УДК 004

Создание графического приложения с помощью языка программирования C# на примере видеоигры

Эрдман Александр Алексеевич
Приамурский государственный университет имени Шолом-Алейхема
Студент

Аннотация
В статье рассмотрен процесс создания графического приложения для ОС Windows на примере видеоигры «Змейка». Приложение разработана на базе платформы WPF языка программирования C#. Результатом исследования является работоспособное приложение и его полное описание работы.
Ключевые слова: C#, WPF, графическое приложение

Creating a graphical application using the C# programming language using the example of a video game

Erdman Alexander Alekseevich
Sholom-Aleichem Priamursky State University
Student

Abstract
The article describes the process of creating a graphical application for Windows using the example of the video game "Snake". The application is developed based on the WPF platform of the C# programming language. The result of the research is a workable application and its full description of work.
Keywords: C#, WPF, graphical application

1 Введение

1.1 Актуальность

В последние годы создание графических приложений и видеоигр стало одной из наиболее динамично развивающихся областей информационных технологий. Язык программирования C#, разработанный компанией Microsoft, предоставляет мощные инструменты и библиотеки для разработки высокопроизводительных и интерактивных графических приложений. В данном исследовании рассматривается процесс создания графического приложения на примере видеоигры с использованием языка программирования C#. Особое внимание уделено использованию такой технологии, как Windows Presentation Foundation (WPF), которая предоставляет разработчикам широкий спектр возможностей для создания сложных графических интерфейсов. Также рассмотрены вопросы оптимизации производительности и обеспечения качества кода, что является критически важным для успешного завершения проекта.

1.2 Обзор исследований

А.В. Писанко и Е.В. Жилина в своей работе рассмотрели способ реализации графических приложений на основе технологии WPF [1]. К.А. Войшнис исследовал эффективность использования приложений, созданных на платформах JS и WPF [2]. И.А. Рудишин в своей статье проанализировал платформу WPF для реализации интерфейсов пользовательских приложений [3]. А.А. Ковальчук, Е.П. Павленко и В.А. Айвазов изучили особенности технологии WPF, языка разметки приложений XAML и механизмы визуализации графического интерфейса [4]. С.Д. Васюта охарактеризовал достоинства технологии WPF и использование её для реализации игрового приложения Windows [5].

1.3 Цель исследования

Целью исследования является изучение возможностей и особенностей языка C# в контексте разработки графических приложений на практическом примере видеоигры.

2 Материалы и методы

Для создания приложения используется язык программирования C# и его платформа WPF. В качестве средства разработки выступает Visual Studio 2019.

3 Результаты и обсуждения

Приложение представляет из себя классическую игру «Змейка», смысл которой управлять постоянно растущей змеёй, собирать предметы и не сталкиваться с границами экрана или своими частями тела. В игре будет присутствовать один собираемый предмет в виде изображения яблока. Остальные компоненты игры реализуются только посредствам самой платформы WPF – змея, игровое поле, кнопки и текстовые поля.

Игра будет содержать в себе одну надпись, в которой будет отображено количество собранных очков; две кнопки – кнопка старта игры и кнопка перезапуска игры. Закрытие приложение будет осуществляться с помощью встроенной кнопки формы.

Проект приложения состоит из двух файлов. Первый файл имеет расширение .xaml, который является языком разметки, используемый для описания пользовательского интерфейса. Второй файл содержит главный класс приложения. В данном файле будет программироваться логика игры.

В первом файле вносятся изменения в разметку такие, как: добавление полотна, который будет являться игровым полем; инициализация двух кнопок, отвечающие за первый старт игры и её перезапуск; текстовое поле для вывода очков (рис. 1).

```
<Grid>
    <Canvas x:Name="GameCanvas" Background="Black" Focusable="True"/>
    <TextBlock x:Name="ScoreTextBlock" Text="Score: 0" Foreground="White" HorizontalAlignment="Left" Margin="10" VerticalAlignment="Top"/>
    <Button x:Name="RestartButton" Content="перезапустить" HorizontalAlignment="Center" VerticalAlignment="Center" Visibility="Collapsed" Click="RestartButton_Click"/>
    <Button x:Name="StartGame" Content="Начать игру" HorizontalAlignment="Center" VerticalAlignment="Center" Visibility="Visible" Click="StartGame_Click"/>
</Grid>
```

Рисунок 1. Объявление графических элементов в разметке приложения

После этого внутри проекта создаётся папка для хранения изображения подбираемого объекта. На данном этапе работа с разметкой завершена. Далее начинается работа с главным классом программы.

