**РУКОВОДСТВО ПРОГРАММИСТА**

**Игра**

**«Змейка»**

2022

**АННОТАЦИЯ**

В данном программном документе приведено руководство программиста по работе с Visual Studio, в которой был разработана «Змейка»

В разделе «Назначение и условия применения программы» указаны назначение и функции, выполняемые программой, условия, необходимые для правильной работы игры.

В разделе «Характеристика программы» приведено описание основных характеристик и особенностей программы.

1. Назначение и условия применения программы

С развитием технологий компьютеры все больше и больше вливаются в повседневную жизнь человека. Персональные компьютеры - это уже не просто машины для математических вычислений, их общедоступность значительно расширила с помощью области игровой индустрии. Индустрия игр быстро набирала обороты, стремительно развивалась, завоевываю все большую и большую популярность у подрастающего поколения. И если первые игры отличались простым дизайном и логикой, то современная компьютерная игра - очень сложная программа и так же дело рук большого коллектива разработчиков. Из-за этого у не которых пользователей ПК нету возможности играть более масштабные игры, поэтому для этого существуют аркадные игры где на мощность компьютера не сильно влияет, и пользователь может в них поиграть. В таком случае из данной вытекающей проблемы была реализована игра «Змейка».

С помощью игры «Змейка» пользователь сможет насладится игровым процессом на более не мощном компьютере и получить заряд положительных эмоций, но также главной целью было создание игры, которая в дальнейшем была реализована.

1. Характеристики и работа программы

**Требования к реализации функций в игре:**

Основная задача игры заключается в том, чтобы пользователь набрал наибольшее количество очков.

Взаимодействие со змейкой будет осуществляется с помощью клавиш «w», «a», «d» на клавиатуре.

• Пользователь нажмет на клавишу «w» и змейка начнет движение по карте.

• Пользователь нажмет на клавишу «a» и змейка повернет влево

• Пользователь нажмет на клавишу «d» и змейка повернет направо

Каждое съеденное яблоко приносит пользователю 100 очков. Игра завершается тогда, когда змейка сталкивается со стенкой или сталкивается со своим хвостом.

**В игре должны присутствовать такие функций:**

● Начать игру – кнопкой “Запуск” в главном меню.

● Выбрать уровень сложности («случайный», «легкий», «средний», «сложный») – кнопкой «Уровень» в главном меню.

* Взаимодействие со змейкой будет осуществляется с помощью клавиш «w», «a», «d» на клавиатуре.

● Выбрать скорость передвижения змейки (от 1 до 50) – кнопкой «Скорость» в главном меню.

● Включить или отключить звук с помощью кнопки «Отключить звук».

● Закрыть приложение с помощью крестика в правом верхнем углу.

● Выбрать количество яблок (от 1 до 50)-кнопкой «Кол-во Яблок» в главном меню.

● Просмотреть таблицу рекордов в главном меню.

● Обновить таблицу рекордов –кнопкой «Обновить».

При создании игры использовался язык программирования Visual Studio.

На данном рисунке предоставлена папка «scr» где храниться исходник проекта.

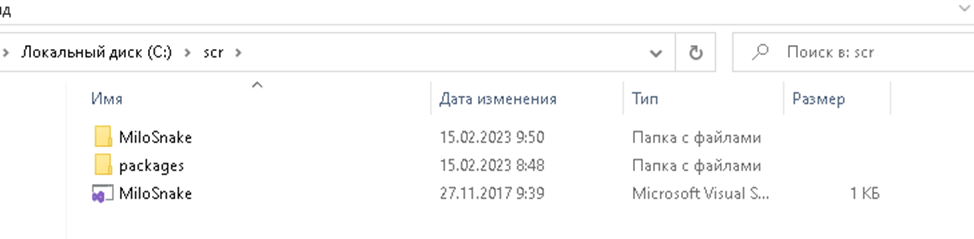


Рисунок 1 – Папка с исходником проекта

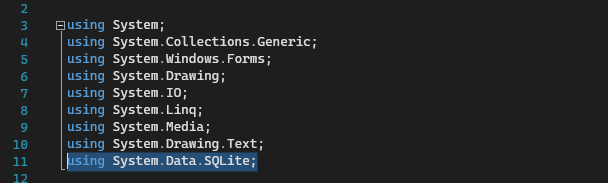


Рисунок 2-Подключение директивы using

На рисунке показана что директива using позволяет использовать типы, определенные в пространстве имен, без указания полного пространства имен этого типа.

Ниже можно увидить подключение бд sql c с которой программа будет работать.

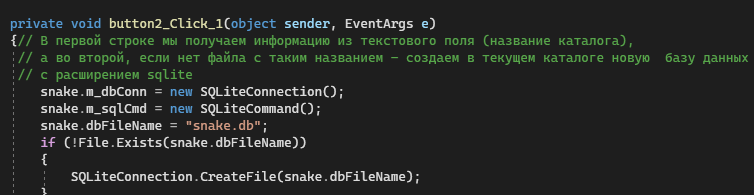


Рисунок 3- подключение базы sql

Далее рассмотрим создание меню в игре. В первую очередь был проработан экземпляр классов и событие на кнопку esc.

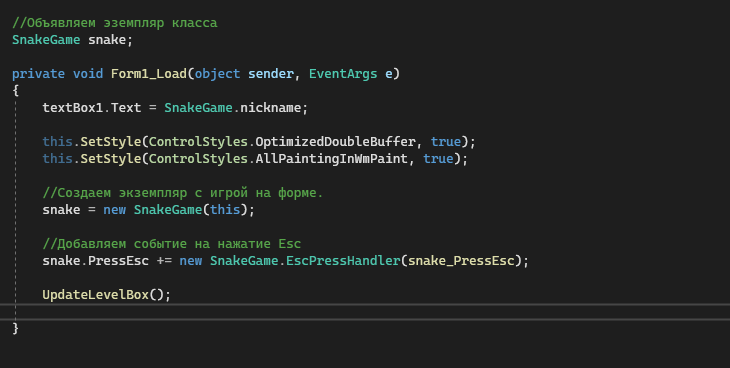


Рисунок 4- Проработка экземпляра класса и события на кнопку esc

Н данном рисунки указывается создание надписи «Уровень» и кнопку «Случайный»

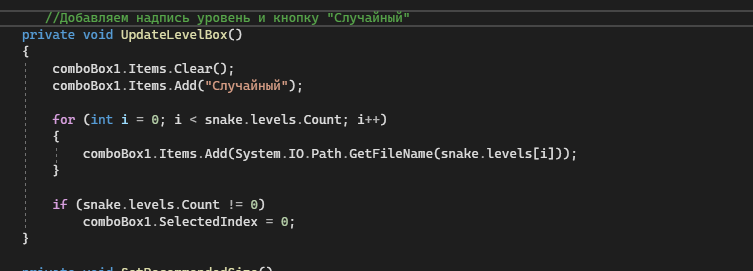


Рисунок 5- Создание надписи «Уровень» и кнопки «Случайный»

На рисунке 5 демонстрируется создание количество яблок и выставление скорости с проигрыванием музыки.

