Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования

ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

СИСТЕМ УПРАВЛЕНИЯ И РАДИОЭЛЕКТРОНИКИ (ТУСУР)

Кафедра автоматизированных систем управления (АСУ)

**ПРОГРАММА «SPACE INVADERS» НА ЯЗЫКЕ С#**

Пояснительная записка к курсовому проекту   
по дисциплине «Объектно-ориентированное программирование»

Студент гр. 449-1

\_\_\_\_\_\_\_\_\_\_\_ А.С. Новикова

«\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_2021 г.

Руководитель

доцент каф. АСУ, к.т.н:

\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_ С.М. Алфёров

«\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_2021 г.

Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования

ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ СИСТЕМ УПРАВЛЕНИЯ И РАДИОЭЛЕКТРОНИКИ (ТУСУР)

Кафедра автоматизированных систем управления (АСУ)

ЗАДАНИЕ К КУРСОВОМУ ПРОЕКТУ

по дисциплине «Объектно-ориентированное программирование»

Студенту группы 449-1 Новиковой Анне Сергеевне

1. Тема работы: Создание программы “Space Invaders” на C#.
2. Срок сдачи студентом законченной работы 31.05. 2021 г.
3. Исходные данные: Описание игры, зображение корабля и монстров.
4. Перечень подлежащих разработке вопросов:
   1. Описание среды разработки;
   2. Описание языка программирования;
   3. Описание основных операторов и объектов;
   4. Описание основных функций;
5. Требования:
   1. Программа должна работать под ОС Windows.
   2. Интерфейс пользователя должен быть оконный.
   3. Требования к отчету ОС ТУСУР 01 2013.
6. Дата выдачи задания: 08.02. 2021 г.

Руководитель доцент каф. АСУ, к.т.н. \_\_\_\_\_\_\_\_\_\_ Алфёров С.М.

Задание принял к использованию \_\_\_\_\_\_\_\_\_\_\_\_ Новикова А.С. «\_\_» \_\_\_\_\_\_\_ 20\_\_ г.

**Реферат**

Данный отчет содержит: 29 страниц, 11 рисунков, 1 приложение, 2 источника.

Ключевые слова: ПРИЛОЖЕНИЕ, SPACE INVADERS, С#, VISUAL STUDIO.

Цель работы – получить навыки объектно-ориентированного программирования

**Оглавление**

[**Введение**](#_heading=h.ttobzamldw9n) **4**

[**1 Техническое задание**](#_heading=h.1fob9te) **5**

[**2 Обзор средств реализации**](#_heading=h.sywuxiccskrx) **7**

[2.1 Описание среды разработки](#_heading=h.2et92p0) 8

[2.2 Обзор языка программирования и технологии](#_heading=h.grc3yxugw7k1) 9

[**3 Описание проекта**](#_heading=h.1t3h5sf) **9**

[3.1 Классы](#_heading=h.qq60uq9em82q) 11

[3.2 Структуры данных](#_heading=h.3nsx5dpm8c7w) 12

[3.2.1 Библиотечные структуры данных](#_heading=h.17dp8vu) 12

[3.2.2 Пользовательские типы данных](#_heading=h.aln3xeyyj877) 12

[3.3 Алгоритмы обработки](#_heading=h.gu5o5j32w9qv) 13

[**4 Результаты работы программы**](#_heading=h.8ix35acj2264) **20**

[**Заключение**](#_heading=h.2jxsxqh) **22**

[**Список использованных источников**](#_heading=h.3j2qqm3) **23**

[**Приложение А   
Исходный код программы**](#_heading=h.ru6t0ap1a5qh) **24**

# **Введение**

Целью курсового проектирования является получение навыков объектно-ориентированного программирования.

Задачи:

1. Познакомиться с методологией DFD.
2. Познакомиться с методологией UML.
3. Написать игру “Space Invaders”.

# 1 Техническое задание

Требуется создать на языке C# игру «Space Invaders», которая должна включать в себя следующие функции:

* Управление кораблем с помощью клавиш A и D, а так же стрелок вправо и влево;
* Выстрел при нажатии клавиши пробел;
* Подсчет очков за уничтожение вражеского объекта;
* Конец игры при соприкосновении с вражеским объектом;
* Отображение игры, включающее корабль, вражеские объекты;

Приведем DFD диаграмму потока данных программы (рисунок 1.1).

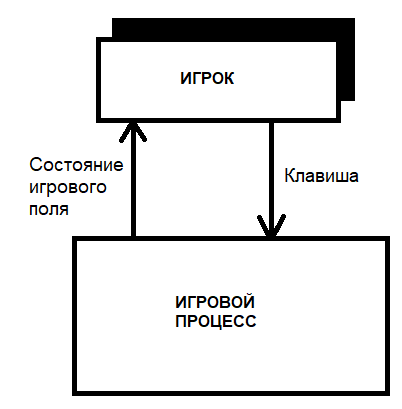


Рисунок 1.1 – Поток входных-выходных данных программы

Пользователь передает программе команды – данные с клавиатуры, программа возвращает визуальные данные.

