УДК 004.032.26

Создание игры «Space Invaders» в Visual Studio

Ульянов Егор Андреевич
Приамурский государственный университет имени Шолом-Алейхема
Студент

Аннотация
Целью данной статьи является, применение Windows Presentation Foundation (WPF) и языка программирования C#, для создания классической игры «Space Invaders». Практическим результатом является разработанная игра.
Ключевые слова: C#, Visual Studio, игра космические захватчики

Creating the game "Space Invaders" in Visual Studio

Ulianov Egor Andreevich
Sholom-Aleichem Priamursky State University
Student

Abstract
The purpose of this article is to use the Windows Presentation Foundation (WPF) and the C# programming language to create the classic game "Space Invaders". The practical result is a developed game.
Keywords: C#, Visual Studio, game Space Invaders

1. Введение

1.1 Актуальность исследования

Самым популярным видом досуга многих людей на сегодняшний момент являются компьютерные игры. Виртуальные миры открывают нам огромные возможности: позволяют отыгрывать различные роли, играя за главного героя, многие игры способствуют развитию интеллекта, внимания, реакции, пространственной ориентации и логического мышления. Разработкой видеоигр может заниматься как один человек, так и команда. Создание игры - это продолжительный и трудоёмкий процесс, состоящий из самых разнообразных этапов, включающий в себя как технические, так и творческие моменты.

1.2 Обзор исследований

В своей работе И. Ю. Просвирнина создала приложение «Морской бой», обладающее игровым искусственным интеллектом, в котором предусмотрен режим игры «Игрок против компьютера» [1]. В статье Н. А. Базеевой и Д. С. Лебедева описано начало игровой индустрии, а также рассмотрены особенности языков программирования для разработки игр [2].

И.А. Савин, О.В. Батенькина рассмотрели процесс написания скриптовых сценариев при разработке виртуального тренажера[3]. В своей работе Э.Р. Гараева, И.И. Бикмуллина, И.А. Барков описали возможности Unity 3D на предмет создания 3D-моделей[4]. С.А. Суродин в своей статье представил сценарий углубленного изучения одного из лучших движков, существующих на данный момент, для создания красивых 2D и 3D игр[5]. В своей работе Р.Ф. Гайнуллин, В.А. Захаров, Е.А. Аксенова изучили инструмент для разработки двух- и трёхмерных игр – Unity 3D [6].

1.3 Цель исследования

Цель исследования – применяя возможности среды разработки Visual Studio и языка программирования C#, создать классическую игру «Space Invaders», со всеми, необходимыми для полноценной игры, механиками.

2. Материалы и методы

2.1 Данные

Графика для такой простой игры, может быть создана в любом редакторе. В данном примере использовался Adobe Photoshop.

2.2 Методы исследования

Для создания игры будем использовать программное обеспечение Visual Studio, а также язык программирования C#.

Для проекта использовалась Windows Presentation Foundation (WPF), платформа пользовательского интерфейса для создания клиентских приложений для настольных систем. Платформа разработки WPF поддерживает широкий набор компонентов для разработки приложений, включая модель приложения, ресурсы, элементы управления, графику, макет, привязки данных, документы и безопасность. Эта платформа является частью платформы .NET.

3 Результаты и дискуссия

Начнём разработку игры, с создания WPF приложения. Назовем проект «Space Invaders» (рис. 1).

