Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования  
«Новосибирский государственный технический университет»

Кафедра Автоматизированных систем управления



**Отчет**

**по лабораторной работе №4**

**Тема:**

**Дисциплина**: Программирование

Вариант **№9**

|  |  |
| --- | --- |
| Выполнили:  *Коваленко Е.А.*  *Арифуллин А.Д.*  Факультет *АВТФ*  Направление подготовки  *09.03.03. – Прикладная информатика*  Группа *АП-327* | Проверил:  *Cтарший преподаватель кафедры АСУ*  *Эстрайх И.В.* |

Новосибирск

2024

Оглавление

[Постановка задачи 3](#_Toc178323991)

[Тестовые данные и ожидаемые результаты 3](#_Toc178323992)

[Описание разработанных классов 3](#_Toc178323993)

[Тестирование программы 5](#_Toc178323994)

[Скриншоты результатов работы программы и выполнения тестов 8](#_Toc178323995)

[Выводы 10](#_Toc178323996)

[Приложение. Исходные коды всех разработанных файлов с комментариями. 10](#_Toc178323997)

# Постановка задачи

Для программы, созданной в лабораторной работе №3, разработать графический интерфейс пользователя. Для разработки было выбрано WPF (Windows Presentation Foundation).

# Тестовые данные и ожидаемые результаты

|  |  |
| --- | --- |
| Нажатие на кнопку «Начать игру» | Запустится новая игра |
| Нажатие на кнопку «Загрузить игру» | Запустится сохраненная игра |

# Описание разработанных классов

**Класс Field**

|  |  |  |  |
| --- | --- | --- | --- |
| Поле/Свойство/Метод | Тип/Возвращаемый тип | Параметры | Описание |
| b\_hor | int | - | Горизонтальное направление движения мяча. |
| b\_ver | int | - | Вертикальное направление движения мяча. |
| speed | double | - | Скорость игры |
| pl\_pos | int | - | Позиция платформы на поле |
| pl\_size | int | - | Размер платформы |
| field | string[,] | - | Основное игровое поле. |
| buff\_field | string[,] | - | Поле, аналогичное игровому по размерам и содержащее бонусы |
| Field() | Field | int b\_h, int b\_v, int pl\_p, int pl\_s, double spd | Инициализирует игровое поле с указанными или стандартными параметрами. |
| FieldPrint() | void | - | Отображает текущее состояние игрового поля в консоли. |
| PlatformMove() | void | int dir | Перемещает платформу в указанном направлении, если это допустимо. |
| BallMove() | bool | - | Перемещает мяч в соответствии с направлением, возвращает false, если мяч достиг нижней границы. |
| CrushBlock() | void | int n, int m | Уничтожает блок на указанной позиции, возможно добавляя бонус в buff\_field. |
| WinCheck() | bool | - | Проверяет, есть ли на поле ещё разрушимые блоки. Возвращает true, если все разрушены. |
| BuffFalling() | void | - | Обрабатывает падение бонусов на одну позицию вниз. Применяет эффект бонусов, достигших платформы. |
| BallCollision() | bool | int n, int m | Проверяет наличие столкновения мяча с объектами. Обрабатывает смену направления мяча и уничтожение блоков, если требуется. |
| N | const int | - | Высота игрового поля. |
| M | const int | - | Ширина игрового поля. |
| Chance | const int | - | Шанс выпадения бонусов |