Чтобы обеспечить защиту от несвоевременного взаимодействия с элементами интерфейса должны быть прописаны условия взаимодействия.

# 2 Обзор средств реализации

## 2.1 Описание среды разработки

Visual Studio 2019 - это стартовая площадка для написания, отладки и сборки кода, а также последующей публикации приложений. Интегрированная среда разработки (IDE) представляет собой многофункциональную программу, которую можно использовать для различных аспектов разработки программного обеспечения. Помимо стандартного редактора и отладчика, которые существуют в большинстве сред IDE, Visual Studio включает в себя компиляторы, средства автозавершения кода, графические конструкторы и многие другие функции для упрощения процесса разработки [1].

Для создания графических интерфейсов с помощью платформы .NET применяются разные технологии - Window Forms, WPF, приложения для магазина Windows Store (для ОС Windows 8/8.1/10). Однако наиболее простой и удобной платформой до сих пор остается Window Forms или формы [2].

Интегрированная среда разработки Visual Studio 2019 была выбрана по следующим причинам:

* Среда поддерживает программную платформу. NET Framework, позволяющую выполнять, отлаживать, компилировать, перестраивать программы, написанные на объектно–ориентированном языке C#.
* Удобный интерфейс и поддержка фирмы–производителя Microsoft. Компания Microsoft является крупнейшей компанией, производящей операционные системы семейства Windows.
* Множество информации о среде разработке, учебных материалов и примеров кода.

## 2.2 Обзор языка программирования и технологии

Для реализации данной программы мной был выбран язык программирования С#.

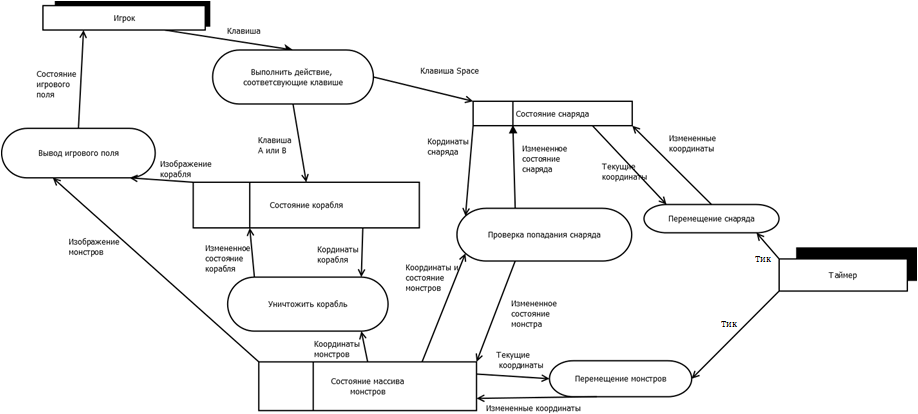
C# - современный объектно-ориентированный и типобезопасный язык программирования. C# позволяет разработчикам создавать множество типов безопасных и надежных приложений, работающих в экосистеме .NET. C# относится к широко известному семейству языков C, и покажется хорошо знакомым любому, кто работал с C, C++, Java или JavaScript. Для более подробного изучения языка C# можно обратиться к учебнику [1].

C# - это объектно- и компонентно-ориентированный язык программирования. C# предоставляет языковые конструкции для непосредственной поддержки такой концепции работы. Благодаря этому C# подходит для создания и применения программных компонентов. С момента создания язык C# обогатился функциями для поддержки новых рабочих нагрузок и современными рекомендациями по разработке ПО.

Элементы управления представляют собой визуальные классы, которые получают введенные пользователем данные и могут инициировать различные события [2].

# 3 Описание проекта

Игра «Space Invaders» позволяет пользователю играть за космический корабль, которому предстоит сразится с полчищем монстров, используя персональный компьютер при помощи клавиатуры. Декомпозицию программы можно увидеть на рисунке 3.1.

Рисунок 3.1 – Декомпозиция игры

На рисунке 3.2 представлена UML-диаграмма входящих в программу классов.