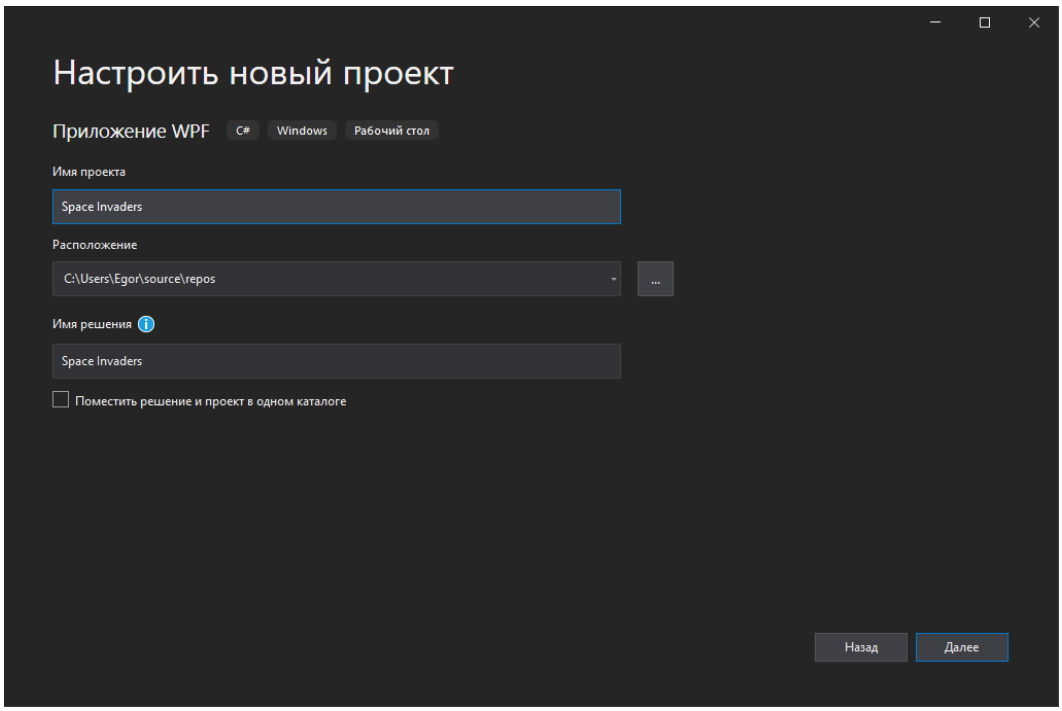

Рисунок 1. Создание проекта

Для начала добавим готовый набор графики в проект, для этого создадим папку (рис.2-4).

Рисунок 2. Набор графики

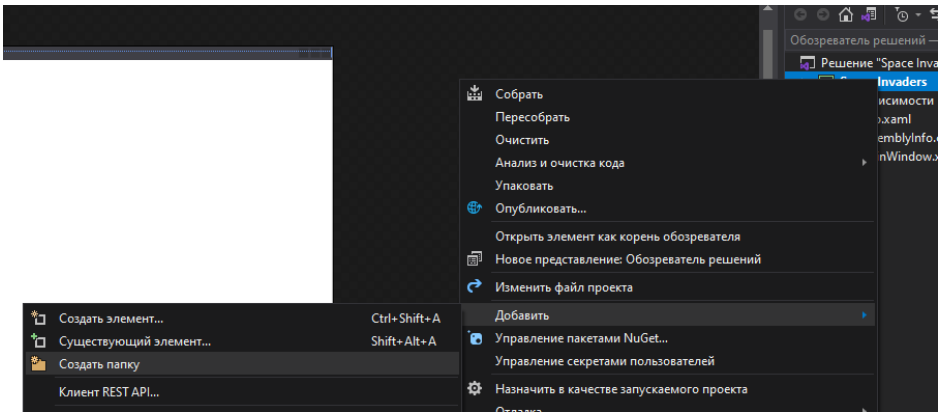

Рисунок 3. Создание папки с ресурсами игры

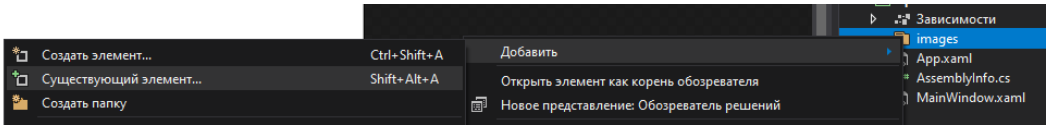

Рисунок 4. Добавление в папку изображений

Для удобства изменим рабочее пространство, а также высоту игрового экрана до 500 единиц (рис.5).