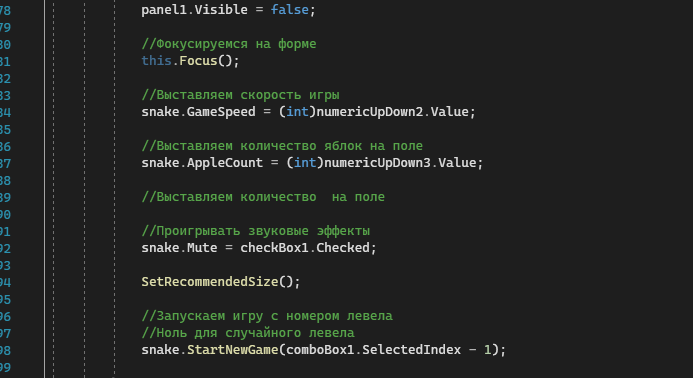


Рисунок 5- Создаем функций по количество яблок и выбора скорости с проигрыванием музыки.

ЛИСТИНГ ПРОГРАММЫ

«Игровые классы»

#region Игровые классы

public String dbFileName;

public SQLiteConnection m\_dbConn;

public SQLiteCommand m\_sqlCmd;

//Определяем набор возможных направлений движения нашей змейки

enum Direction

{

Up,

Down,

Left,

Right

}

//Базовый класс - игровой объект

//Любой объект на игровом поле имеет позицию (координаты) и картинку (текстуру)

private class GameObject

{

//Конструктор

public GameObject(int x, int y, int imgN)

{

this.Pos = new Point(x, y);

this.imgIndex = imgN;

}

//Свойства класса

//Виртуальная позиция изображения, реальня позиция объекта

private Point pos;

public Point Pos

{

get

{

return pos;

}

set

{

pos = value;

//При изменении свойства, рассчитываем реальную позицию изображения на игровом поле (координаты с учетом размера пикселя)

CalculateImagePos();

}

}

public void CalculateImagePos()

{

this.imgPos = new Point(pos.X \* PixelLen, pos.Y \* PixelLen);

}

public int imgIndex { set; get; }

public Point imgPos { set; get; }

}

//Сегмент змейки, наследник базового класса GameObject

private class SnakeSegment : GameObject

{

public SnakeSegment(int x, int y, int imgN, Direction dir): base(x, y, imgN)

{

//Добавляем всего одной свойство - направление, в которое повернута текстура

Dir = dir;

}

//Изрбражение (текстуру) достаем из списка, по номеру

public void ChangeImg(int start)

{

//start - указывает на тип сегмента (позиция в списке)

//0-3 голова, 4-7 тело, 8-11 хвост, 12-15 поворот

switch (Dir)

{

case Direction.Down:

imgIndex = start;

break;

case Direction.Right:

imgIndex = start + 1;

break;

case Direction.Left:

imgIndex = start + 2;

break;

case Direction.Up:

imgIndex = start + 3;

break;

}

}

//Метод для удобного копирования сегмента

public SnakeSegment Copy()

{

return new SnakeSegment(Pos.X, Pos.Y, imgIndex, Dir);

}

//Единственное свойство наследника - направление, в которое повернута текстура

public Direction Dir { set; get; }

}

//Класс для сериализации информации об уровне в бинарный файл

//Сохраняем и загружаем уровни с использованием этого класса

[Serializable]

class GameLevel

{

//Конструктор

public GameLevel(Point start, Direction dir, List<Point> rocks)

{

this.Start = start;

this.Dir = dir;

this.Rocks = rocks;

}

//Всего 3 свойства. Стартовая позиция змеи, ее направление, список координат камней

public Direction Dir { set; get; }

public Point Start { set; get; }

public List<Point> Rocks { set; get; }

}

**«Настройка интерфейса и проигрывание музыки»**

//Исходные картинки текстур, их изменять не будем

private static Image appleScr = new Bitmap(texturePath + "apple.png");

private static Image grassScr = new Bitmap(texturePath + "grass.jpg");

private static Image rocks\_texture = new Bitmap(texturePath + "rocks.png");

private static Image snake\_texture = new Bitmap(texturePath + "snake\_texture.png");

private static Image chicken\_texture = new Bitmap(texturePath + "chicken.png");

//С этими картинками работаем

private static Image appleImg;

private static Image grassImg;

private static Image background;

//Список рисунков разных камней, для разнообразия

private static List<Image> rocksTexList = new List<Image>();

//Список изображений, в котором текстуры змейки.

//4 разных сегмента, по 4 копии для каждого направления + 4 копии головы для эффекта смерти

private static List<Image> snakeTexList = new List<Image>();

//Список изображений, в котором хранятся текстуры курицы, 4 копии для каждого направления

private static List<Image> chickenTexList = new List<Image>();

//Змейка состоит из списка сегментов, каждый сегмент должен иметь свои координаты.

//Представим сегмент точкой, а саму змейку - списком точек.

private List<SnakeSegment> snake = new List<SnakeSegment>();

//Список объектов, камней, сюда загружаются координаты камней из левелов

private List<GameObject> rocks = new List<GameObject>();

//Еда для змейки, список яблок и куриц

private List<GameObject> apples = new List<GameObject>();

private List<GameObject> chickens = new List<GameObject>();

//Список строк, тут храним левелы (расположение камней)

public List<string> levels = new List<string>();

//Рекорды

public List<string> record = new List<string>();

//Хранит номер текущего уровня

private int currentLev = 0;

//Текущее направление движения, по умолчанию вниз

private Direction direction = Direction.Down;

//Таймер, по которому будет происходить обновление позиции (координат сегментов) нашей змейки

private Timer gameTimer = new Timer();

//Логическое значение, отвечающее за смерть змейки.

private bool gameOver = false;

//Фиксированный размер игрового поля в пикселях, его не меняем.

private static Size bounds = new Size(40, 30);

//Реальный размер игрового поля, рассчитывается умножением размера игрового поля на размер пикселя

private static Size realSize;

//Точка по которой сдвигается игровое поле при отрисовке (нужно для установки области отрисовки по центру формы)

private static Point transform = new Point(0, 0);

//Очки

private int gameScore = 0;

//Формат строки для отрисовки текста

private StringFormat sf = new StringFormat();

//Шрифт, которым будет рисоваться текст

private Font drawFont;

//Центр игрового поля, для рисования текста

private static Point screenCenter;

//Рандомайзер

private Random random = new Random();

//Через плеер будем проигрывать wave-файлы

private SoundPlayer appleCrunch = new SoundPlayer(soundsPath + "apple.wav");

private SoundPlayer chickenScream = new SoundPlayer(soundsPath + "chicken.wav");

private SoundPlayer hitSound = new SoundPlayer(soundsPath + "hit.wav");

//Событие, которое будет вызываться по нажатию на клавишу Esc

public event EscPressHandler PressEsc;

public EventArgs e = null;

public delegate void EscPressHandler(object sg, EventArgs e);

//Константы, указывающие в папку с ресурсами

private const string texturePath = "data/textures/";

private const string levelPath = "data/levels/";

private const string soundsPath = "data/sounds/";

//Константы с текстом

private const string overText = "Игра окончена!";

private const string infoText = "Нажмите R, чтобы начать уровень заново\r\nили Esc, чтобы вернуться к выбору уровня.";

private const string levelExt = ".msl";

private const string dialogFilt = "Файлы уровней|\*" + levelExt;

//Диалоги для сохарения и открытия файла уровня

private SaveFileDialog saveFileDialog = new SaveFileDialog();

private OpenFileDialog openFileDialog = new OpenFileDialog();

//Указывает на состояние, запущен редактор или игра

private bool isEditor = false;

**«Свойств класса»**

//Канвас - холст, на котором будем рисовать игру.