## 3.1 Классы

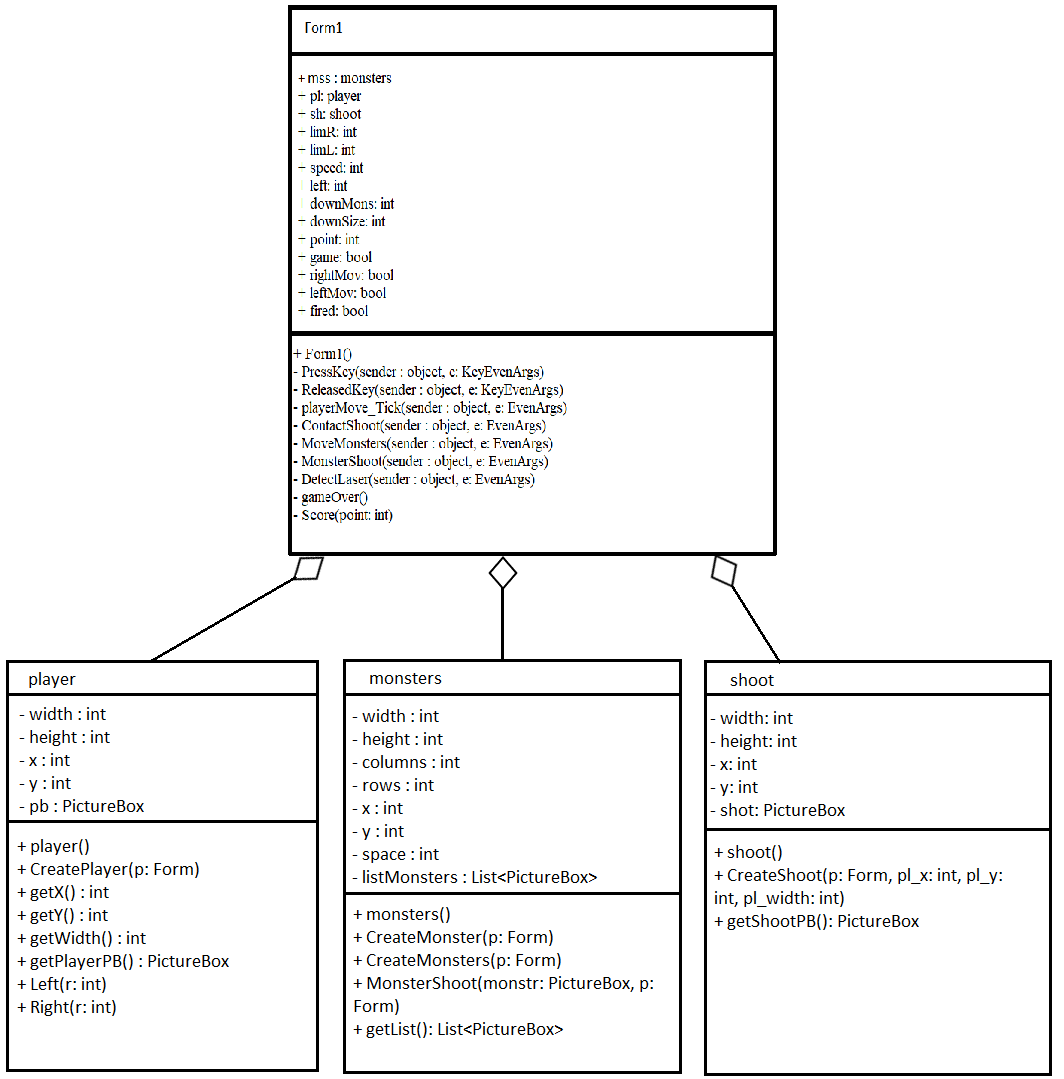


Рисунок 3.2 – Диаграмма классов игры

Для реализации поставленной задачи были созданы следующие классы:

* player – космический корабль игрока, его создание на форме и перемещение.
* monsters – появляющиеся монстры, их создание на форме, создание выстрела от монстров.
* shoot – класс выстрела, его создание на форме.

## 3.2 Структуры данных

### 3.2.1 Библиотечные структуры данных

Так же в проекте использовались структуры данных, приведенные ниже с их описаниями и примерами.

* Список List – строго типизированный список объектов, доступных по индексу. Поддерживает методы для поиска по списку, выполнения сортировки и других операций со списками.

Список используется для хранения всех активных монстров, их удобного передвижения, а так же выстрелов монстров.

Создание нового списка целочисленных значений:

List<PictureBox> listMonster = new List<PictureBox>();

Добавление нового элемента в список: listMonster.Add();

Удаляем монстра из листа: listMonster.Remove(monster);

### 3.2.2 Пользовательские типы данных

Для использования в программе был разработан приведенный ниже пользовательские типы данных – классы.