**Класс MainWindow**

|  |  |  |  |
| --- | --- | --- | --- |
| Поле/Свойство/Метод | Тип/Возвращаемый тип | Параметры | Описание |
| gameField | Field | - | Игровое поле |
| gameTimer | DispatcherTimer | - | |  | | --- | |  |  |  | | --- | | Таймер для игрового цикла. | |
| platform | Rectangle | - | Прямоугольник, представляющий платформу. |
| ball | Ellipse | - | Круг, представляющий шарик. |
| bricks | Rectangle[,] | - | Двумерный массив прямоугольников, представляющих кирпичи. |
| bonuses | TextBlock[,] | - | Двумерный массив текстовых блоков, представляющих бонусы. |
| WindowWidth | const double | - | Ширина окна. |
| WindowHeight | const double | - | Высота окна. |
| offsetX | double | - | Горизонтальное смещение игрового поля. |
| offsetY | double | - | Вертикальное смещение игрового поля. |
| isPaused | bool | - | Находится ли игра на паузе |
| InitializeUI() | void | - | Создание элементов пользовательского интерфейса. |
| StartGame\_Click() | void | object sender, RoutedEventArgs e | Обработчик кнопки "Начать игру". |
| LoadGame\_Click() | void | object sender, RoutedEventArgs e | Обработчик кнопки "Загрузить игру". |
| SaveGame() | void | - | Сохранение текущего состояния игры. |
| InitializeGame() | void | - | Инициализация игрового процесса. |
| DrawGame() | void | - | Отображение состояния игрового поля и объектов. |
| GameLoop() | void | object sender, EventArgs e | Главный игровой цикл. |
| UpdateGame() | void | - | Логика обновления состояния игры. |
| KeyControls() | void | object sender, KeyEventArgs e | Управление платформой стрелочками |
| PauseControl() | void | object sender, MouseButtonEventArgs e | Управление паузой через нажатие мыши. |
| DrawBorders() | void | - | Отрисовка границ игрового поля. |
| DrawBorderCell() | void | int row, int col | Отрисовка отдельной ячейки границы. |

# Описание приложения

Изначально пользователя встречают инструкции управления и предложение загрузить игру или начать новую игру. После начала игры рисуется игровое поле. Пользователю необходимо, двигая платформой внизу игрового поля с помощью стрелочек влево-вправо не допускать падения мяча за нижнюю границу поля и разбить с помощью этого самого мяча все кирпичи. При нажатии на левую кнопку мыши игрок может поставить игру на паузу. При нажатии на ESC происходит сохранение игры и выход из неё. Если игрок сумел разбить все кирпичи мячом, то программа уведомляет его о победе. Если же мяч упал за нижнюю границу поля игра говорит игроку о поражении.

# Тестирование программы

Результаты тестирования программы оказались удовлетворительными. Полученные результаты совпали с теоретическими представлениями. Модульные тесты логики программы (Field) уже были выполнены в третьей лабораторной, графический интерфейс работает правильно.