private Control canvas { set; get; }

//Показывается ли игровое поле, требуется для события изменения размера формы

private bool GameShowed { set; get; }

//Выключен звук или нет

public bool Mute { get; set; }

//Скорость игры

private int gameSpeed = 10;

public int GameSpeed

{

get

{

return gameSpeed;

}

//При назначении свойства меняем интервал игрового таймера

set

{

gameSpeed = value;

gameTimer.Interval = 1000 / gameSpeed;

}

}

//Количество яблок на поле

private int appleCount = 10;

public int AppleCount

{

get

{

return appleCount;

}

//При изменение свойства делаем проверку, ограничиваем максимальное количество

set

{

if (value < 50)

appleCount = value;

else

appleCount = 50;

}

}

//Количество куриц на поле

private int chickCount = 5;

public int ChickCount

{

get

{

return chickCount;

}

//При изменение свойства делаем проверку, ограничиваем максимальное количество

set

{

if (value < 50)

chickCount = value;

else

chickCount = 50;

}

}

//Размер пикселя, из которых состоит игровое поле

private static int pixelLen = 20;

private static int PixelLen

{

get

{

return pixelLen;

}

set

{

//Во избежание проблем с мизерным скейлингом

if (value == 0)

pixelLen = 1;

else

pixelLen = value;

}

}

«Запуск игры»

//Инициализация класса

public SnakeGame(Control control)

{

try

{

//Предаем форму в канвас, добавляем на него события отрисовки и нажатия клавишей

canvas = control;

canvas.Paint += new PaintEventHandler(canvas\_Paint);

canvas.KeyDown += new KeyEventHandler(canvas\_KeyDown);

canvas.KeyUp += new KeyEventHandler(canvas\_KeyUp);

canvas.Resize += new EventHandler(canvas\_Resize);

canvas.MouseDown +=new MouseEventHandler(canvas\_MouseDown);

canvas.MouseMove += new MouseEventHandler(canvas\_MouseMove);

//Определяем интервал игрового таймера и событие на тик (обновление позиции змейки)

//Интервал определяет скорость змейки. Чем он выше, тем змейка медленее двигается

gameTimer.Tick += new EventHandler(gameTimer\_Tick);

gameTimer.Interval = 1000 / gameSpeed;

//Задаем свойства формата для рисования выравнивания текста строго по центру

sf.Alignment = StringAlignment.Center;

sf.LineAlignment = StringAlignment.Center;

//Задаем свойства диалогам сохранения и открытия файлов уровня

saveFileDialog.InitialDirectory = levelPath;

saveFileDialog.Title = "Сохранить уровень как...";

saveFileDialog.Filter = dialogFilt;

saveFileDialog.DefaultExt = levelExt;

saveFileDialog.AddExtension = true;

openFileDialog.InitialDirectory = levelPath;

openFileDialog.Title = "Открыть уровень";

openFileDialog.Filter = dialogFilt;

openFileDialog.DefaultExt = levelExt;

openFileDialog.AddExtension = true;

//Предзагрузка звуков

chickenScream.Load();

appleCrunch.Load();

hitSound.Load();

//Загружаем левелы в список строк

OpenLevelFiles();

}

//Ловим ошибку, в случае, если потеряна папка с данными

catch (Exception ex)

{

MessageBox.Show(ex.Message, ex.Source, MessageBoxButtons.OK, MessageBoxIcon.Error);

Application.Exit();

}

}

//Загружаем левелы из ресурсов в список строк

private void OpenLevelFiles()

{

levels.Clear();

//Ищем текстовые файлы в папке с левелами

var txtFiles = Directory.EnumerateFiles(levelPath, "\*.msl");

//Пробегаемся по списку файлов, добавляем их содержимое в список levels

foreach (var item in txtFiles)

{

levels.Add(item);

}

}

public void StartNewGame(int level)

{

//Свойство, показывающее, показывается ли игровое поле, требуется для события изменения размера формы

GameShowed = true;

//Инициализация игрового поля

//Расчет размеров текстур, положения поля на форме

InitGameField();

//Запускаем игру

StartGame(level, false);

}

//Инициализация игрового поля

//Расчет размеров текстур, положения поля на форме

private void InitGameField()

{

var canW = canvas.ClientSize.Width;

var canH = canvas.ClientSize.Height;

//Определяем размер пикселя, в зависимости от размера холста

if (canW < canH)

{

PixelLen = canW / bounds.Width;

}

else

{

PixelLen = canH / bounds.Height;

}

//Реальный размер игрового поля, фиксированный размер поля, умножаем на размер пикселя

realSize = new Size(bounds.Width \* PixelLen, bounds.Height \* PixelLen);

//Рассчитываем точку для смещения игрового поля в центр формы (холста)

transform = new Point(canW / 2 - realSize.Width / 2, canH / 2 - realSize.Height / 2);

//Задаем размер шрифта, равный размеру пикселя

drawFont = new Font("Arial", pixelLen);

//Изменяем размеры текстур в зависимости от размера пикселя

appleImg = Utils.ResizeImage(appleScr, PixelLen, PixelLen);

grassImg = Utils.ResizeImage(grassScr, realSize.Width, realSize.Height);

//Для удобства, заранее определяем центр игрового поля

screenCenter = new Point(realSize.Width / 2, realSize.Height / 2);

//Загружаем текстуру камней в список

LoadRocksTex();

//Загружаем текстуру змеи в список

LoadSnakeTex();

//Загружаем текстуру курицы в список

LoadChickenTex();

}

//Дербаним текстуру на 5 частей

private void LoadRocksTex()

{

rocksTexList.Clear();

var sourceWidth = rocks\_texture.Width;

//В цикле, выполняем 5 раз

for (int i = 0; i < 5; i++)

{

//Определяем область для вырезания, квадрат, в цикле меняется его позиция только по Y

Rectangle cropArea = new Rectangle(0, i \* sourceWidth, sourceWidth, sourceWidth);

//Изменяем размер картинки в соответсивии с размером пикселя

var source = Utils.ResizeImage(Utils.CropImage(rocks\_texture, cropArea), PixelLen, PixelLen);

//Добавляем в список

rocksTexList.Add(source);

}

}

//Дербаним картинку текстуры на куски, размером с пиксель, копируем каждый с поворотом

//Всего 4 области (голова, тело, хвост, поворот), по 4 копии на каждое направление, всего в списке 16 изображений

private void LoadSnakeTex()

{

snakeTexList.Clear();

var sourceWidth = snake\_texture.Width;

//Пробегаеся по картинке текстуры, снизу вверх. Внизу голова, сверху поворот.

for (int i = 3; i >= 0; i--)

{

//Определяем область для вырезания, квадрат, в цикле меняется его позиция только по Y

Rectangle cropArea = new Rectangle(0, i \* sourceWidth, sourceWidth, sourceWidth);

//Первая картника, которая уже смотрит вниз, ее вращать не требуется

var source = Utils.ResizeImage(Utils.CropImage(snake\_texture, cropArea), PixelLen, PixelLen);

//Делаем 3 копии первой картинки, их будем вращать

var source270 = new Bitmap(source);

var source90 = new Bitmap(source);

var source180 = new Bitmap(source);

//Вращаем по остальным направлениям

//Смотри вправо

source270.RotateFlip(RotateFlipType.Rotate270FlipNone);

//Смотрит влево

source90.RotateFlip(RotateFlipType.Rotate90FlipNone);

//Смотрит вверх

source180.RotateFlip(RotateFlipType.Rotate180FlipNone);