Класс Form1 содержит 14 полей:

* monsters mss;
* player pl;
* shoot sh;
* const int limR;
* const int limL;
* int speed;
* int left;
* int downMons;
* int downSize;
* int point;
* bool game;
* bool rightMov, leftMov;
* bool fired;

Класс player содержит пять полей:

* private int width, height;
* private int x, y;
* private PictureBox pb;

Класс monsters содержит восемь полей:

* private int width, height;
* private int x, y, space;
* private int columns, rows;
* private List<PictureBox> listMonsters;

Класс MyButton содержит пять полей:

* private int width, height;
* private int x, y;
* private PictureBox shot;

Диаграмма классов приложения представлена на рисунке 3.1.

## 3.3 Алгоритмы обработки

Для использования в программе были разработаны несколько алгоритмов. Приведем ниже описание и блок-схемы некоторых из них.

1. Алгоритм метода PressKey, обрабатывающего нажатие клавиш на клавиатуре, представлен на рисунке 3.3.

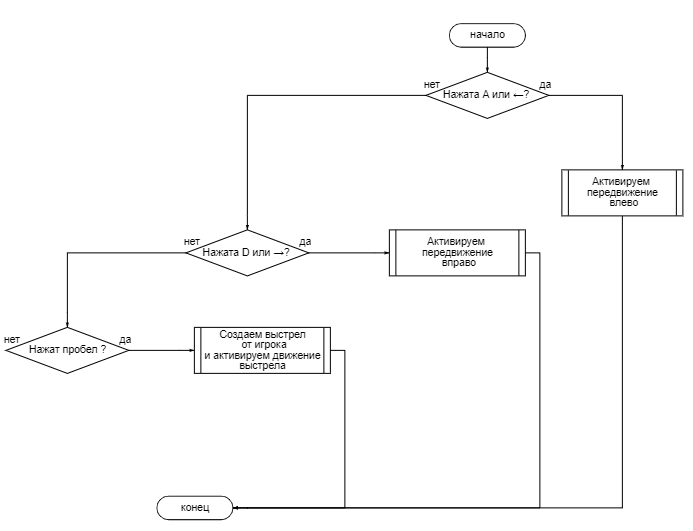


Рисунок 3.3 - Алгоритм метода PressKey

1. Алгоритм метода ReleasedKey, обрабатывающего событие, когда кнопка отпущена, представлен на рисунке 3.4.

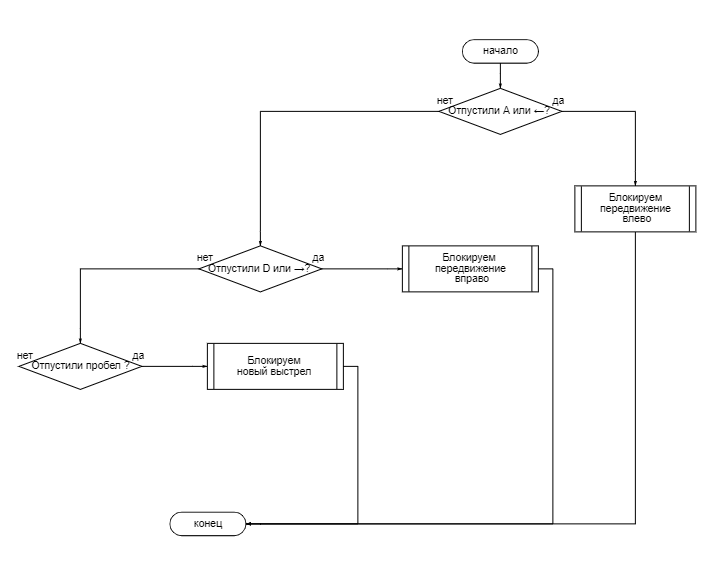


Рисунок 3.4 - Алгоритм метода ReleasedKey



1. Алгоритм метода playerMove\_Tick, обрабатывающего передвижение корабля по таймеру представлен на рисунке 3.5.

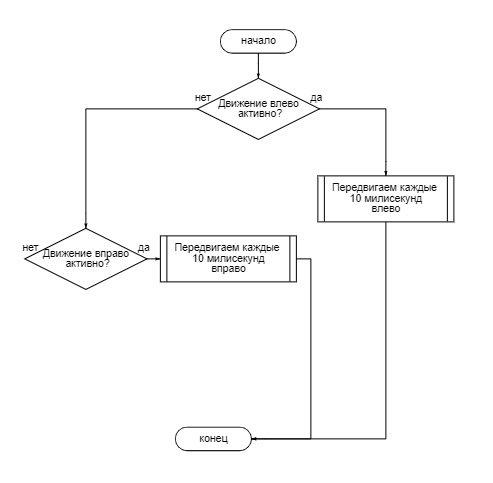


Рисунок 3.5 - Алгоритм метода playerMove\_Tick

1. Алгоритм метода ContactShoot, обрабатывающего соприкосновение выстрела корабля или его вылет за карту, представлен на рисунке 3.6.