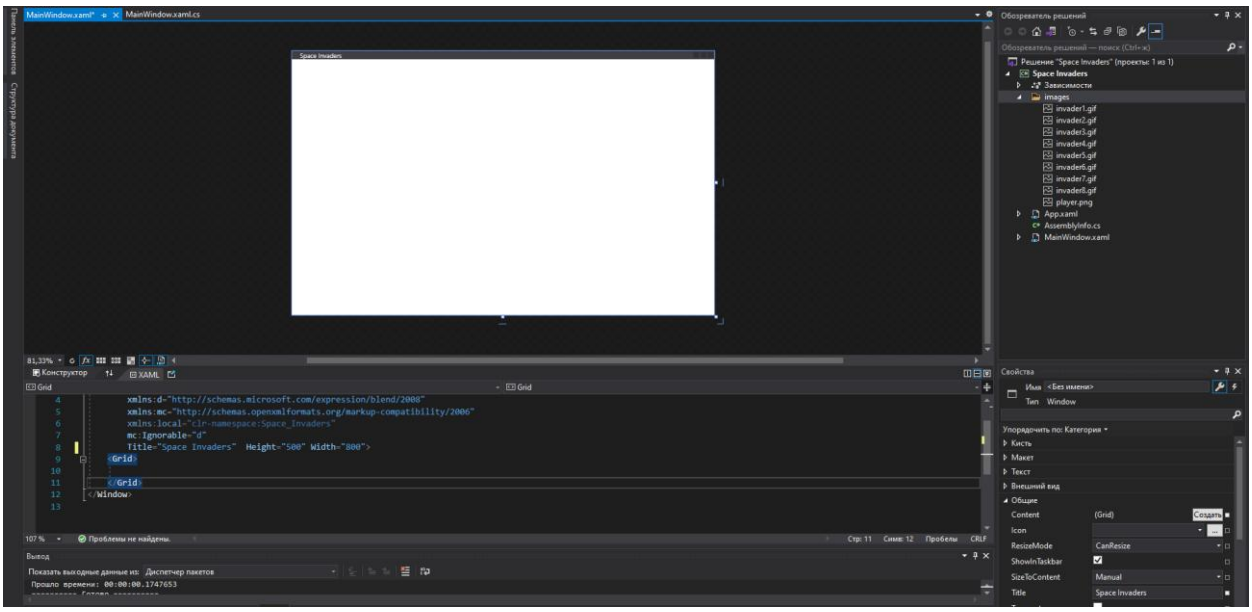

Рисунок 5. Изменение рабочего пространства

Сейчас игровой экран представляет собой сетку (Grid), что для игры не подходит, меняем «Grid» на «Canvas», называем его любым именем, меняем фон на черный, а команда «Focusable="True"» позволит сосредоточить экран только на игре. Также добавим два события, на клавиши клавиатуры, вверх и вниз. (рис.6).

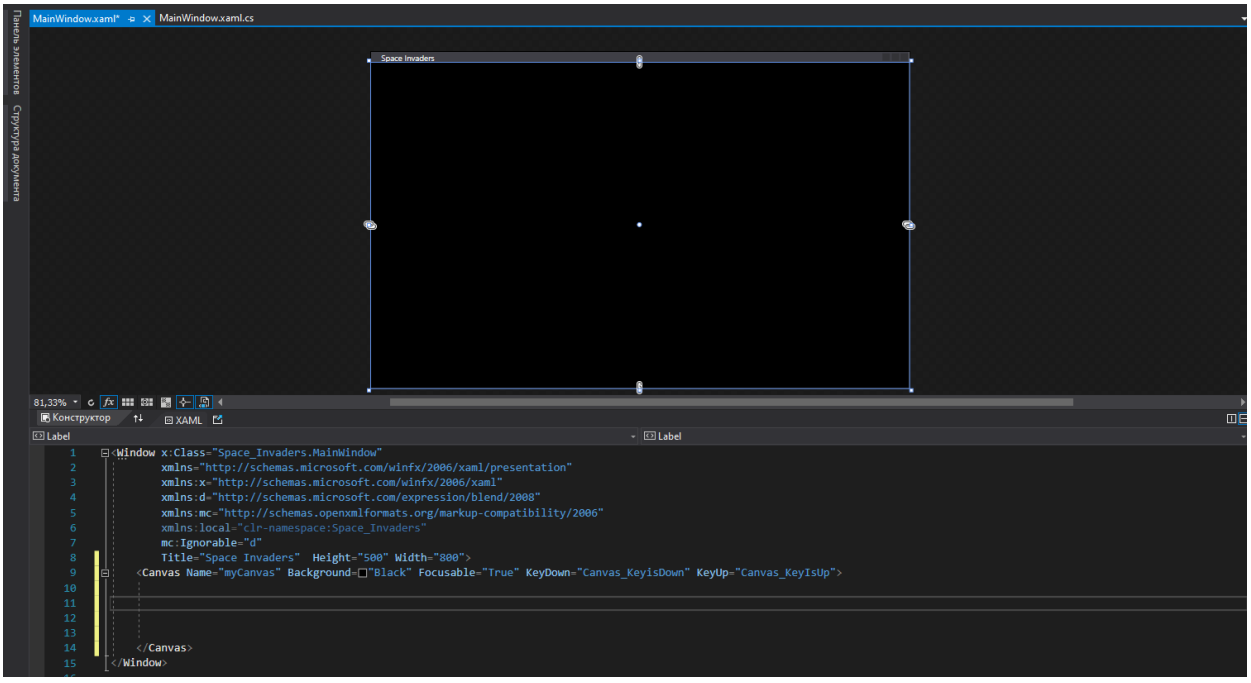

Рисунок 6. Выявление значения по умолчания

В верхний левый угол добавим счетчик противников, и на экран добавим будущего игрока (рис.7-8).

```
<Canvas Name="myCanvas" Background="Black" Focusable="True" KeyDown="Canvas_KeyisDown" KeyUp="Canvas_KeyIsUp">

    <Label Foreground="White" Name="enemiesLeft" FontSize="16" FontWeight="ExtraBold">Enemies Left:</Label>

    <Rectangle Name="player1" Fill="White" Height="65" Width="55" Canvas.Left="568" Canvas.Top="394" />


</Canvas>
```