//Добавляем в список 4 копии текстуры, каждая повернута в своем направлении

snakeTexList.Add(source);

snakeTexList.Add(source270);

snakeTexList.Add(source90);

snakeTexList.Add(source180);

}

//Пачка изображений для создания эффекта сплющенной от удара головы

int halfPix = PixelLen / 2;

//Во избежание проблем в мизерном скейлинге

if (halfPix == 0)

halfPix = 1;

//Добавляем 4 копии головы (по 1й для каждого направления) для создания эффекта смерти

snakeTexList.Add(new Bitmap(snakeTexList[0], new Size(PixelLen, halfPix)));

snakeTexList.Add(new Bitmap(snakeTexList[1], new Size(halfPix, PixelLen)));

snakeTexList.Add(new Bitmap(snakeTexList[2], new Size(halfPix, PixelLen)));

snakeTexList.Add(new Bitmap(snakeTexList[3], new Size(PixelLen, halfPix)));

}

//Дербаним текстуру курицы на список изображений, 4 шутки, по одной на каждое направление

private void LoadChickenTex()

{

chickenTexList.Clear();

var sourceWidth = chicken\_texture.Width;

//Пробегаеся по картинке текстуры, снизу вверх.

//Всего 3 картинки. Снизу курица, смотрящая вниз, посередине курица смотрящая направо, сверху курица, смотрящая вверх.

for (int i = 2; i >= 0; i--)

{

//Определяем область для вырезания, квадрат размером с ширину текстуры, в цикле меняется его позиция только по Y

Rectangle cropArea = new Rectangle(0, i \* sourceWidth, sourceWidth, sourceWidth);

//Первая картника, которая уже смотрит вниз, ее вращать не требуется

var source = Utils.ResizeImage(Utils.CropImage(chicken\_texture, cropArea), PixelLen, PixelLen);

chickenTexList.Add(source);

//Если середина, попадаем на курицу, смотряющую вправо

if (i == 1)

{

//Копируем ее и отражаем по оси Х, получается курица, смотрящая влево

var sourceFlip = new Bitmap(source);

sourceFlip.RotateFlip(RotateFlipType.RotateNoneFlipX);

chickenTexList.Add(sourceFlip);

}

}

//В итоге, в список добавлены 4 картинки, с курицей, смотрящей во все 4 направления

}

//Запуск игры

private void StartGame(int levNum, bool skipLevelLoad)

{

//Т.к. данный метод вызывается и после смерти змейки, надо возвращать все значения в дефолт (по умолчанию).

gameOver = false;

isEditor = false;

apples.Clear();

chickens.Clear();

gameScore = 0;

//Если загрузка уровня из файла разрешена

if (!skipLevelLoad)

{

//Чистим список сегментов змейки от точек

snake.Clear();

//Загружаем список камней из уровня, добавляем змейку на стартовую позицию

//Если при загрузке уровня ошибка - выходим

if (!LoadLevel(levNum))

return;

}

//Если загрузка уровня запрещена, используются существующие списки сегментов змейки и камней

//Нужно для редактирования уровня прямо из игры (на лету)

//Текущее направление, равняем направлению головы змейки

direction = snake[0].Dir;

//Рисуем фон с камнями из левела

DrawLevel();

//Генерируем еду, точку в случайной позиции в пределах игрового поля

//Яблоки

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

{

apples.Add(GenerateFood());

}

//Курочки

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

{

//chickens.Add(GenerateFood());

}

//Запускаем таймер обновления позиции змейки и курицы

gameTimer.Start();

}

//Загружаем камни из уровня

private bool LoadLevel(int levNum)

{

//Для начала загрузим из левела список камней

//Если номер левела ноль или больше количества загруженных левелов

if ((levNum == - 1) | (levNum > levels.Count))

{

//Случайный левел

currentLev = random.Next(levels.Count);

}

else

{

//Выбираем левел по указанному номеру

currentLev = levNum;

}

//Указываем на путь к файлу

var path = levels[currentLev];

//Если файл существует, продолжаем

if (File.Exists(path))

{

try

{

//Читаем уровень из файла

ReadLevel(path);

}

catch (Exception)

{

//В случае ошибки информируем пользователя

MessageBox.Show("Поврежден файл уровня!", "Ошибка загрузки уровня", MessageBoxButtons.OK, MessageBoxIcon.Error);

//Возвращаемся в главное меню

ReturnToMenu();

return false;

}

//Уровень загружен успешно, метод возвращает истину

return true;

}

else

{

//Если файл уровня не существует, информируем пользователя

MessageBox.Show("Файл уровня не существует: " + path, "Ошибка загрузки уровня", MessageBoxButtons.OK, MessageBoxIcon.Error);

//Возвращаемся в главное меню

ReturnToMenu();

return false;

}

//Камни загружены из левела, теперь можно их нарисовать!

}

//Метод, добавляющий змейку в точку с учетом направления

private void SpawnSnake(int x, int y, Direction dir)

{

//Чистим список сегментов

snake.Clear();

//В зависимости от направления

switch (dir)

{

case Direction.Up:

//Добавляем голову, смотрящую вверх

snake.Add(new SnakeSegment(x, y, 3, dir));

//Добавляем хвост, смотрящий назад

snake.Add(new SnakeSegment(x, y + 1, 11, dir));

break;

//Остальные направления по аналогии

case Direction.Down:

snake.Add(new SnakeSegment(x, y, 0, dir));

snake.Add(new SnakeSegment(x, y - 1, 8, dir));

break;

case Direction.Left:

snake.Add(new SnakeSegment(x, y, 2, dir));

snake.Add(new SnakeSegment(x + 1, y, 10, dir));

break;

case Direction.Right:

snake.Add(new SnakeSegment(x, y, 1, dir));

snake.Add(new SnakeSegment(x - 1, y, 9, dir));

break;

}

}

//Метод для чтения уровня из файла

private void ReadLevel(string path)

{

//Чистим список камней, т.к. этот метод будет вызываться повторно

rocks.Clear();

//Создаем экземляр класса, в котором будет храниться информация об уровне

//Используем метод для загрузки бинарного файла (см. класс утилит)

GameLevel gl = (GameLevel)Utils.LoadBinary(path);

//Создаем змейку на позиции с учетом направления

SpawnSnake(gl.Start.X, gl.Start.Y, gl.Dir);

//Наполняем список камней

rocks = new List<GameObject>();

//В цикле пробегаемся по списку координат gl.Rocks

for (int i = 0; i < gl.Rocks.Count; i++)

{

//Добавляем в список камни на координатах из списка

rocks.Add(new GameObject(gl.Rocks[i].X, gl.Rocks[i].Y, 0));

}

}

//Рисуем камни на фоне

private void DrawLevel()

{

//Берем копию нашего фона, на нем будем рисовать (на исходной картинке рисовать нельзя)

background = Utils.ResizeImage(grassImg, realSize.Width, realSize.Height);

//Используем этот класс для рисования по картинке

Graphics g = Graphics.FromImage(background);

//Пробегаемся по списку камней и рисуем каждый в своей позиции

for (int i = 0; i < rocks.Count; i++)

{

g.DrawImage(rocksTexList[random.Next(5)], rocks[i].imgPos);

}

}

**«Перемещение объектов»**

//Тик игрового таймера, по нему происходит передвижение змейки и яблока

private void gameTimer\_Tick(object sender, EventArgs e)

{

if (snake.Count != 0)