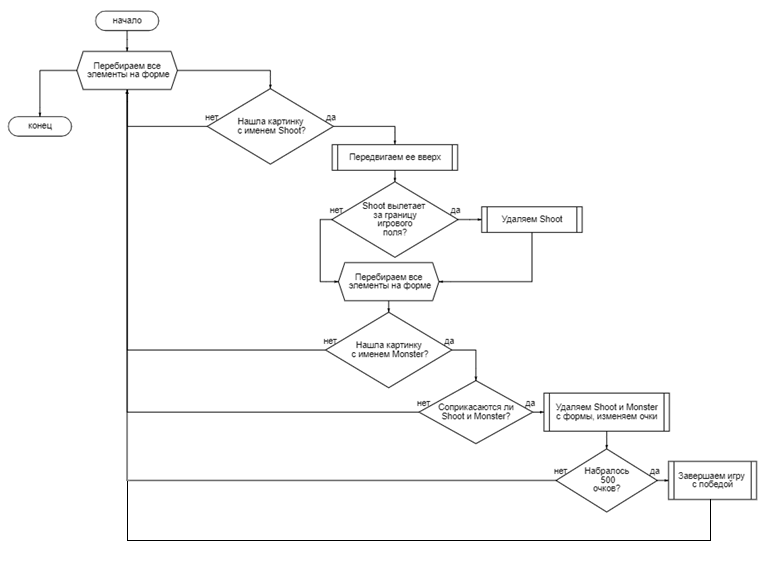


Рисунок 3.6 - Алгоритм метода ContactShoot

5) Алгоритм метода MoveMonsters обрабатывает передвижение монстров при соприкосновением с краем игрового поля, представлен на рисунке 3.7.

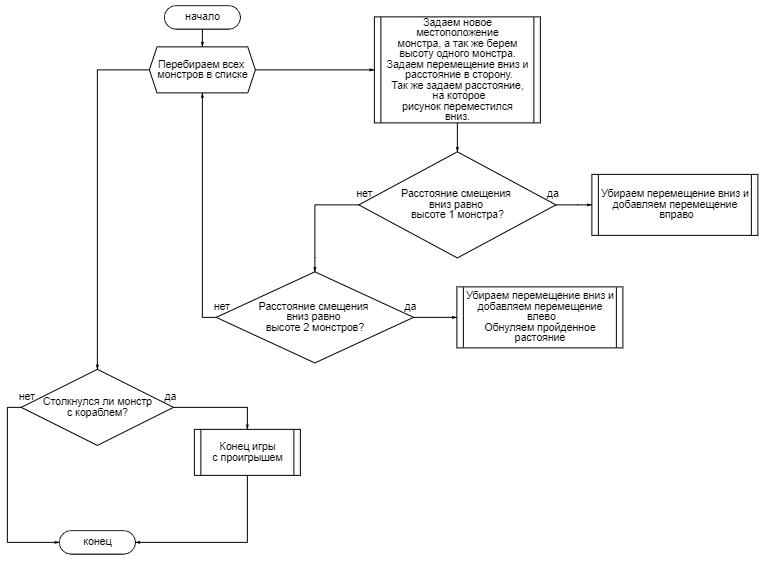


Рисунок 3.7 - Алгоритм метода MoveMonsters

6) Алгоритм метода MonsterShoot создает выстрел от монстра, представлен на рисунке 3.8.

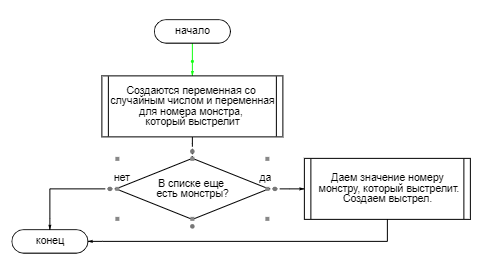


Рисунок 3.8 - Алгоритм метода MonsterShoot

7) Алгоритм метода DetectLaser ищет выстрел игрока, передвигает его и ищет соприкосновение или вылет данного выстрела , представлен на рисунке 3.9.