Рисунок 7. Создание счетчика и модели игрока

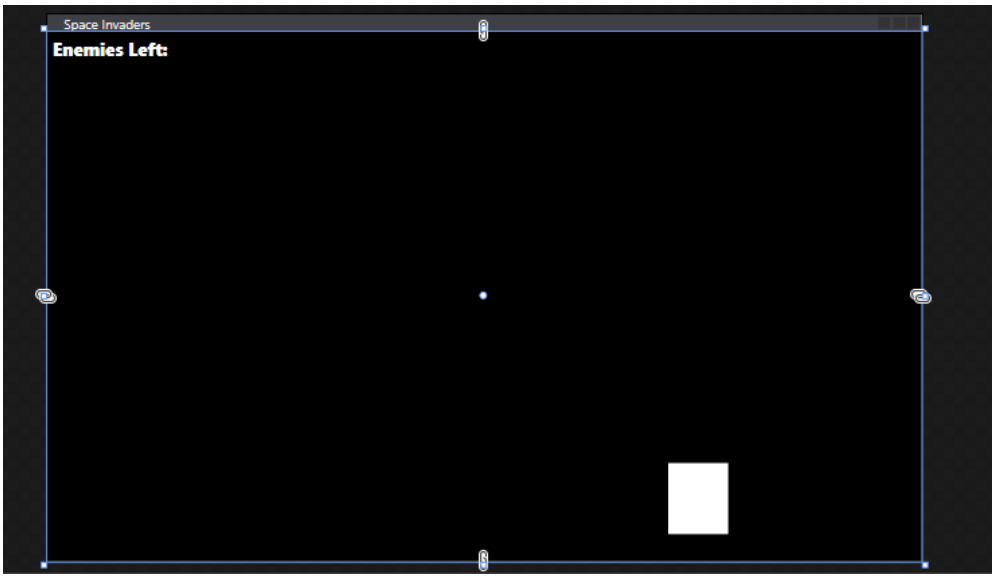

Рисунок 8. Создание счетчика и модели игрока

Приступаем к написанию логики игры. Для этого добавим обработку нажатий клавиш, кликаем правой кнопкой мыши на «KeyDown="Canvas_KeyisDown"», «KeyUp="Canvas_KeyIsUp"» и нажимаем перейти к определению. Также добавим три функции: функция вражеской стрельбы, функция создания врагов, которая будут генерировать вражеские спрайты для игры, и функция игрового движка, которая будет управлять движением, стрельбой по врагам, и анимацией космических захватчиков на экране (рис.9).

```
13      ng System.Windows.Navigation;
14      ng System.Windows.Shapes;
15      ng System.Windows.Threading;
16
17      espace Space_Invaders
18
19      /// <summary>
20      /// Interaction logic for MainWindow.xaml
21      /// </summary>
        Ссылок: 2
22      public partial class MainWindow : Window
23      {
            Ссылок: 0
24          public MainWindow()
25          {
26              InitializeComponent();
27          }
28
            ссылка: 1
29          private void Canvas_KeyisDown(object sender, KeyEventArgs e)
30          {
31
32          }
33
            ссылка: 1
34          private void Canvas_KeyIsUp(object sender, KeyEventArgs e)
35          {
36
37          }
            Ссылок: 0
38          private void makeEnemies(int limit)
39          {
40
41          }
42
            Ссылок: 0
43          private void gameEngine(object sender, EventArgs e)
44          {
45
46          }
47      }
48
```

Рисунок 9. Создание функций

Далее создаем несколько переменных. Первые два логических значения «goLeft» и «goRight» будут контролировать движение игрока, затем идет список «itemstoremove», этот список будет содержать все дополнительные, непостоянные прямоугольные формы, которые будут в этой игре, например, «использованные» пули, которые будут удаляться, когда не нужны. Изображения врагов, таймер пули, ограничение таймера пули и скорость врага - все это целочисленные значения для игры. Таймер диспетчера - это основной таймер игры, в котором содержаться инструкции и правила данной игры, включая то, как игрок может выиграть и проиграть. Кисть изображения - это класс текстуры изображения, который используется для применения различных изображений к персонажам игры (рис.10).

```
public partial class MainWindow : Window
{

    bool goLeft, goRight = false;

    List<Rectangle> itemstoremove = new List<Rectangle>();

    int enemyImages = 0;
    //таймер вражеской пули
    int bulletTimer;
    //предел и частота таймера вражеской пули
    int bulletTimerLimit = 90;
    // сохранение общего количества врагов
    int totalEnemeis;

    DispatcherTimer dispatcherTimer = new DispatcherTimer();

    ImageBrush playerSkin = new ImageBrush();
    // скорость противника по умолчанию
    int enemySpeed = 6;
```

Рисунок 10. Добавление переменных

Переходим к добавлению главной функции. Для начала привязываем тик таймера диспетчера к событию игрового движка, которое создалось ранее. Время интервала устанавливаем 20 миллисекунд, а затем запускаем таймер. Изображение игрока связываем с файлом в папке с ресурсами игры, указывая путь, затем применяем заливку к прямоугольнику player1, который находиться на «Canvas». В строке «makeEnemies(30)» говорится о запуске функции создания врагов, это означает, что функция создаст 30 различных космических захватчиков для игры (рис.11).

```
Ссылок: 0
public MainWindow()
{
    InitializeComponent();


    dispatcherTimer.Tick += gameEngine;
    dispatcherTimer.Interval = TimeSpan.FromMilliseconds(20);
    dispatcherTimer.Start();
    playerSkin.ImageSource = new BitmapImage(new Uri("C:/Users/Egor/source/repos/Space Invaders/Space Invaders/images/player.png"));
    player1.Fill = playerSkin;
    makeEnemies(30);


}
```