{

//Передвигаем змейку на шаг в один пиксель

MoveSnake();

//Пробегаемся по списку куриц

for (int i = 0; i < chickens.Count; i++)

{

//Передвигаем курочку на шаг в один пиксель

MoveChicken(chickens[i]);

}

}

//Принудительно перерисовываем холст

canvas.Invalidate();

}

//Метод для воспроизведения звука из wave-файла

private void PlaySound(SoundPlayer sp)

{

//Если звук включен

if (!Mute)

{

sp.Play();

}

}

private void MoveSnake()

{

//Запоминаем позицию и направление головы до следующего шага

//Понадобится для определения определения направления следующего сегмента

Point oldHeadPos = snake[0].Pos;

Direction oldDir = snake[0].Dir;

//Теперь меняем позицию головы на один пиксель в сторону, в зависимости от текущего направления

switch (direction)

{

case Direction.Right:

snake[0].Pos = new Point(snake[0].Pos.X + 1, snake[0].Pos.Y);

break;

case Direction.Left:

snake[0].Pos = new Point(snake[0].Pos.X - 1, snake[0].Pos.Y);

break;

case Direction.Up:

snake[0].Pos = new Point(snake[0].Pos.X, snake[0].Pos.Y - 1);

break;

case Direction.Down:

snake[0].Pos = new Point(snake[0].Pos.X, snake[0].Pos.Y + 1);

break;

}

//Задаем направление и изображение с учетом направления

snake[0].Dir = direction;

//В методе ChangeImg, 0 - указывает на текстуру головы

snake[0].ChangeImg(0);

//В обратном цикле (начинаем с последнего элемента списка) прогоняем все точки змейки, кроме головы

for (int i = snake.Count - 1; i > 0; i--)

{

//Сдвигаем каждую точку тела на позицию предыдущей точки в списке. Получается эффект движения.

//В итоге, вторая точка займет место первой точки (головы), голову двигаем отдельно

//Если любой сегмент, кроме исключением головы и следующего после головы

if (i != 1)

{

//Двигаем сегмент на позицию предыдущего

snake[i] = snake[i - 1].Copy();

//Если попался хвост

if (i == snake.Count - 1)

{

//Определеяем направление хвоста (костыль)

if (snake.Count > 3)

snake[i].Dir = snake[snake.Count - 3].Dir;

else

snake[i].Dir = oldDir;

//Меняем изображение сегмента на хвост с учетом направления

snake[i].ChangeImg(8);

}

}

else

{

//Если первый после головы сегмент. C него копируются последующие сегменты

snake[1].Pos = oldHeadPos;

snake[1].Dir = oldDir;

//Если хвост

if (i == snake.Count - 1)

{

//Задаем ему направление головы

snake[1].Dir = snake[0].Dir;

//Меняем изображение на хвост, с учетом направления

//8 - указывает на позицию текстуры с хвостом в списке

snake[1].ChangeImg(8);

}

else

{

//Если другой сегмент, прямое тело или поворот

//Если направление головы не равно направлению первого после головы сегмента, значит змейка поворачивает!

//Определеям направление поворота, чтобы задать сегменту правильное изображение

if (snake[0].Dir != snake[1].Dir)

{

if (((oldDir == Direction.Left) && (snake[0].Dir == Direction.Down)) || ((oldDir == Direction.Up) && (snake[0].Dir == Direction.Right)))

{

//90 градусов

snake[1].imgIndex = 14;

}

else

if (((oldDir == Direction.Right) && (snake[0].Dir == Direction.Down)) || ((oldDir == Direction.Up) && (snake[0].Dir == Direction.Left)))

{

//180 градусов

snake[1].imgIndex = 15;

}

else

if (((oldDir == Direction.Right) && (snake[0].Dir == Direction.Up)) || ((oldDir == Direction.Down) && (snake[0].Dir == Direction.Left)))

{

//270 градусов

snake[1].imgIndex = 13;

}

else

//0 градусов

snake[1].imgIndex = 12;

}

else

//Если змейка не поворачивает, рисуем текстуру прямого тела с учетом направления

snake[1].ChangeImg(4);

}

}

}

//Проверяем наличие столкновений головы змейки с игровыми объектами

CheckCollisions();

}

private void CheckCollisions()

{

//Проверяем наличие столкновения головы с границами игрового поля

//Игровое поле представляет собой прямоугольник: (0, 0, bounds.Width, bounds.Height);

if (checkBounds(snake[0].Pos))

//Воткнулись в стену и умерли

Die();

//Проверяем наличие столкновений головы змейки со своим телом

for (int j = 1; j < snake.Count; j++)

{

if (snake[0].Pos == snake[j].Pos)

{

//Воткнулись в себя, умерли

Die();

}

}

//Проверяем наличие столкновения головы с камнем

//Пробегаем по списку камней. Если координаты головы и камня совпадают - смерть.

if (checkRocks(snake[0].Pos))

Die();

//Пробегаемся по списку яблок

for (int i = 0; i < apples.Count; i++)

{

//Проверяем наличие столкновения с едой

if (snake[0].Pos.X == apples[i].Pos.X && snake[0].Pos.Y == apples[i].Pos.Y)

{

//Кушаем

Eat();

//Проигрываем звук съеденного яблока

PlaySound(appleCrunch);

//Создаем новый кусок еды

apples[i] = GenerateFood();

}

}

//Пробегаемся по списку куриц

for (int i = 0; i < chickens.Count; i++)

{

if (snake[0].Pos.X == chickens[i].Pos.X && snake[0].Pos.Y == chickens[i].Pos.Y)

{

//Пищевая ценность курицы равняется 3м яблокам!

for (int j = 0; j < 3; j++)

{

Eat();

}

//Проигрываем звук умирающей курицы

PlaySound(chickenScream);

//Создаем новую курицу

chickens[i] = GenerateFood();

}

}

}

//Просто пачка переменных для хранения информации о гипотетическом курином шаге

struct ChickInfo

{

//Расстояние до головы змеи

public double Distance;

//Позиция шага в списке

public int index;

//Координаты шага

public Point pos;

}

//Двигаем курицу

private void MoveChicken(GameObject chick)

{

//Объявляем список

var posPoints = new List<ChickInfo>();

//Набиваем массив точек четярьмя возможными точками для новой позиции

Point[] plist = new Point[4];

plist[0] = new Point(chick.Pos.X, chick.Pos.Y + 1);

plist[1] = new Point(chick.Pos.X - 1, chick.Pos.Y);

plist[2] = new Point(chick.Pos.X + 1, chick.Pos.Y);

plist[3] = new Point(chick.Pos.X, chick.Pos.Y - 1);

//Проверяем в цикле, возможность "встать" в новую точку

for (int i = 0; i < 4; i++)

{

if (checkBounds(plist[i]) || checkRocks(plist[i]) || checkSnake(plist[i]))

{

//Если следующий шаг выходит за границы игрового поля, попадает на камень или на змейку

//Ничего не делаем

}

else

{

//Если шаг выпадает на свободное место

ChickInfo chickInfo;

//Курица считает расстояние от головы змейки до своего следующего шага

chickInfo.Distance = Utils.GetDistance(plist[i], snake[0].Pos);

//Запоминаем ее номер в массиве

chickInfo.index = i;

//Запоминаем точку

chickInfo.pos = plist[i];

//Добавляем структуру в список

posPoints.Add(chickInfo);

}

}

if (posPoints.Count != 0)

{

//Получим минимальную дистанцию

//Linq, медленный, но простой способ. Сортируем список по минимальной дистанции, берем первый элемент.