# Скриншоты результатов работы программы

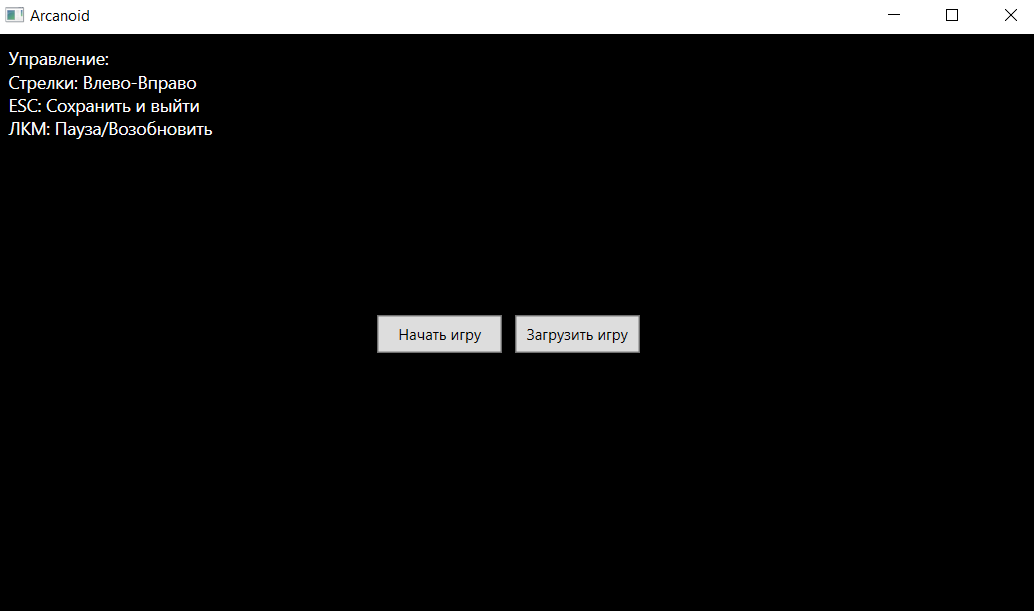


Рисунок 1. Стартовый экран

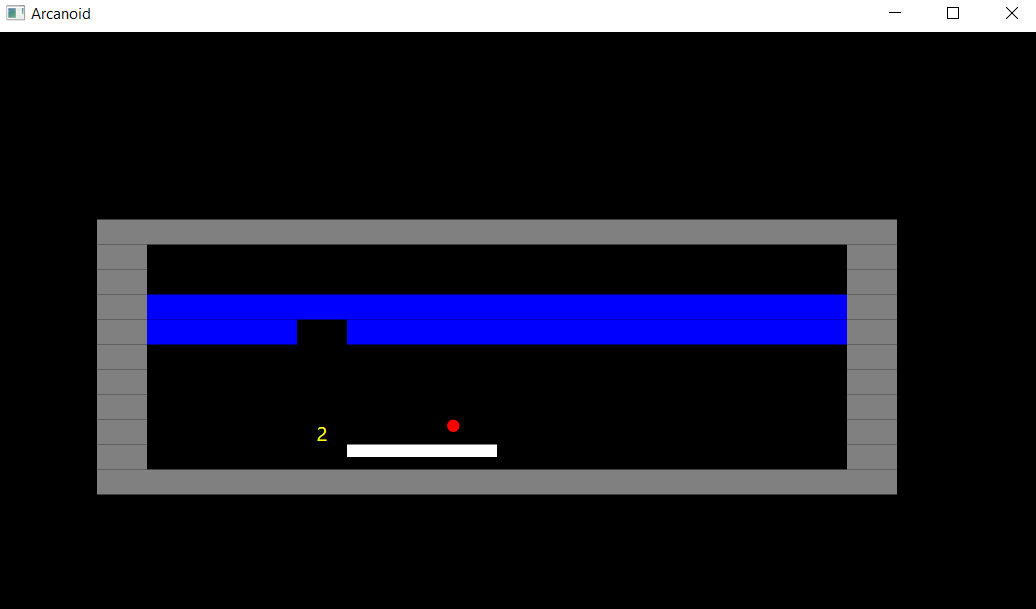


Рисунок 2 Сама игра

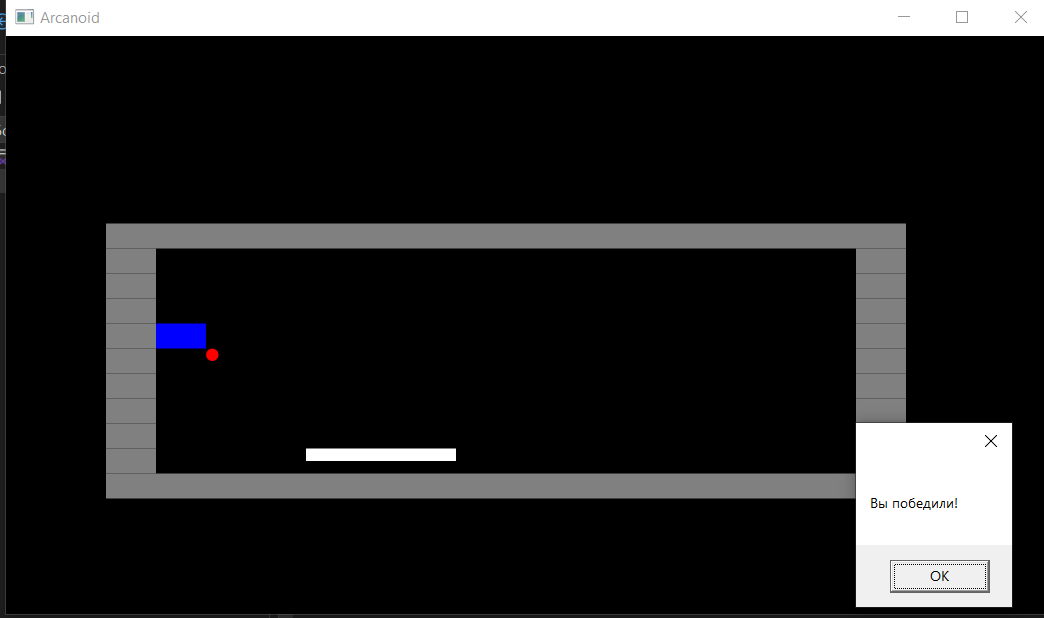


Рисунок Уведомление о конце игры

# Выводы

При выполнении данной лабораторной работы мы разработали графический интерфейс для ранее созданной игры в жанре Арканоид. Мы освоили основы работы с Windows Presentation Foundation и сделали игру более дружелюбной для пользователя, чем когда она была оформлена в виде консольного приложения.

# Приложение. Исходные коды всех разработанных файлов с комментариями.

**Field**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Channels;

using System.Threading.Tasks;

namespace WpfApp1

{

[Serializable]

public class Field

{

public const int N = 10;

public const int M = 16;

public const int Chance = 2;

public int b\_hor { get; set; } // horizontal ball direction

public int b\_ver { get; set; } // vertical ball direction

public double speed { get; set; } // higher the number - lower the speed to obtain a bonus

public int pl\_pos { get; set; } // platform position

public int pl\_size { get; set; } // platform size

public string[,] field { get; set; }

public string[,] buff\_field { get; set; }

public Field(int b\_h = 1, int b\_v = -1, int pl\_p = 0, int pl\_s = 3, double spd = 400)

{

b\_hor = b\_h;

b\_ver = b\_v;

pl\_pos = pl\_p;

pl\_size = pl\_s;

speed = spd;

field = new string[N, M];