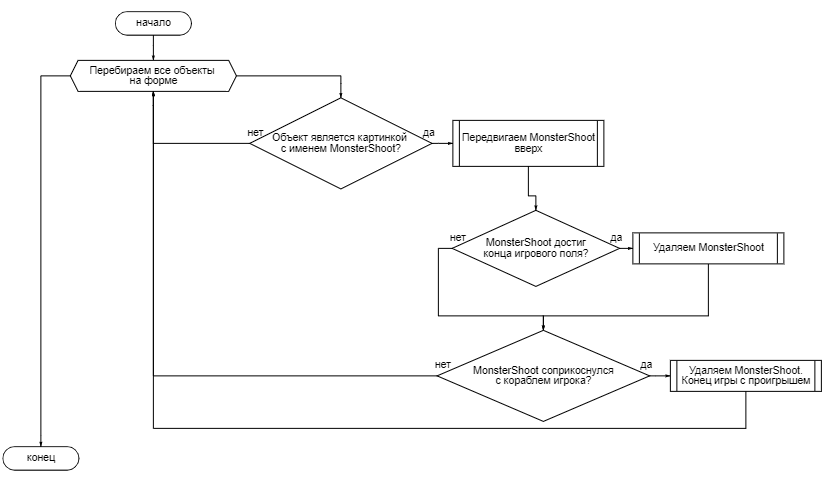


Рисунок 3.9 - Алгоритм метода DetectLaser

8) Метод gameOver заканчивает игру, показывает текстовое сообщение об окончании игры с проигрышем и набранные очки.

9) Метод Score обновляет очки на форме.

# **4 Результаты работы программы**

В ходе выполнения курсовой работы были повторены основы объектно-ориентированного программирования на языке С#:

* Созданы 3 класса;
* Создан лист объектов одного из классов;

Написание программы способствовало закреплению теоретического материала на практике.

В результате разработки была разработана программа со следующим пользовательским интерфейсом: Игровое поле игры “Space Invaders” (рисунок 4.1).



Рисунок 4.1 - Игра “Space Invaders”

# **Заключение**

В результате курсового проектирования были решены следующие задачи:

* Создана игра «Space Invaders»;
* Реализованы функции: передвижение игрока, передвижение монстров, выстрелы игрока и монстров, подсчет очков и конец игровой сессии;
* Спроектирован пользовательский интерфейс;
* Заданы условия своевременного взаимодействия пользователя с элементами интерфейса.

Цель работы достигнута.

# **Список использованных источников**

1. Microsoft. Developer Network. Средства разработчика, техническая документация и примеры кода [электронный ресурс] – Режим доступа: https://docs.microsoft.com
2. Metanit. Руководство по программированию в Windows Forms [электронный ресурс] – Режим доступа: https://metanit.com/sharp/windowsforms/

# **Приложение А Исходный код программы**

**using** System;

**using** System.Collections.Generic;

**using** System.ComponentModel;

**using** System.Data;

**using** System.Drawing;

**using** System.Linq;

**using** System.Text;

**using** System.Threading.Tasks;

**using** System.Windows.Forms;

**namespace** SpaceInvaders

{

**class** shoot *//Класс выстрела корабля*

{

**private** **int** width, height; *//Ширина и высота*

**private** **int** x, y; *//Начальное положение*

PictureBox shot; *//Объект-картинка*

**public** shoot() *//Начальные свойства выстрела*

{

width = 5;

height = 20;

}

**public** **void** CreateShoot(Form p, **int** pl\_x, **int** pl\_y, **int** pl\_width) *//Создать выстрел, относительно корабля*

{

shot = new PictureBox();

x = pl\_x + pl\_width / 2;

y = pl\_y - 20;

shot.Location = new Point(x, y);

shot.Size = new Size(width, height);

shot.BackColor = Color.Red;

shot.Name = "Shoot";

p.Controls.**Add**(shot);

}

**public** PictureBox getShootPB() *//Получить объект-картинку выстрела*

{

**return** shot;

}

}

**class** monsters *//Класс монстров*

{

**private** **int** width, height; *//Высота и ширина*

**private** **int** columns, rows; *//Количество монстров (столбцы и строки)*

**private** **int** x, y, space; *// Положение монстра и расстояние между ними*

**private** List<PictureBox> listMonsters; *//Список всех монстров*

**public** monsters() *//Начальные свойства монстра*

{

width = 40;

height = 40;

columns = 10;

rows = 5;

space = 10;

x = 150;

y = 0;

listMonsters = new List<PictureBox>();

}

**private** **void** CreateMonster(Form p) *//Создаем 1 монстра и ставим его на форму, добавляем в лист*

{

PictureBox pb = new PictureBox();

pb.Location = new Point(x, y);

pb.Size = new Size(width, height);

pb.BackgroundImage = Image.FromFile(@"C:\Users\beloh\Desktop\Space Invaders\Space Invaders\monster.png");

pb.BackgroundImageLayout = ImageLayout.Stretch;

pb.Name = "Monster";

listMonsters.**Add**(pb);

p.Controls.**Add**(pb);

}

**public** **void** CreateMonsters(Form p) *//Создаем много монстров*

{

**for** (**int** i = 0; i < rows; i++)

{

**for**(**int** j = 0; j < columns; j++)

{

CreateMonster(p);

x += width + space;

}

y += height + space;

x = 150;

}

}

**public** **void** MonsterShoot(PictureBox monstr, Form p) *//Создание выстрела от монстра*

{

PictureBox MonsterShoot = new PictureBox();