Рисунок 11. Создание главной функции

Далее необходимо дополнить код обработчиков нажатий клавиш. Внутри этой функции есть два оператора «if», сначала проверяем, нажата ли клавиша влево, если да, то устанавливается логическое значение «go left» в значение «true», и то же самое для клавиши вправо, с логическим значением «go right» (рис.12).

```
ссылка: 1
private void Canvas_KeyisDown(object sender, KeyEventArgs e)
{


    if (e.Key == Key.Left)
    {
        goLeft = true;
    }
    if (e.Key == Key.Right)
    {
        goRight = true;
    }


}
```

Рисунок 12. Обработка нажатия клавиш влево или вправо

В данную функцию также добавим выстрел на клавишу пробел (рис.13).

```
ссылка: 1
private void Canvas_KeyIsUp(object sender, KeyEventArgs e)
{
    if (e.Key == Key.Left)
    {
        goLeft = false;
    }
    if (e.Key == Key.Right)
    {
        goRight = false;
    }

    if (e.Key == Key.Space)
    {

        itemstoremove.Clear();


        Rectangle newBullet = new Rectangle
        {
            Tag = "bullet",
            Height = 20,
            Width = 5,
            Fill = Brushes.White,
            Stroke = Brushes.Red
        };


        Canvas.SetTop(newBullet, Canvas.GetTop(player1) - newBullet.Height);
        Canvas.SetLeft(newBullet, Canvas.GetLeft(player1) + player1.Width / 2);
        myCanvas.Children.Add(newBullet);

    }
}
```

Рисунок 13. Обработка отпускания клавиш

Задаем настройки для вражеских снарядов (рис.14).

```
ссылка: 1
private void enemyBulletMaker(double x, double y
{

    Rectangle newEnemyBullet = new Rectangle
    {
        Tag = "enemyBullet",
        Height = 40,
        Width = 15,
        Fill = Brushes.Yellow,
        Stroke = Brushes.Black,
        StrokeThickness = 5

    };

    Canvas.SetTop(newEnemyBullet, y);
    Canvas.SetLeft(newEnemyBullet, x);
    myCanvas.Children.Add(newEnemyBullet);
}
```

Рисунок 14. Функция вражеских выстрелов

Далее дополняем функцию противников, умением стрелять, а также подключаем изображения для индивидуальности каждого врага (рис.15-16).

```
private void makeEnemies(int limit)
{

    int left = 0;
    totalEnemeis = limit;

    for (int i = 0; i < limit; i++)
    {
        ImageBrush enemySkin = new ImageBrush();
        Rectangle newEnemy = new Rectangle
        {
            Tag = "enemy",
            Height = 45,
            Width = 45,
            Fill = enemySkin,

        };

        Canvas.SetTop(newEnemy, 10);
        Canvas.SetLeft(newEnemy, left);

        myCanvas.Children.Add(newEnemy);

        left -= 60;

        enemyImages++;
        if (enemyImages > 8)
        {
            enemyImages = 1;
        }

        switch (enemyImages)
        {
            case 1:
                enemySkin.ImageSource = new BitmapImage(new Uri("C:/Users/Egor/source/repos/Space Invaders/Space Invaders/images/invader1.gif"));
                break;
            case 2:
                enemySkin.ImageSource = new BitmapImage(new Uri("C:/Users/Egor/source/repos/Space Invaders/Space Invaders/images/invader2.gif"));
                break;
```

Рисунок 15. Функция логики противника

```
    case 3:
        enemySkin.ImageSource = new BitmapImage(new Uri("C:/Users/Egor/source/repos/Space Invaders/Space Invaders/images/invader3.gif"));
        break;
    case 4:
        enemySkin.ImageSource = new BitmapImage(new Uri("C:/Users/Egor/source/repos/Space Invaders/Space Invaders/images/invader4.gif"));
        break;
    case 5:
        enemySkin.ImageSource = new BitmapImage(new Uri("C:/Users/Egor/source/repos/Space Invaders/Space Invaders/images/invader5.gif"));
        break;
    case 6:
        enemySkin.ImageSource = new BitmapImage(new Uri("C:/Users/Egor/source/repos/Space Invaders/Space Invaders/images/invader6.gif"));
        break;
    case 7:
        enemySkin.ImageSource = new BitmapImage(new Uri("C:/Users/Egor/source/repos/Space Invaders/Space Invaders/images/invader7.gif"));
        break;
    case 8:
        enemySkin.ImageSource = new BitmapImage(new Uri("C:/Users/Egor/source/repos/Space Invaders/Space Invaders/images/invader8.gif"));
        break;
}
```