В начале создаются все необходимые переменные (рис. 2).

```
private const int SnakeSquareSize = 20;

private readonly SolidColorBrush _snakeColor = Brushes.Green;

14 references
private enum Direction
{
    Left, Right, Top, Bottom
}

private Direction _direction = Direction.Right;
private const int TimerInterval = 200;
private DispatcherTimer _timer;

private Rectangle _snakeHead;
private Point _foodPosition;

private static readonly Random RandomPositionFood = new Random();

private List<Rectangle> _snake = new List<Rectangle>();

private int _score = 0;
```

Рисунок 2. Объявление переменных

Первая переменная представляет из себя константу для хранения размера сегментов, из которых будет состоять змея. Следом за неё объявляется непосредственно её цвет. Затем создаётся список с возможными направления движения. Объект класса «Direction» хранит в себе направления движения по умолчанию при первом запуске игры. Вторая константа определяет скорость выполнения команд с помощью такого элемента, как таймер приложения. То есть данная константа отвечает за скорость воспроизведения игры в целом. Переменная типа «DispatcherTimer» инициализирует сам таймер. «_snakeHead» хранит отдельно «головую» змеи.

Это нужно для того, чтобы отслеживать начало персонажа, его направление движения, столкновения и наращивания новых сегментов. Для хранения подбираемого объекта выступает «_foodPosition». Также имеется переменная для очков, список, хранящий в себе все элементы персонажа и объект для генерации случайных значений для повеления на игровом поле предметов.

После объявления переменных следуют методы для загрузки компонентов игры (рис. 3).

```
public MainWindow()
{
    InitializeComponent();
}
0 references
private void GameCanvas_Loaded(object sender, RoutedEventArgs e)
{
    InitialGame();
}
3 references
private void InitialGame ()
{
    _snakeHead = CreateSnakeSegment(new Point(5, 5));
    _snake.Add(_snakeHead);
    GameCanvas.Children.Add(_snakeHead);

    PlaceFood();
    _timer = new DispatcherTimer();
    _timer.Tick += Timer_Tick;
    _timer.Interval = TimeSpan.FromMilliseconds(TimerInterval);
    _timer.Start();
}
1 reference
private void StartGame_Click(object sender, RoutedEventArgs e)
{
    InitialGame();
    StartGame.Visibility = Visibility.Collapsed;
}
2 references
private void EndGame()
{
    _timer.Stop();
    RestartButton.Visibility = Visibility.Visible;
}
```

Рисунок 3. Методы инициализации окна приложения и графических компонентов

В первом методе происходит инициализация компонентов окна, загрузка XAML-разметки и установка начальных значений. Сразу после него объявляется обработчик событий, вызывающий метод «InitialGame» для начала игры, когда игровой холст загружен: создаётся голова змеи, добавляется на игровой холст, размещается подбираемый предмет, задаётся и настраивается таймер обновления игрового состояния. Для старта игры используется кнопка запуска со своим обработчиком события нажатия на кнопку «StartGame_Click» - вызывается метод «InitialGame» и происходит сокрытие кнопки. Для завершения игрового процесса используется функция «EndGame», в котором происходит остановка таймер и кнопка перезапуска становится видимой.