MonsterShoot.Location = new Point(monstr.Location.X + monstr.Width / 3, monstr.Location.Y + 20);

MonsterShoot.Size = new Size(5, 20);

MonsterShoot.BackColor = Color.White;

MonsterShoot.Name = "MonsterShoot";

p.Controls.**Add**(MonsterShoot);

}

**public** List<PictureBox> getList() *//Получить лист с монстрами*

{

**return** listMonsters;

}

}

**class** player *//Класс для космического корабля*

{

**private** **int** width, height; *//Ширина и высота корабля*

**private** **int** x, y; *//Координаты начала корабля*

**private** PictureBox pb; *// Область картинки корабля*

**public** player() *// Начальные свойства корабля*

{

width = 60;

height = 50;

x = 150;

y = 700;

}

**public** **void** CreatePlayer(Form p) *//Создаем корабль на форме*

{

pb = new PictureBox();

pb.Location = new Point(x, y);

pb.Size = new Size(width, height);

pb.BackgroundImageLayout = ImageLayout.Stretch;

pb.BackgroundImage = Image.FromFile(@"C:\Users\beloh\Desktop\Space Invaders\Space Invaders\SpaceShip.png");

p.Controls.**Add**(pb);

}

**public** **int** getX() *//Получаем положение корабля по х*

{

**return** x;

}

**public** **int** getY() *//Получаем положение корабля по у*

{

**return** y;

}

**public** **int** getWidth() *//Получаем высоту корабля*

{

**return** width;

}

**public** PictureBox getPlayerPB() *//Получаем объект-картинку корабля*

{

**return** pb;

}

**public** **void** Left(**int** r) *//Движение корабля влево на r*

{

**if**(x>=0){

x-=r;

pb.Location = new Point(x,y);

}

}

**public** **void** Right(**int** r) *//Движение корабля вправо на r*

{

**if** (x <= 730)

{

x += r;

pb.Location = new Point(x, y);

}

}

}

**public** **partial** **class** Form1 : Form

{

monsters mss = new monsters();

player pl = new player();

shoot sh = new shoot();

**public** Form1()

{

InitializeComponent();

mss.CreateMonsters(**this**); *//Добавляем монстров на экран*

pl.CreatePlayer(**this**); *//Добавляем игрока на экран*

playerMove.Tick += playerMove\_Tick; *//Добавляем таймер для игрока, чтобы передвижение было плавным*

}

**const** **int** limR = 730; *//Правый край экрана*

**const** **int** limL = 0; *//Левый край экрана*

**int** speed = -1; *//Скорость монстров*

**int** left = -1; *//Передвижение монстров в лево/право*

**int** downMons = 0; *//Перемещение монстров вниз*

**int** downSize = 0; *// Для перемещения монстров вниз, равным размеру 1 монстра*

**int** point = 0; *//Очки*

**bool** game = **true**; *//Игра активна?*

**bool** rightMov, leftMov; *//Нажата ли клавиша право/лево?*

**bool** fired; *//Есть ли выстрел?*

**private** **void** PressKey(**object** sender, KeyEventArgs e) *//Для зажатой клавиши*

{

**if** (e.KeyCode == Keys.A || e.KeyCode == Keys.Left)

{

leftMov = **true**;

}

**else** **if** (e.KeyCode == Keys.D || e.KeyCode == Keys.Right)

{

rightMov = **true**;

}

**else** **if** (e.KeyCode == Keys.Space && game && !fired) *// Создается один выстрел, а не последовательность*

{

new shoot().CreateShoot(**this**, pl.getX() , pl.getY(), pl.getWidth());

fired = **true**;

}

}

**private** **void** ReleasedKey(**object** sender, KeyEventArgs e) *//Когда отпустили клавишу*

{

**if** (e.KeyCode == Keys.A || e.KeyCode == Keys.Left)

{

leftMov = **false**;

}

**else** **if** (e.KeyCode == Keys.D || e.KeyCode == Keys.Right)

{

rightMov = **false**;

}

**else** **if** (e.KeyCode == Keys.Space)

{

fired = **false**;

}

}

**private** **void** playerMove\_Tick(**object** sender, EventArgs e) *//Для плавного движения (timer)*

{

**if** (leftMov) pl.Left(1);

**else** **if** (rightMov) pl.Right(1);

}

**private** **void** ContactShoot(**object** sender, EventArgs e) *//При соприкосновении выстрела с монстром или вылетом за карту (timer)*

{

**foreach** (Control shot **in** **this**.Controls)

{

**if** (shot is PictureBox && shot.Name == "Shoot") *// Для картинки с именем Shoot*

{

PictureBox shoot = (PictureBox)shot;

shoot.Top -= 5;

