стрелки;\n2. Вам нужно съесть наибольшее количество фруктов;\n3. Если Питон съест себя - конец игры;\n4. Если Питон врежется в ограждение - его скорость увеличится вдвое;");

return System::Void();

}

System::Void Python::GameForm::новаяИграаToolStripMenuItem\_Click(System::Object^ sender, System::EventArgs^ e)

{

NewGame();

return System::Void();

}

System::Void Python::GameForm::паузапродолжитьToolStripMenuItem\_Click(System::Object^ sender, System::EventArgs^ e)

{

if (play)

{

play = false;

}

else

{

play = true;

timer->Start();

}

return System::Void();

}

System::Void Python::GameForm::настройкиToolStripMenuItem\_Click(System::Object^ sender, System::EventArgs^ e)

{

if (groupBoxSettings->Visible == false)

{

play = false;

buttonApplySpeed->Enabled = true;

numericUpDownSpeed->Enabled = true;

groupBoxSettings->Visible = true;

}

else

{

play = true;

timer->Start();

buttonApplySpeed->Enabled = false;

numericUpDownSpeed->Enabled = false;

groupBoxSettings->Visible = false;

};

}

System::Void Python::GameForm::buttonApplySpeed\_Click(System::Object^ sender, System::EventArgs^ e)

{

updateInterval = Convert::ToSingle(numericUpDownSpeed->Value);

timer->Interval = updateInterval;

buttonApplySpeed->Enabled = false;

numericUpDownSpeed->Enabled = false;

groupBoxSettings->Visible = false;

play = true;

timer->Start();

return System::Void();

}

System::Void Python::GameForm::GameForm\_KeyDown(System::Object^ sender, System::Windows::Forms::KeyEventArgs^ e)

{

if (e->KeyCode.ToString() == "Right")

{

direction.X = 1;

direction.Y = 0;

}

else if (e->KeyCode.ToString() == "Left")

{

direction.X = -1;

direction.Y = 0;

}

else if (e->KeyCode.ToString() == "Up")

{

direction.Y = -1;

direction.X = 0;

}

else if (e->KeyCode.ToString() == "Down")

{

direction.Y = 1;

direction.X = 0;

}

return System::Void();

}

void Python::GameForm::GameForm\_Update(Object^ obgect, EventArgs^ e)

{

if (!die && play)

{

Movement();

Eating();

EatYourself();

CheckBorders();

}

else if (die && play)

{

timer->Stop();

MessageBox::Show("GAME OVER");

}

else if (!play && !die)

{

timer->Stop();

MessageBox::Show("STOP");

}

}

## **5.2. GameForm.h**

#pragma once

namespace Python {

using namespace System;

using namespace System::ComponentModel;

using namespace System::Collections;

using namespace System::Windows::Forms;

using namespace System::Data;

using namespace System::Drawing;

/// <summary>

/// Сводка для GameForm

/// </summary>

public ref class GameForm : public System::Windows::Forms::Form

{

public:

GameForm(void);

protected:

/// <summary>

/// Освободить все используемые ресурсы.

/// </summary>

~GameForm()

{

if (components)

{

delete components;

}

}

private: System::Windows::Forms::MenuStrip^ menuStrip1;

protected:

private: System::Windows::Forms::ToolStripMenuItem^ менюToolStripMenuItem;

private: System::Windows::Forms::ToolStripMenuItem^ новаяИграаToolStripMenuItem;

private: System::Windows::Forms::ToolStripMenuItem^ паузапродолжитьToolStripMenuItem;

private: System::Windows::Forms::ToolStripMenuItem^ настройкиToolStripMenuItem;

private: System::Windows::Forms::ToolStripMenuItem^ справкаToolStripMenuItem;

private: System::Windows::Forms::ToolStripMenuItem^ выходToolStripMenuItem;

private: System::Windows::Forms::GroupBox^ groupBoxGameData;

private: System::Windows::Forms::Label^ labelScore;

private: System::Windows::Forms::GroupBox^ groupBoxSettings;

private: System::Windows::Forms::Button^ buttonApplySpeed;

private: System::Windows::Forms::NumericUpDown^ numericUpDownSpeed;

private: System::Windows::Forms::Label^ label2;

private: System::Windows::Forms::PictureBox^ LowerBorder;

private: System::Windows::Forms::PictureBox^ UpperBorder;

private: System::Windows::Forms::PictureBox^ RightBorder;

private: System::Windows::Forms::PictureBox^ LeftBorder;

private: System::Windows::Forms::Timer^ timer;

private: System::Windows::Forms::Label^ labelGameOver;

private: System::ComponentModel::IContainer^ components;

private:

/// <summary>

/// Обязательная переменная конструктора.

/// </summary>

Далее, с 63 по 328 строки, код, который автоматически генерирует VisualStudio при создании формы.

#pragma endregion

// Данные игры

private: PictureBox^ fruit; // фрукт

private: array<PictureBox^>^ Serpens; // змея

private: System::Media::SoundPlayer^ soundEat; // звук жевания

private: bool play; // играть

private: bool die; // смерть

private: bool firstLaunch; // первый запуск

private: int step = 10; // шаг

private: float updateInterval = 100; // интервал обновления

private: int score = 0; // счёт

// Действия игры

private: void GeneratePositionFruit(); // Генерация позиции фрукта

private: void Eating(); // Питон съедает фрукт

private: void Movement(); // Движение Питона

private: void EatYourself(); // Питон съел себя

private: void GameOver(); // Конец игры

private: void NewGame(); // Новая игра

private: void CheckBorders(); // Проверка, не врезалась ли змейка в бортики

private: void SoundPlay(); // Воспроизведение аудио

// Обработчики нажатия на кнопки

private: System::Void выходToolStripMenuItem\_Click(System::Object^ sender, System::EventArgs^ e);

private: System::Void справкаToolStripMenuItem\_Click(System::Object^ sender, System::EventArgs^ e);

private: System::Void новаяИграаToolStripMenuItem\_Click(System::Object^ sender, System::EventArgs^ e);

private: System::Void паузапродолжитьToolStripMenuItem\_Click(System::Object^ sender, System::EventArgs^ e);

private: System::Void настройкиToolStripMenuItem\_Click(System::Object^ sender, System::EventArgs^ e);

private: System::Void buttonApplySpeed\_Click(System::Object^ sender, System::EventArgs^ e);

// Отслеживание нажатой клавиши

private: System::Void GameForm\_KeyDown(System::Object^ sender, System::Windows::Forms::KeyEventArgs^ e);

// Событие обновления игры

private: void GameForm\_Update(Object^ obgect, EventArgs^ e);

};

}