Управление персонажем определяется методом «CalculateNewHeadPosition» (рис. 4).

```
private Point CalculateNewHeadPosition()
{
    double left = Canvas.GetLeft(_snakeHead) / SnakeSquareSize;
    double top = Canvas.GetTop(_snakeHead) / SnakeSquareSize;

    Point headCurrentPosition = new Point(left, top);
    Point newHeadPosition = new Point();

    switch (_direction)
    {
        case Direction.Left:
            newHeadPosition = new Point(headCurrentPosition.X - 1, headCurrentPosition.Y);
            break;
        case Direction.Right:
            newHeadPosition = new Point(headCurrentPosition.X + 1, headCurrentPosition.Y);
            break;
        case Direction.Top:
            newHeadPosition = new Point(headCurrentPosition.X, headCurrentPosition.Y - 1);
            break;
        case Direction.Bottom:
            newHeadPosition = new Point(headCurrentPosition.X, headCurrentPosition.Y + 1);
            break;
    }
    return newHeadPosition;
}
```

Рисунок 4. Функция движения змеи в различные стороны

В общем метод отвечает за расчет новой позиции головы змеи в зависимости от текущего направления движения. Логика функции начинается с получения текущих координат головы змеи на игровом холсте и нормализации их, деля на размер одного сегмента змеи, определённого в константе в начале класса. С помощью объекта «Point» представляется текущая позиция головы змеи. Для того, чтобы хранить информацию о новой позиции головы, создаётся новый объект такого же типа. Затем происходит расчёт новой позиции в зависимости от направления движения, которое будет определяться по нажатию той или иной клавиши (рис. 5).

```
private void Window_KeyDown(object sender, KeyEventArgs e)
{
    switch (e.Key)
    {
        case Key.Up:
            if (_direction != Direction.Bottom)
            {
                _direction = Direction.Top;
            }
            break;

        case Key.Down:
            if (_direction != Direction.Top)
            {
                _direction = Direction.Bottom;
            }
            break;

        case Key.Left:
            if (_direction != Direction.Right)
            {
                _direction = Direction.Left;
            }
            break;

        case Key.Right:
            if (_direction != Direction.Left)
            {
                _direction = Direction.Right;
            }
            break;
    }
}
```

Рисунок 5. Метод нажатия на клавишу

Управление строится на базе конструкции «switch-case», где отслеживается нажатие на конкретную клавишу и выполнение присвоение направления движения для метода «CalculateNewHeadPosition».

Манипуляция змеёй настроена и готова к работе. Остаётся настроить процесс «увеличение» змеи за счёт того, что она будет касаться специальных объектов в виде изображения (рис. 6).

```
private void PlaceFood()
{
    int maxX = (int)(GameCanvas.ActualWidth / SnakeSquareSize);
    int maxY = (int)(GameCanvas.ActualHeight / SnakeSquareSize);
    int foodX = RandomPositionFood.Next(0, maxX);
    int foodY = RandomPositionFood.Next(0, maxY);

    _foodPosition = new Point(foodX, foodY);
    Image foodImage = new Image
    {
        Width = SnakeSquareSize,
        Height = SnakeSquareSize,
        Source = new BitmapImage(new Uri("D:\\Projects\\VS2022\\WpfApp1\\images\\aple.png"))
    };

    Canvas.SetLeft(foodImage, foodX * SnakeSquareSize);
    Canvas.SetTop(foodImage, foodY * SnakeSquareSize);
    GameCanvas.Children.Add(foodImage);
}
1 reference
private void EatFood()
{
    _score++;
    ScoreTextBlock.Text ="Очки: " + _score.ToString();

    GameCanvas.Children.Remove(GameCanvas.Children.OfType<Image>().FirstOrDefault());

    Rectangle newSnake = CreateSnakeSegment(_foodPosition);
    _snake.Add(newSnake);
    GameCanvas.Children.Add(newSnake);
}

2 references
private Rectangle CreateSnakeSegment(Point position)
{
    Rectangle rectangle = new Rectangle
    {
        Width = SnakeSquareSize,
        Height = SnakeSquareSize,
        Fill = _snakeColor
    };
    Canvas.SetLeft(rectangle, position.X * SnakeSquareSize);
    Canvas.SetTop(rectangle, position.Y * SnakeSquareSize);

    return rectangle;
}
```

Рисунок 6. Методы появления предметов и их исчезновение при касании, а также метод увеличения размера змеи

Метод «PlaceFood» отвечает за создание предмета для подбора, загрузку его изображения, для чего используется объект класса «Image» для работы с картинками, и дальнейшее его размещения на игровом поле (Canvas).