Рисунок 16. Продолжение функции

Последним этапом, пропишем правила игры: смерть от касания с противником, проверка на попадание, сборщик и утилизатор мусора и так далее (рис.17-20).

```
private void gameEngine(object sender, EventArgs e)
{

    Rect player = new Rect(Canvas.GetLeft(player1), Canvas.GetTop(player1), player1.Width, player1.Height);

    enemiesLeft.Content = "Invaders Left: " + totalEnemeis;

    if (goLeft && Canvas.GetLeft(player1) > 0)
    {
        Canvas.SetLeft(player1, Canvas.GetLeft(player1) - 10);
    }

    else if (goRight && Canvas.GetLeft(player1) + 80 < Application.Current.MainWindow.Width)
    {
        Canvas.SetLeft(player1, Canvas.GetLeft(player1) + 10);
    }



    bulletTimer -= 3;


    if (bulletTimer < 0)
    {
        enemyBulletMaker((Canvas.GetLeft(player1) + 20), 10);
        bulletTimer = bulletTimerLimit;
    }
    if (totalEnemeis < 10)
    {
        enemySpeed = 20;
    }

    foreach (var x in myCanvas.Children.OfType<Rectangle>())
    {
```

Рисунок 17. Функции правил игры

```
if (x is Rectangle && (string)x.Tag == "bullet")
{

    Canvas.SetTop(x, Canvas.GetTop(x) - 20);

    Rect bullet = new Rect(Canvas.GetLeft(x), Canvas.GetTop(x), x.Width, x.Height);

    if (Canvas.GetTop(x) < 10)
    {
        itemstoremove.Add(x);
    }

    foreach (var y in myCanvas.Children.OfType<Rectangle>())
    {

        if (y is Rectangle && (string)y.Tag == "enemy")
        {

            Rect enemy = new Rect(Canvas.GetLeft(y), Canvas.GetTop(y), y.Width, y.Height);

            if (bullet.IntersectsWith(enemy))
            {

                itemstoremove.Add(x);
                itemstoremove.Add(y);
                totalEnemeis -= 1;
            }
        }


    }

}
```

Рисунок 18. Продолжение функции

```
if (x is Rectangle && (string)x.Tag == "enemy")
{
    Canvas.SetLeft(x, Canvas.GetLeft(x) + enemySpeed);

    if (Canvas.GetLeft(x) > 820)
    {

        Canvas.SetLeft(x, -80);

        Canvas.SetTop(x, Canvas.GetTop(x) + (x.Height + 10));
    }


    Rect enemy = new Rect(Canvas.GetLeft(x), Canvas.GetTop(x), x.Width, x.Height);

    if (player.IntersectsWith(enemy))
    {

        dispatcherTimer.Stop();
        MessageBox.Show("you lose");
    }
}


if (x is Rectangle && (string)x.Tag == "enemyBullet")
{

    Canvas.SetTop(x, Canvas.GetTop(x) + 10);

    if (Canvas.GetTop(x) > 480)
    {
        itemstoremove.Add(x);

    }

    Rect enemyBullets = new Rect(Canvas.GetLeft(x), Canvas.GetTop(x), x.Width, x.Height);
```

Рисунок 19. Продолжение функции

```
            if (enemyBullets.IntersectsWith(player))
            {
                dispatcherTimer.Stop();
                MessageBox.Show("you lose");
            }
        }
    }

    foreach (Rectangle y in itemstoremove)
    {
        myCanvas.Children.Remove(y);
    }

    if (totalEnemeis < 1)
    {
        dispatcherTimer.Stop();
        MessageBox.Show("you win");
    }
}
```

Рисунок 20. Продолжение функции

Осталось только проверить игру. Жмем кнопку «Play» (рис.21-23).