ChickInfo minItem = posPoints.OrderBy(p => p.Distance).First();

//Если расстояние до головы меньше 10 пикселей, курица начинает сходить с ума, убегать

if (minItem.Distance < 10)

{

//В 97 случаях из 100, двигаемся! Даем змее небольшое преимущество в скорости

if (random.Next(100) > 3)

{

//Если шагов в списке больше чем 1

if (posPoints.Count != 1)

{

//Удаляем нежелательный шаг, что сократит расстояние до головы

posPoints.Remove(minItem);

}

//Переставляем курицу

ChickStep(posPoints, chick);

return;

}

}

else

{

//Иначе, курица ведет себя спойно, клюет корм

//Изредка двигаемся

if (random.Next(100) < 10)

{

//Переставляем курицу

ChickStep(posPoints, chick);

return;

}

}

}

//Если до этого момента дошло, значит курица стоит на месте, просто поворачиваем ее текстуру

//Будто оглядывается по сторонам

if (random.Next(100) < 20)

{

chick.imgIndex = random.Next(4);

}

}

//Метод для перестановки курицы слуайную возможную позицию

private void ChickStep(List<ChickInfo> posPoints, GameObject chick)

{

var rand = posPoints[random.Next(posPoints.Count)];

//Определяем позицию объекта

chick.Pos = rand.pos;

//Присваиваем текстуру, соответствующую направлению движения

chick.imgIndex = rand.index;

}

//Проверка, находится ли точка в пределах игрового поля

public bool checkBounds(Point input)

{

return (input.X < 0 || input.Y < 0 || input.X >= bounds.Width || input.Y >= bounds.Height);

}

//Проверка, находится ли точка на камне

public bool checkRocks(Point input)

{

for (int i = 0; i < rocks.Count; i++)

{

if (rocks[i].Pos == input)

{

return true;

}

}

return false;

}

//Проверка, содержит ли змейка точку

private bool checkSnake(Point input)

{

for (int i = 0; i < snake.Count; i++)

{

//Если координата сегмента совпадает с точкой

if (snake[i].Pos == input)

{

return true;

}

}

return false;

}

//Кушаем еду

private void Eat()

{

//Добавляем новую точку, с координатами, равными последней точке в списке

//В следующем шаге тело подрастет

snake.Add(snake[snake.Count - 1].Copy());

//Добавляем очки

gameScore += 100;

}

private GameObject GenerateFood()

{

//Создаем точку в случайной позиции в пределах игрового поля

Point randpoint = new Point(random.Next(0, bounds.Width), random.Next(0, bounds.Height));

//Проверяем, содержит ли тело змейки нашу случайную точку, в таком случае еда появится на теле змейке

//Пробегаемся по списку сегментов

//Если содержит случайная коодината выпадает на змейку или камень, рекурсивно гененируем новую

if (checkSnake(randpoint) | checkRocks(randpoint))

return GenerateFood();

else

return new GameObject(randpoint.X, randpoint.Y, 0);

}

//Создаем эффект сплющенной головы

private void FlatHead()

{

int halfPix = PixelLen / 2;

//В зависимости от направления сдвигаем изображение на предыдущий сегмент

switch (snake[0].Dir)

{

case Direction.Down:

snake[0].imgIndex = 16;

snake[0].imgPos = new Point(snake[0].imgPos.X, snake[0].imgPos.Y - halfPix);

break;

case Direction.Right:

snake[0].imgIndex = 17;

snake[0].imgPos = new Point(snake[0].imgPos.X - halfPix, snake[0].imgPos.Y);

break;

case Direction.Left:

snake[0].imgIndex = 18;

snake[0].imgPos = new Point(snake[0].imgPos.X + PixelLen, snake[0].imgPos.Y);

break;

case Direction.Up:

snake[0].imgIndex = 19;

snake[0].imgPos = new Point(snake[0].imgPos.X, snake[0].imgPos.Y + PixelLen);

break;

}

}

//Умерли

private void Die()

{

//Плющим текстуру головы

FlatHead();

//Проигрываем звук удара

PlaySound(hitSound);

//Объявляем смерть

gameOver = true;

//Останавливаем игровой таймер

gameTimer.Stop();

if (gameScore != 0)

{

m\_dbConn = new SQLiteConnection();

m\_sqlCmd = new SQLiteCommand();

dbFileName = "snake.db";

if (!File.Exists(dbFileName))

{

SQLiteConnection.CreateFile(dbFileName);

}

try

{

m\_dbConn = new SQLiteConnection("Data Source=" + dbFileName);

m\_dbConn.Open();

m\_sqlCmd.Connection = m\_dbConn;

m\_sqlCmd.CommandText = "CREATE TABLE IF NOT EXISTS Record (id INTEGER PRIMARY KEY AUTOINCREMENT, author TEXT, score INTEGER)";

m\_sqlCmd.ExecuteNonQuery();

}

catch (SQLiteException ex)

{

MessageBox.Show("Error: " + ex.Message);

}

try

{

m\_sqlCmd.CommandText = "INSERT INTO Record ('author', 'score') values ('" + nickname + "' , '" + gameScore + "')";

m\_sqlCmd.ExecuteNonQuery();

MessageBox.Show("Рекорд записан!");

}

catch (SQLiteException ex)

{

MessageBox.Show("Error: " + ex.Message);

}

}

}

//Если расстояние до головы меньше 10 пикселей,то яюлоко стоит на месте

if (minItem.Distance < 10)

{

//В 97 случаях из 100, двигаемся! Даем змее небольшое преимущество в скорости

if (random.Next(100) > 3)

{

//Если шагов в списке больше чем 1

if (posPoints.Count != 1)

{

//Удаляем нежелательный шаг, что сократит расстояние до головы

posPoints.Remove(minItem);

}

//Переставляем курицу

ChickStep(posPoints, chick);

return;

}

}

else

{

//Иначе, курица ведет себя спойно, клюет корм

//Изредка двигаемся

if (random.Next(100) < 10)

{

//Переставляем курицу

ChickStep(posPoints, chick);

return;

}

}

}

//Если до этого момента дошло, значит курица стоит на месте, просто поворачиваем ее текстуру

//Будто оглядывается по сторонам

if (random.Next(100) < 20)

{

chick.imgIndex = random.Next(4);

}

}

//Метод для перестановки курицы слуайную возможную позицию

private void ChickStep(List<ChickInfo> posPoints, GameObject chick)

{

var rand = posPoints[random.Next(posPoints.Count)];

//Определяем позицию объекта

chick.Pos = rand.pos;

//Присваиваем текстуру, соответствующую направлению движения

chick.imgIndex = rand.index;

}

//Проверка, находится ли точка в пределах игрового поля

public bool checkBounds(Point input)

{

return (input.X < 0 || input.Y < 0 || input.X >= bounds.Width || input.Y >= bounds.Height);

}

//Проверка, находится ли точка на камне

public bool checkRocks(Point input)

{

for (int i = 0; i < rocks.Count; i++)

{

if (rocks[i].Pos == input)

{

return true;

}

}

return false;

}

//Проверка, содержит ли змейка точку

private bool checkSnake(Point input)

{

for (int i = 0; i < snake.Count; i++)

{

//Если координата сегмента совпадает с точкой

if (snake[i].Pos == input)

{

return true;

}

}

return false;

}

//Кушаем еду

private void Eat()