Для расчёта касания друг с другом используется функция «EatFood». В ней происходит увеличения счётчика очков на единицу, а также вывод в текстовый элемент. Затем изображение предмета удаляется и перемещается в случайное положение, благодаря соответствующим командам в предыдущем методе. После исчезновения происходит «рост» змеи: в список с сегментами змеи добавляется новый элемент, который и является новой частью.

Непосредственно сам рост определяется «CreateSnakeSegment», где создаётся новый прямоугольник заданного цвета и установка его позиции относительно головы змеи (относительно индекса списка _snake).

Для того, чтобы все функции выполнялись постоянно используется метод «Timer_Tick» (рис. 7).

```
private void Timer_Tick(object sender, EventArgs e)
{
    Point newHeadPosition = CalculateNewHeadPosition();

    if (newHeadPosition == _foodPosition)
    {
        EatFood();
        PlaceFood();
    }

    if (newHeadPosition.X < 0 || newHeadPosition.Y < 0
        || newHeadPosition.X >= GameCanvas.ActualWidth / SnakeSquareSize
        || newHeadPosition.Y >= GameCanvas.ActualHeight / SnakeSquareSize)
    {
        EndGame();
        return;
    }

    if (_snake.Count >= 4)
    {
        for (int i = 0; i < _snake.Count; i++)
        {
            Point currentPos = new Point(Canvas.GetLeft(_snake[i]), Canvas.GetTop(_snake[i]));

            for (int j = i + 1; j < _snake.Count; j++)
            {
                Point nextPos = new Point(Canvas.GetLeft(_snake[j]), Canvas.GetTop(_snake[j]));

                if (currentPos == nextPos)
                {
                    EndGame();
                    return;
                }
            }
        }
    }


    for (int i = _snake.Count - 1; i > 0; i--)
    {
        Canvas.SetLeft(_snake[i], Canvas.GetLeft(_snake[i - 1]));
        Canvas.SetTop(_snake[i], Canvas.GetTop(_snake[i - 1]));
    }

    Canvas.SetLeft(_snakeHead, newHeadPosition.X * SnakeSquareSize);
    Canvas.SetTop(_snakeHead, newHeadPosition.Y * SnakeSquareSize);
}
```

Рисунок 7. Метод игрового процесса

В данном сегменте кода вызываются все вышеописанные функции, происходят проверки между головой змеи и предметом, с границами игрового поля и другими сегментами змеи.

После проделанных манипуляций проводится проверка и тестирование приложения (рис. 8).