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

{

field[0, j] = "w";

}

for (int i = 1; i < N - 1; i++)

{

field[i, 0] = "w";

for (int j = 1; j < M - 1; j++)

{

field[i, j] = "-";

}

field[i, M - 1] = "w";

}

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

{

field[N - 1, j] = "-";

}

for (int j = 1; j < M - 1; j++)

{

field[N / 2 - 2, j] = "c";

}

for (int j = 1; j < M - 1; j++)

{

field[N / 2 - 1, j] = "c";

}

field[N - 2, 1] = "b";

buff\_field = new string[N, M];

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

{

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

{

buff\_field[i, j] = "-";

}

}

}

public void FieldPrint()

{

string tmp = "";

Console.Write("╔");

for (int j = 1; j < M - 1; j++)

{

Console.Write("═");

}

Console.WriteLine("╗");

for (int i = 1; i < N - 1; i++)

{

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

{

switch (field[i, j])

{

case "w":

tmp = "║";

break;

case "b":

tmp = "o";

break;

case "c":

tmp = "H";

break;

default:

tmp = "-";

break;

}

if (buff\_field[i, j] != "-")

{

tmp = buff\_field[i, j];

}

Console.Write(tmp);

}

Console.WriteLine();

}

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

{

if ((j >= pl\_pos) && (j <= pl\_pos + pl\_size - 1))

{

Console.Write("p");

}

else

{

Console.Write("-");

}

}

Console.WriteLine();

}

public void PlatformMove(int dir)

{

if ((pl\_pos + dir >= 0) && (pl\_size + pl\_pos + dir <= M - 2))