{

//Добавляем новую точку, с координатами, равными последней точке в списке

//В следующем шаге тело подрастет

snake.Add(snake[snake.Count - 1].Copy());

//Добавляем очки

gameScore += 100;

}

private GameObject GenerateFood()

{

//Создаем точку в случайной позиции в пределах игрового поля

Point randpoint = new Point(random.Next(0, bounds.Width), random.Next(0, bounds.Height));

//Проверяем, содержит ли тело змейки нашу случайную точку, в таком случае еда появится на теле змейке

//Пробегаемся по списку сегментов

//Если содержит случайная коодината выпадает на змейку или камень, рекурсивно гененируем новую

if (checkSnake(randpoint) | checkRocks(randpoint))

return GenerateFood();

else

return new GameObject(randpoint.X, randpoint.Y, 0);

}

//Создаем эффект сплющенной головы

private void FlatHead()

{

int halfPix = PixelLen / 2;

//В зависимости от направления сдвигаем изображение на предыдущий сегмент

switch (snake[0].Dir)

{

case Direction.Down:

snake[0].imgIndex = 16;

snake[0].imgPos = new Point(snake[0].imgPos.X, snake[0].imgPos.Y - halfPix);

break;

case Direction.Right:

snake[0].imgIndex = 17;

snake[0].imgPos = new Point(snake[0].imgPos.X - halfPix, snake[0].imgPos.Y);

break;

case Direction.Left:

snake[0].imgIndex = 18;

snake[0].imgPos = new Point(snake[0].imgPos.X + PixelLen, snake[0].imgPos.Y);

break;

case Direction.Up:

snake[0].imgIndex = 19;

snake[0].imgPos = new Point(snake[0].imgPos.X, snake[0].imgPos.Y + PixelLen);

break;

}

}

//Умерли

private void Die()

{

//Плющим текстуру головы

FlatHead();

//Проигрываем звук удара

PlaySound(hitSound);

//Объявляем смерть

gameOver = true;

//Останавливаем игровой таймер

gameTimer.Stop();

if (gameScore != 0)

{

m\_dbConn = new SQLiteConnection();

m\_sqlCmd = new SQLiteCommand();

dbFileName = "snake.db";

if (!File.Exists(dbFileName))

{

SQLiteConnection.CreateFile(dbFileName);

}

try

{

m\_dbConn = new SQLiteConnection("Data Source=" + dbFileName);

m\_dbConn.Open();

m\_sqlCmd.Connection = m\_dbConn;

m\_sqlCmd.CommandText = "CREATE TABLE IF NOT EXISTS Record (id INTEGER PRIMARY KEY AUTOINCREMENT, author TEXT, score INTEGER)";

m\_sqlCmd.ExecuteNonQuery();

}

catch (SQLiteException ex)

{

MessageBox.Show("Error: " + ex.Message);

}

try

{

m\_sqlCmd.CommandText = "INSERT INTO Record ('author', 'score') values ('" + nickname + "' , '" + gameScore + "')";

m\_sqlCmd.ExecuteNonQuery();

MessageBox.Show("Рекорд записан!");

}

catch (SQLiteException ex)

{

MessageBox.Show("Error: " + ex.Message);

}

}

}

**«События формы, отрисовка, ресайз, нажатие клавиш»**

//Отрисовка игры на холст

private void canvas\_Paint(object sender, PaintEventArgs e)

{

//Если игра не показывается и фон не существует, выходим из метода, ничего не рисуем

if (!GameShowed | background == null) return;

Graphics g = e.Graphics;

//Сдвигаем начало координат для смещения области отрисовки в центр формы

g.TranslateTransform(transform.X, transform.Y);

//Рисуем фон (трава с камнями)

g.DrawImage(background, 0, 0);

if (isEditor)

{

//Если показывается редактор

//Пробегаемся по списку камней и рисуем каждый в своей позиции

for (int i = 0; i < rocks.Count; i++)

{

g.DrawImage(rocksTexList[0], rocks[i].imgPos);

}

}

else

{

//Если показывается игра

//Рисуем текстуры яблок

for (int i = 0; i < apples.Count; i++)

{

g.DrawImage(appleImg, apples[i].imgPos);

}

//Рисуем текстуры куриц

for (int i = 0; i < chickens.Count; i++)

{

//Изображение хватаем из списка

g.DrawImage(chickenTexList[chickens[i].imgIndex], chickens[i].imgPos);

}

}

//Рисуем текстуры сегментов змейки

for (int i = snake.Count - 1; i >= 0; i--)

{

//Изображение хватаем из списка, по номеру который хранится в поле класса imgIndex

g.DrawImage(snakeTexList[snake[i].imgIndex], snake[i].imgPos);

}

//Если показывается игра

if (!isEditor)

{

//Отрисовка текста в самую последнюю очередь, чтобы он был поверх всей картинки

//Включаем сглаживание шрифтов для рисования текста

g.TextRenderingHint = TextRenderingHint.AntiAlias;

//Рисуем очки

g.DrawString("Очки: " + gameScore.ToString(), drawFont, Brushes.White, screenCenter.X, PixelLen, sf);

if (gameOver)

{

//Если змейка мертва, надо оповестить об этом игрока

//Рисуем надпись

g.DrawString(overText + Environment.NewLine + infoText, drawFont, Brushes.White, screenCenter.X, screenCenter.Y, sf);

}

}

}

//Событие по изменению размера формы

private void canvas\_Resize(object sender, EventArgs e)

{

//Если показывается игры

if (GameShowed)

{

//Перерасчет размера игрового поля, размеров текстур

InitGameField();

//Перерасчет позиции текстур камней

for (int i = 0; i < rocks.Count; i++)

{

rocks[i].CalculateImagePos();

}

//Перерасчет позиции текстур сегментов змеи

for (int i = 0; i < snake.Count; i++)

{

snake[i].CalculateImagePos();

}

//Если показывается игра

if (!isEditor)

{

//Если змея умерла, перерасчет позиции текстуры сплющенной головы

if (gameOver)

{

FlatHead();

}

//Перерасчет координат текстур яблок и куриц

for (int i = 0; i < chickens.Count; i++)

{

chickens[i].CalculateImagePos();

}

for (int i = 0; i < apples.Count; i++)

{

apples[i].CalculateImagePos();

}

//Перерисовка камней на заднем фоне

DrawLevel();

}

else

{

//Если показывается редактор, перерисовываем сетку на фоне

DrawEditorBack();

}

//Принудительная перерисовка холста

canvas.Invalidate();

}

}

//Собираем список координат камней

//На выходе список точек

private List<Point> RocksToPoints()

{

//linq

return rocks.Select(rock => rock.Pos).ToList();

}

//Событие, вызывающееся по нажатию клавиш

private void canvas\_KeyDown(object sender, KeyEventArgs e)

{

//Определяем управление игрой

switch (e.KeyCode)

{

//По стрелочкам или WSAD меняем текущее направление змейки

case Keys.A:

case Keys.Left:

if (!isEditor)

{

//Во избежание смерти, от поворота "в себя"

if (snake[0].Dir != Direction.Right)

direction = Direction.Left;

}

break;

case Keys.D:

case Keys.Right:

if (!isEditor)

{

if (snake[0].Dir != Direction.Left)

direction = Direction.Right;

}

break;

case Keys.W:

case Keys.Up:

if (!isEditor)

{

if (snake[0].Dir != Direction.Down)

direction = Direction.Up;

}

break;

case Keys.S:

case Keys.Down:

if (!isEditor)