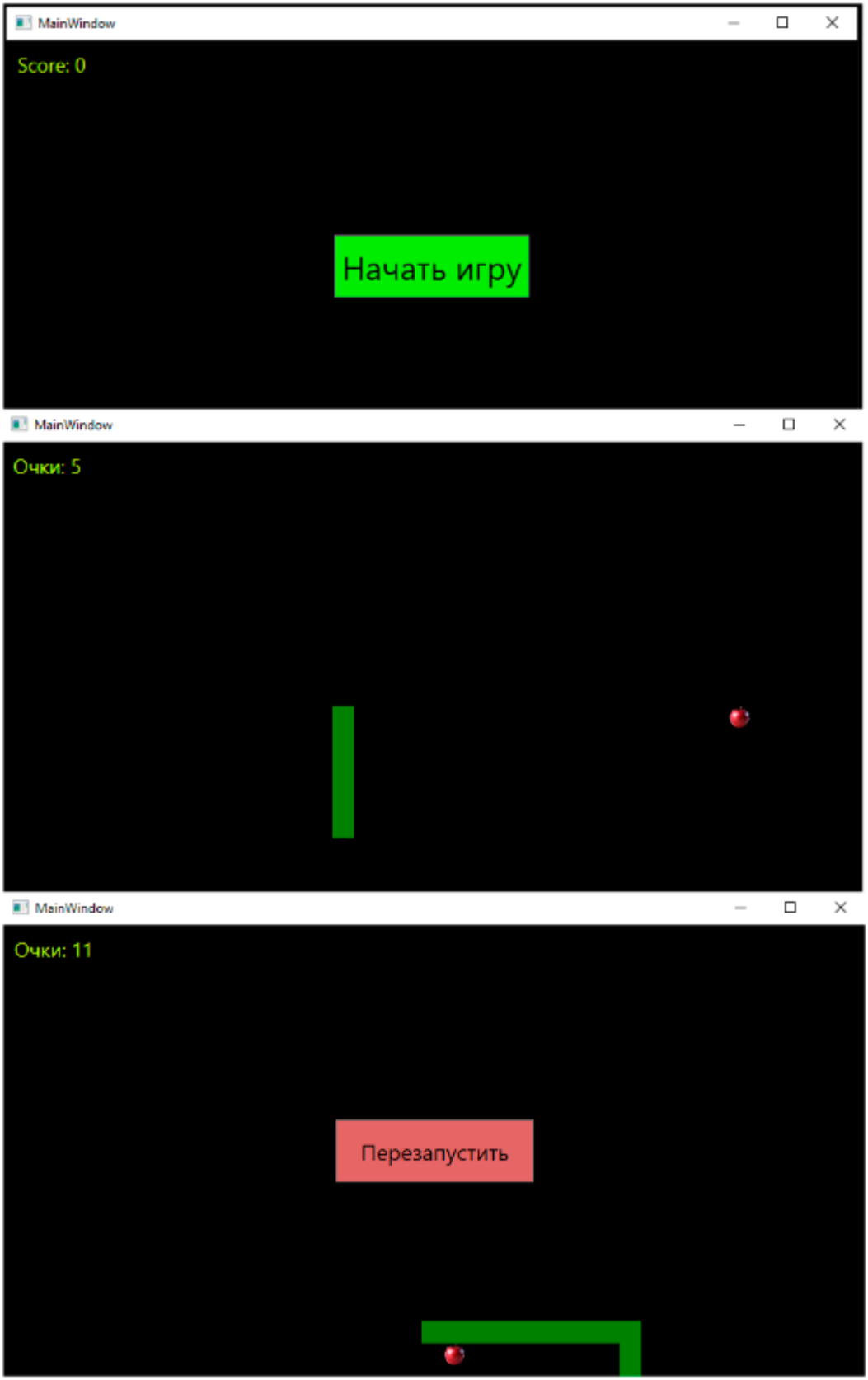

Рисунок 8. Результат работы приложения

В результате исследования было разработано графическое приложение на базе платформы WPF и языка программирования C#. Приложение

представлено в виде динамической классической игры «Змейка» с подробным описание работы приложения.

Библиографический список

1. Писанко А.В., Жилина Е.В. Реализация графических интерфейсов на основе технологии WPF // Интернаука. 2020. № 47-1 (176). С. 12-14.
2. Войшнис К.А. Эффективность использования приложений, созданных на платформах JS иWPF // В сборнике: Вестник экономического научного общества студентов и аспирантов. Межвузовский студенческий научный журнал. Санкт-Петербург, 2022. С. 36-39.
3. Рудишин И.А. Средства WPF для реализации интерфейсов пользовательских приложений // Труды Братского государственного университета. Серия: Естественные и инженерные науки. 2013. Т. 2. С. 53-55.
4. Ковальчук А.А., Павленко Е.П., Айвазов В.А. Применение программных технологий WPF и Silverlight при разработке информационных систем // Восточно-Европейский журнал передовых технологий. 2009. Т. 1. № 2 (37). С. 21-25.
5. Васюта С.Д. Достоинства технологии WPF и использование её для реализации игрового приложения Windows // В сборнике: Студенческая наука для развития информационного общества. Материалы II Всероссийской научно-технической конференции. 2015. С. 153-155.