{

pl\_pos = pl\_pos + dir;

}

}

public bool BallMove()

{

int n = 0;

int m = 0;

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

{

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

{

if (field[i, j] == "b")

{

n = i;

m = j;

}

}

}

if (n == N - 1)

{

return false;

}

if (!BallCollision(n, m))

{

field[n, m] = "-";

field[n + b\_ver, m + b\_hor] = "b";

}

return true;

}

public void CrushBlock(int n, int m)

{

if (field[n, m] == "c")

{

field[n, m] = "-";

Random rnd = new Random();

int dice;

if (rnd.Next(Chance) == 0)

{

dice = rnd.Next(1, 5); // 1-platform\_size\_up 2-platform\_size\_down 3-game\_speed\_up 4-game\_speed\_down

buff\_field[n, m] = dice.ToString();

}

}

}

public bool WinCheck()

{

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

{

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

{

if (field[i, j] == "c")

{

return false;

}

}

}

return true;

}

public void BuffFalling()

{

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

{

switch (buff\_field[N - 1, j])

{

case "1":

pl\_size = pl\_size + 1;

break;

case "2":

if (pl\_size - 1 >= 2)

{

pl\_size = pl\_size - 1;

}

break;

case "3":

speed = speed / 1.4;

if (speed < 300)

{

speed = 300;

}

break;

case "4":

speed = speed \* 1.4;

if (speed > 600)

{

speed = 600;

}

break;

default:

break;

}

}

for (int i = N - 1; i > 0; i--)

{

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

{

buff\_field[i, j] = buff\_field[i - 1, j];

}

}

}

public bool BallCollision(int n, int m)

{

bool diagonal\_check = true;

if (field[n + b\_ver, m] != "-")

{

diagonal\_check = false;

CrushBlock(n + b\_ver, m);

b\_ver = b\_ver \* -1;

return true;

}

else

{

if ((n + b\_ver == N - 1) && (pl\_pos <= m) && (pl\_pos + pl\_size >= m))

{

diagonal\_check = false;

b\_ver = b\_ver \* -1;

return true;

}

}

if (field[n, m + b\_hor] != "-")

{

diagonal\_check = false;

CrushBlock(n, m + b\_hor);

b\_hor = b\_hor \* -1;

return true;

}

if (diagonal\_check && field[n + b\_ver, m + b\_hor] != "-")

{

CrushBlock(n + b\_ver, m + b\_hor);

b\_ver = b\_ver \* -1;

b\_hor = b\_hor \* -1;

return true;

}

else

{

if (diagonal\_check && (n + b\_ver == N - 1) && (pl\_pos <= m + b\_hor) && (pl\_pos + pl\_size >= m + b\_hor))

{

b\_ver = b\_ver \* -1;

b\_hor = b\_hor \* -1;

return true;

}

}

return false;

}

}

}

**MainWindow.xaml**

<Window x:Class="Arcanoid.MainWindow"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

Title="Arcanoid" Height="500" Width="850">

<Grid>

<Canvas Name="GameCanvas" Background="Black" />

<StackPanel Orientation="Horizontal" HorizontalAlignment="Center" VerticalAlignment="Top" Margin="10">

</StackPanel>

</Grid>

</Window>

**MainWindow.xaml.cs**

using System;

using System.IO;

using System.Runtime.Serialization.Formatters.Binary;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Input;

using System.Windows.Media;

using System.Windows.Shapes;

using System.Windows.Threading;

using WpfApp1;

namespace Arcanoid

{

[Serializable]

public partial class MainWindow : Window

{

private Field gameField;

private DispatcherTimer gameTimer;

private Rectangle platform;

private Ellipse ball;

private Rectangle[,] bricks;

private TextBlock[,] bonuses;

private const double WindowWidth = 850;

private const double WindowHeight = 500;

private double offsetX;

private double offsetY;

private BinaryFormatter formatter = new BinaryFormatter();

private bool isPaused = false;

public MainWindow()