**if** (shoot.Location.Y <= 0) *//Если вылетает за карту*

{

**this**.Controls.**Remove**(sh.getShootPB()); *//Удалить*

}

**foreach**(Control mnstr **in** **this**.Controls)

{

**if** (mnstr is PictureBox && mnstr.Name == "Monster") *//Для картинок с именем Monster*

{

PictureBox monster = (PictureBox)mnstr;

**if** (shoot.Bounds.IntersectsWith(monster.Bounds)) *//Если прямоугольники монстра и выстрела соприкасаются*

{

**this**.Controls.**Remove**(shoot); *//Удаляем выстрел*

**this**.Controls.**Remove**(monster); *//Удаляем монстра*

mss.getList().**Remove**(monster); *//Удаляем монстра из листа*

point += 10; *//Добавляем очки*

Score(point); *//Отображаем измененные очки*

**if** (point == 500) *//Проверка на победу*

{

game = **false**;

Label win = new Label(); *//В случае победы выставляем соответствующую надпись*

win.AutoSize = **true**;

win.Location = new Point(limR / 2, limR / 2);

win.Text = "Вы победили!**\n**" + "Очки: " + point.ToString();

win.ForeColor = Color.White;

win.Font = new Font(win.Font.FontFamily, 20, FontStyle.Bold);

**this**.Controls.**Add**(win);

}

}

}

}

}

}

}

**private** **void** MoveMonsters(**object** sender, EventArgs e) *//Передвижение монстров (timer)*

{

**foreach** (PictureBox monstr **in** mss.getList()) *//Для любой картинке в листе с Монстрами*

{

monstr.Location = new Point(monstr.Location.X + left, monstr.Location.Y + downMons); *//Новое расположение картинки*

**int** size = monstr.Height; *//Высота прямоугольника(картинки) с монстром*

**if** ((monstr.Location.X <= limL || monstr.Location.X >= limR)&&game) *//Если не выходит за границы и игра продолжается*

{

downMons = 1; *//Спустить картинку на 1, изменяя у*

left = 0; *// Не перемещать в сторону*

downSize++; *//Растояние, на которое переместился рисунок*

**if** (downSize == size) *//Если расстояние равно размеру монстра, то убираем изменения по у, возвращаем скорость перемещения и меняем сторону движения монстров*

{

downMons = 0;

left = speed \* (-1);

}

**else** **if** (downSize == size \* 2) *// При столкновении с другой стороной расстояние станет размеру 2-ух монстров, тогда начинаем движение в другую сторону*

{

downMons = 0;

left = speed;

downSize = 0;

} *//Разделение связано с тем, что left обновляется*

}

**if** (monstr.Bounds.IntersectsWith(pl.getPlayerPB().Bounds)) *//Если прямоугольник-монстр столкнется с каорблем, игра закончится*

{

gameOver();

}

}

}

**private** **void** MonsterShoot(**object** sender, EventArgs e) *// Создание выстрела от монстра через интервал (timer)*

{

Random sh = new Random();

**int** num; *// Номер монстра для выстрела*

**if** (mss.getList().Count > 0) *//Возвращаем количество элементов в листе с монстрами и если монстры есть*

{

num = sh.Next(mss.getList().Count);

new monsters().MonsterShoot(mss.getList()[num], **this**);

}

}

**private** **void** DetectLaser(**object** sender, EventArgs e) *//(timer)*

{

**foreach**(Control shot **in** **this**.Controls)

{

**if** (shot is PictureBox && shot.Name == "MonsterShoot") *//Если есть картинка с именем MonsterShoot*

{

PictureBox MonsterShoot = (PictureBox)shot;

MonsterShoot.Top += 5; *//Передвигается вверх на 5*

**if** (MonsterShoot.Location.Y >= limR) *//Если вылетает за карту, то удаляется*

{

**this**.Controls.**Remove**(MonsterShoot);

}

**if** (MonsterShoot.Bounds.IntersectsWith(pl.getPlayerPB().Bounds)) *//Если соприкосается с прямоугольником игрока, то проигрыш*

{

**this**.Controls.**Remove**(MonsterShoot);

gameOver();

}

}

}

}

**private** **void** gameOver() *//Проигрыш*

{

game = **false**;

Label lose = new Label(); *//В случае поражения выставляем соответствующую надпись*

lose.AutoSize = **true**;

lose.Location = new Point(limR / 2, limR / 2);

lose.Text = "Вы проиграли!**\n**" + "Очки: " + point.ToString();

lose.ForeColor = Color.White;

lose.Font = new Font(lose.Font.FontFamily, 20, FontStyle.Bold);

**this**.Controls.**Add**(lose);

}

**private** **void** Score(**int** point) *//Показываем очки*

{

label2.Text = "Score: " + point.ToString();

}

}

}