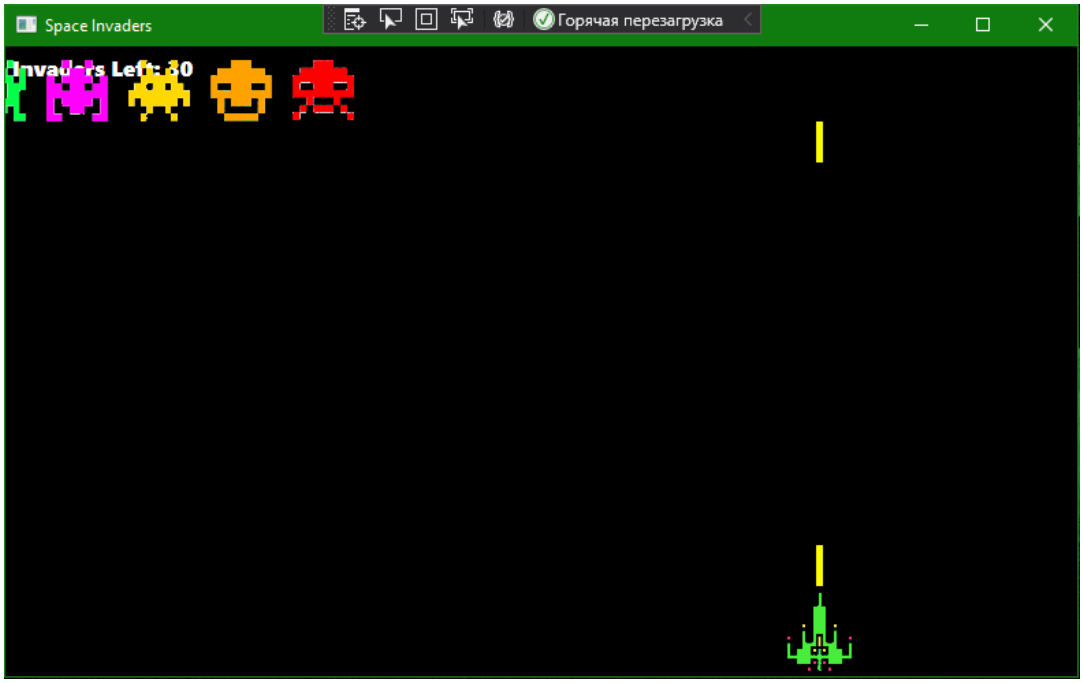

Рисунок 21. Игровой процесс

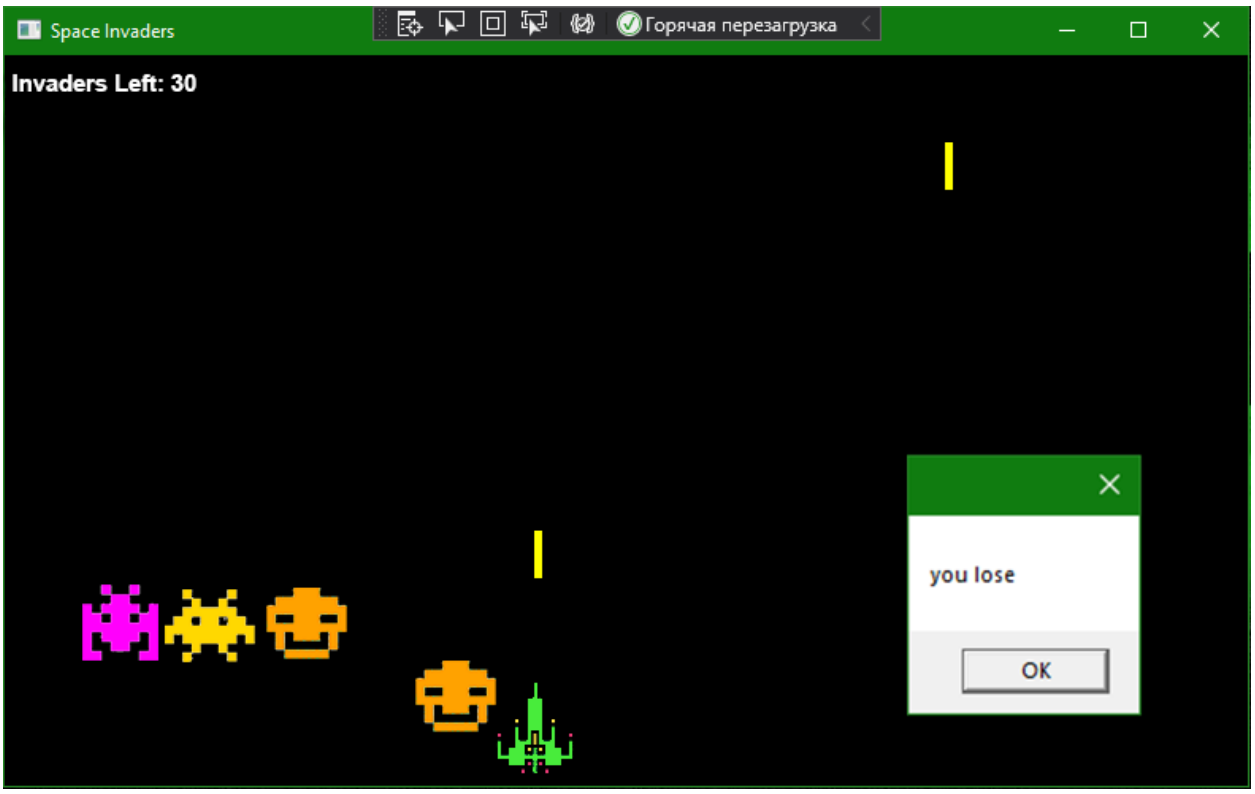

Рисунок 22. Поражение игрока

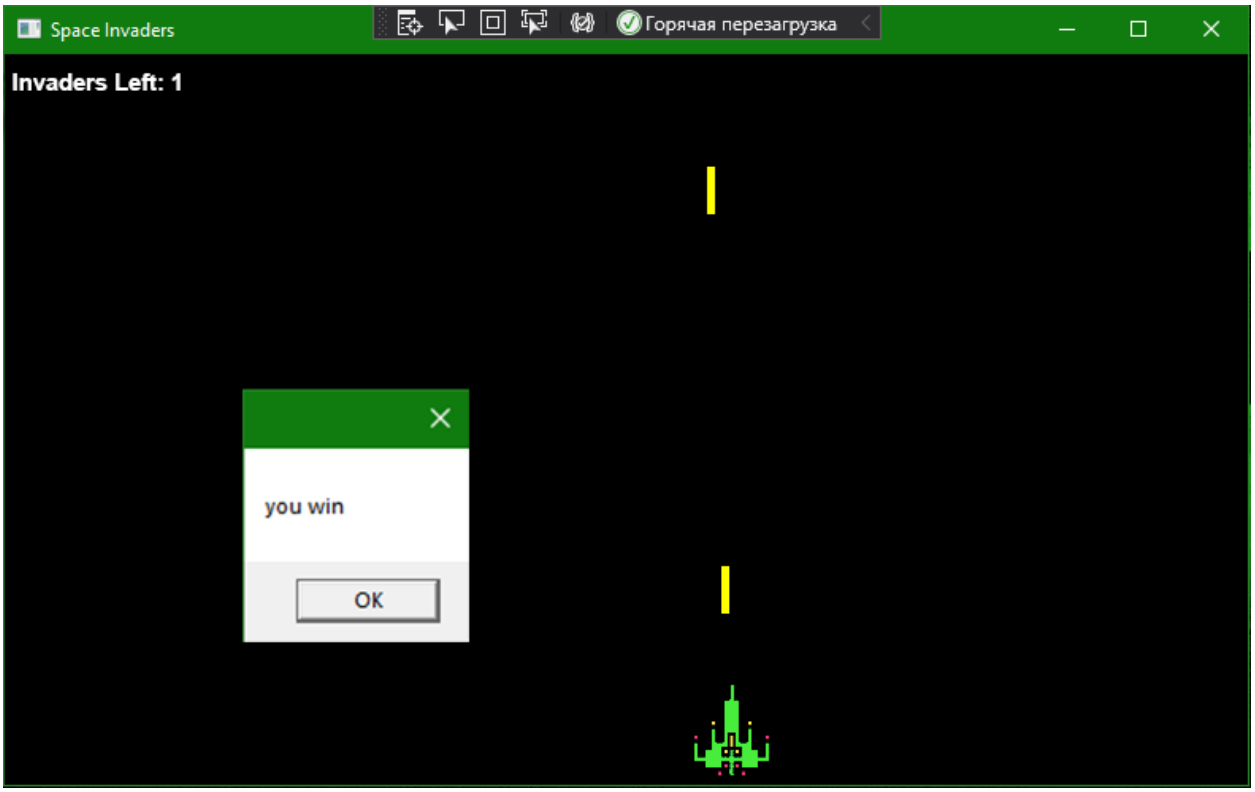

Рисунок 23. Победа игрока

4 Выводы

Были проанализированы существующие аналоги и методы разработки, а также выбрана среда разработки. Для реализации поставленной задачи отлично подошла разработка с помощью Visual Studio и языка программирования C#. Такой выбор заметно упростил разработку игры, так как в интернете имеется достаточное количество документации. Во время создания игры был полученный ценный опыт работы с этим средством разработки. В итоге была разработана классическая игра «Space Invaders». Тесты игра прошла хорошо, были исправлены незначительные ошибки. Созданная игра имеет потенциал к развитию, а именно: добавление новых функций; улучшение интерфейса; увеличение количества контента, добавление мультиплеера.

Библиографический список

1. Просвирнина И. Ю., Егунова А. И., Аббакумов А. А. Среда разработки Microsoft Visual Studio на примере создания игры "морской бой" //Интеграционные процессы в науке в современных условиях. 2017. С. 123-125.
2. Базеева Н. А., Лебедев Д. С. Языки программирования для создания игр //E-Scio. 2019. №4. С. 31-39.
3. Савин И.А., Батенькина О.В. Написание скриптов для трехмерного графического движка // Визуальная культура: дизайн, реклама, информационные технологии. 2018. № 12-7 (28). С. 7-15.
4. Долженко А.И., Глушенко С.А. Особенности подготовки 3D-объектов, смоделированных в Blender, для импорта в Unity 3D // Прикаспийский журнал: управление и высокие технологии. 2014. №6. С. 92-96.
5. Суродин С.А. Unity 3D. разработка сценария проектирования в среде Unity 3D// Информатика и вычислительная техника. 2015. №3. С. 504-511.
6. Гайнуллин Р.Ф., Захаров В.А., Аксенова Е.А. Создание 2d игры на Unity 3D 5.4 // Вестник современных исследований. 2018. №4. С. 78-82.