{

InitializeComponent();

InitializeUI();

KeyDown += KeyControls;

MouseDown += PauseControl;

}

private void InitializeUI()

{

// Управление

var controls = new TextBlock

{

Text = "Управление:\nСтрелки: Влево-Вправо\nESC: Сохранить и выйти\nЛКМ: Пауза/Возобновить",

FontSize = 14,

Foreground = Brushes.White,

Margin = new Thickness(10),

HorizontalAlignment = HorizontalAlignment.Left,

VerticalAlignment = VerticalAlignment.Top

};

// Кнопки управления

var startButton = new Button

{

Content = "Начать игру",

Width = 100,

Height = 30,

Margin = new Thickness(5)

};

startButton.Click += StartGame\_Click;

var loadButton = new Button

{

Content = "Загрузить игру",

Width = 100,

Height = 30,

Margin = new Thickness(5)

};

loadButton.Click += LoadGame\_Click;

// Панель для кнопок

var buttonPanel = new StackPanel

{

Orientation = Orientation.Horizontal,

HorizontalAlignment = HorizontalAlignment.Center

};

buttonPanel.Children.Add(startButton);

buttonPanel.Children.Add(loadButton);

// Установка позиций для инструкций и кнопок

Canvas.SetLeft(controls, 0);

Canvas.SetTop(controls, 0);

Canvas.SetLeft(buttonPanel, 300);

Canvas.SetTop(buttonPanel, 220);

// Добавление элементов на холст

GameCanvas.Children.Add(controls);

GameCanvas.Children.Add(buttonPanel);

}

private void StartGame\_Click(object sender, RoutedEventArgs e)

{

gameField = new Field();

InitializeGame();

}

private void LoadGame\_Click(object sender, RoutedEventArgs e)

{

try

{

using (var fs = new FileStream("gmamelevel.dat", FileMode.OpenOrCreate))

{

gameField = (Field)formatter.Deserialize(fs);

}

InitializeGame();

}

catch (Exception ex)

{

MessageBox.Show($"Ошибка загрузки игры: {ex.Message}");

}

}

private void SaveGame()

{

try

{

using (var fs = new FileStream("gmamelevel.dat", FileMode.OpenOrCreate))

{

MessageBox.Show("Игра сохранена!", "Сохранение игры", MessageBoxButton.OK);

formatter.Serialize(fs, gameField);

}

}

catch (Exception ex)

{

MessageBox.Show($"Ошибка сохранения игры: {ex.Message}");

}

}

private void InitializeGame()

{

gameTimer?.Stop();

GameCanvas.Children.Clear();

bricks = new Rectangle[Field.N, Field.M];

bonuses = new TextBlock[Field.N, Field.M];

double fieldWidth = Field.M \* 40;

double fieldHeight = Field.N \* 20;

// Отступы для центрирования игрового поля в окне

offsetX = (WindowWidth - fieldWidth - 50) / 2;

offsetY = (WindowHeight - fieldHeight) / 2;

// Создание объектов игры

platform = new Rectangle

{

Height = 10,

Fill = Brushes.White

};

GameCanvas.Children.Add(platform);

ball = new Ellipse

{

Width = 10,

Height = 10,

Fill = Brushes.Red

};

GameCanvas.Children.Add(ball);

for (int i = 0; i < Field.N; i++)

{

for (int j = 0; j < Field.M; j++)

{

if (gameField.field[i, j] == "c") // Создание поля игры проходит по созданной в третьей лабораторной системе

{

bricks[i, j] = new Rectangle

{

Width = 40,

Height = 20,

Fill = Brushes.Blue

};

Canvas.SetLeft(bricks[i, j], offsetX + j \* 40);

Canvas.SetTop(bricks[i, j], offsetY + i \* 20);

GameCanvas.Children.Add(bricks[i, j]);