{

if (snake[0].Dir != Direction.Up)

direction = Direction.Down;

}

else

//Если комбинация Ctrl+S и показывается редактор

if (e.Control && isEditor)

{

saveFileDialog.FileName = string.Empty;

//Вызываем диалог сохранения файла уровня

if (saveFileDialog.ShowDialog() == DialogResult.OK)

{

//Собираем информацию об уровне в экземпляр класса GameLevel

GameLevel gl = new GameLevel(snake[0].Pos, snake[0].Dir, RocksToPoints());

//Сериализуем и сохраняем в файл

Utils.SaveBinary(saveFileDialog.FileName, gl);

}

}

break;

case Keys.O:

//Если комбинация Ctrl+O и показывается редактор

if (e.Control && isEditor)

{

openFileDialog.FileName = string.Empty;

//Вызываем диалог открытия файла

if (openFileDialog.ShowDialog() == DialogResult.OK)

{

//Читаем левел

ReadLevel(openFileDialog.FileName);

//Принудительно перерисовываем холст

canvas.Invalidate();

}

}

break;

case Keys.X:

//Если комбинация Ctrl+X и показывается редактор

if (e.Control && isEditor)

{

//Чистим список камней

rocks.Clear();

//Принудительно перерисовываем холст

canvas.Invalidate();

}

break;

case Keys.R:

//На R и если показывается игра, начинаем игру заново

if (!isEditor)

{

//Останавливаем игровой таймер

gameTimer.Stop();

//Перезапускаем игру на том же уровне

StartGame(currentLev, false);

}

break;

case Keys.Space:

//Если показывается игра, по пробелу ускоряем игру до скорости 20 (= 1000/50)

if (!isEditor)

{

gameTimer.Interval = 50;

}

break;

case Keys.E:

//Если нажата клавиша E и показывается игра

if (!isEditor)

{

//Если змейка жива, запускаем редактор

if (!gameOver)

RunEditor(currentLev, false);

}

else

{

//Если показывается редактор, запускаем игру, пропускаем загрузку уровня из файла

StartGame(currentLev, true);

}

break;

case Keys.P:

if (!gameOver)

{

//Пауза

gameTimer.Enabled = !gameTimer.Enabled;

}

break;

case Keys.Escape:

//На Esc возвращемся в главное меню

ReturnToMenu();

break;

}

}

//Возвращение в главное меню

private void ReturnToMenu()

{

if (PressEsc != null)

{

//Если показывается игра

if (!isEditor)

{

//Если змейка жива

if (!gameOver)

{

//Объявляем смерть и останавливаем игровой таймер

gameOver = true;

gameTimer.Stop();

}

}

//Обновляем список файлов уровней

OpenLevelFiles();

//Сообщаем, что игра больше не показывается

GameShowed = false;

//Срабатывает событие, сообщающее главной форме, что Esc был нажат

PressEsc(this, e);

}

}

//Событие отпускания клавиши

private void canvas\_KeyUp(object sender, KeyEventArgs e)

{

//Если пробел отпущен, возвращаем старую скорость игры

if (e.KeyCode == Keys.Space)

gameTimer.Interval = 1000 / gameSpeed;

}

**«Редактор, события нажатий кнопки мыши и передвижение»**

//Рисуем фон для редактора

private void DrawEditorBack()

{

//Меняем размер фона в соответствии с размером клиентской области формы

background = Utils.ResizeImage(grassImg, realSize.Width, realSize.Height);

//Рисуем вспомогательную сетку

using (var g = Graphics.FromImage(background))

{

//Полупрозрачная красная ручка

Pen p2 = new Pen(Color.FromArgb(100, Color.Red));

//Пробегаемся по ширине и высоте, рисуем линии через пиксель

for (int i = 1; i < bounds.Width; i++)

{

g.DrawLine(p2, i \* pixelLen, 0, i \* pixelLen, realSize.Height);

}

for (int i = 1; i < bounds.Height; i++)

{

g.DrawLine(p2, 0, i \* pixelLen, realSize.Width, i \* pixelLen);

}

}

}

//Запуск редактора

public void RunEditor(int levnum, bool isNew)

{

//Останавливаем игровой таймер

gameTimer.Stop();

GameShowed = true;

//Указываем на запуск редактора

isEditor = true;

//Инициализация игрового поля

//Расчет размеров текстур, положения поля на форме

InitGameField();

//Если редактор запускается из главного меню, а не из игры

//Нам надо загрузить уровень

if (isNew)

{

snake.Clear();

LoadLevel(levnum);

}

//Рисуем фон с сеткой

DrawEditorBack();

//Перерисовываем холст

canvas.Invalidate();

}

//Производим обработку нажатий мыши

private void EditRocks(MouseEventArgs e)

{

//Получаем координаты курсора в виртуальном игровом поле с учетом сдвига игрового поля

var virtPt = new Point((e.X - transform.X) / pixelLen, (e.Y - transform.Y) / pixelLen);

//Если координата в границах игрового поля

if (!checkBounds(virtPt))

{

//Если ЛКМ

if (e.Button == MouseButtons.Left)

{

//Проверяем находится ли точка на существующем камне или на змейке

if (!checkRocks(virtPt) && !checkSnake(virtPt))

{

//Если нет, добавляем новый камень в позицию виртуального курсора

rocks.Add(new GameObject(virtPt.X, virtPt.Y, 0));

}

}

else

//Если ПКМ

if (e.Button == MouseButtons.Right)

{

//Проверяем находится ли точка на существующем камне или на змейке

if (checkRocks(virtPt) && !checkSnake(virtPt))

{

//Удаляем камень в позиции курсора

//linq

rocks.RemoveAll((x) => x.Pos == virtPt);

}

}

else

//Если средняя кнопка (колесо)

if (e.Button == MouseButtons.Middle)

{

//Если курсор не на камне

if (!checkRocks(virtPt))

{

//Вращаем змейку по часовой стрелке

Direction clockRot = Direction.Down;

//Если змейка уже существует, будем отталкиваться от ее текущего направления

if (snake.Count != 0)

clockRot = snake[0].Dir;

//Меняем направление с учетом текущей позиции змейки

if (checkSnake(virtPt))

{

switch (clockRot)

{

//Если змейка смотри вверх, поворачиваем вправо, по аналогии с остальными направлениями

case Direction.Up:

clockRot = Direction.Right;

break;

case Direction.Down:

clockRot = Direction.Left;

break;

case Direction.Left:

clockRot = Direction.Up;

break;

case Direction.Right:

clockRot = Direction.Down;

break;

default:

break;

}

}

//Создаем норвую змейку в виртуальной позиции курсора, в новом направлении

SpawnSnake(virtPt.X, virtPt.Y, clockRot);

}

}

}

//Перерисовываем холст

canvas.Invalidate();

}

//Событие, вызывающееся по нажатию клавиш мыши

private void canvas\_MouseDown(object sender, MouseEventArgs e)

{

///Если показывается редактор

if (isEditor)

{

//Обрабатываем игровое поле

EditRocks(e);

}

}

//Событие, вызывающееся по перемещению указателя мыши

private void canvas\_MouseMove(object sender, MouseEventArgs e)

{

//Если показывается редактор

if (isEditor)

{

//Если удерживается ПКМ или ЛКМ

if (e.Button == MouseButtons.Right | e.Button == MouseButtons.Left)

{

//Обрабатываем игровое поле

EditRocks(e);

}

}

}