}

bonuses[i, j] = new TextBlock

{

Width = 40,

Height = 20,

FontSize = 16,

Foreground = Brushes.Yellow,

TextAlignment = TextAlignment.Center,

VerticalAlignment = VerticalAlignment.Center

};

Canvas.SetLeft(bonuses[i, j], offsetX + j \* 40);

Canvas.SetTop(bonuses[i, j], offsetY + i \* 20);

GameCanvas.Children.Add(bonuses[i, j]);

}

}

DrawBorders();

gameTimer = new DispatcherTimer

{

Interval = TimeSpan.FromMilliseconds(gameField.speed)

};

gameTimer.Tick += GameLoop;

gameTimer.Start();

}

private void DrawGame()

{

// Настройка параметров платформы

platform.Width = gameField.pl\_size \* 40;

Canvas.SetLeft(platform, offsetX + gameField.pl\_pos \* 40 + 40);

Canvas.SetTop(platform, offsetY + (Field.N - 1) \* 20);

for (int i = 0; i < Field.N; i++) // Передвижение шарика реализуется с помощью логики, разработанной в третьей лабораторной

{

for (int j = 0; j < Field.M; j++)

{

if (gameField.field[i, j] == "b")

{

Canvas.SetLeft(ball, offsetX + j \* 40);

Canvas.SetTop(ball, offsetY + i \* 20);

break;

}

}

}

for (int i = 0; i < Field.N; i++)

{

for (int j = 0; j < Field.M; j++)

{

// При разрушении блока в ячейке блока генерируется случайный бонус (или его отсутствие) и записываетсяв buff\_field. Здесь этот бонус принимает своё значение в графическом интерфейсе

bonuses[i, j].Text = gameField.buff\_field[i, j] != "-" ? gameField.buff\_field[i, j] : "";

}

}

// Удаление разрушенных блоков

for (int i = 0; i < Field.N; i++)

{

for (int j = 0; j < Field.M; j++)

{

if (gameField.field[i, j] == "-")

{

if (bricks[i, j] != null)

{

GameCanvas.Children.Remove(bricks[i, j]);

bricks[i, j] = null;

}

}

}

}

}

private void GameLoop(object sender, EventArgs e)

{

if (!isPaused)

{

UpdateGame();

DrawGame();

}

}

private void UpdateGame()

{

// Проверка условий победы или поражения

if (!gameField.BallMove())

{

gameTimer.Stop();

MessageBox.Show("Вы проиграли!");

this.Close();

}

if (gameField.WinCheck())

{

gameTimer.Stop();

MessageBox.Show("Вы победили!");

this.Close();

}

// Обновление состояния игры в зависимости от бонуса

gameField.BuffFalling();

}

private void KeyControls(object sender, KeyEventArgs e) // Настройка движения

{

if (e.Key == Key.Left)

{

gameField.PlatformMove(-1);

}

else if (e.Key == Key.Right)

{

gameField.PlatformMove(1);

}

else if (e.Key == Key.Escape)

{

isPaused = true;

SaveGame();

Close();

}

}

private void PauseControl(object sender, MouseButtonEventArgs e) // Настройка паузы

{

if (e.LeftButton == MouseButtonState.Pressed)

{

isPaused = !isPaused;

}

}

private void DrawBorders()

{

for (int j = 0; j < Field.M; j++) // Верхние границы

{

DrawBorderCell(0, j);

}

for (int i = 1; i < Field.N; i++) // Боковые границы

{

DrawBorderCell(i, 0);

DrawBorderCell(i, Field.M - 1);

}

for (int j = 0; j < Field.M; j++) // Нижние границы

{

DrawBorderCell(Field.N, j);

}

}

private void DrawBorderCell(int row, int col)

{

var border = new Rectangle

{

Width = 40,

Height = 20,

Fill = Brushes.Gray

};

Canvas.SetLeft(border, offsetX + col \* 40);

Canvas.SetTop(border, offsetY + row \* 20);

GameCanvas.Children.Add(border);

}

}

}