



Space Invaders in C# WinForm



NickNs1991

6 Dec 2019 CPOL

Simple Space Invaders

[Download source code - 324 KB](#)

Introduction

Simple Space Invaders game made in C# WinForm. The sprites are being taken from:

- <https://www.mooict.com/wp-content/uploads/2017/03/Spaceinvaders-resources-mooict.zip>

How the Game Works

- Player moves left using the **A** and **Left arrow** buttons, or right using the **D** and **Right arrow** buttons or space to fire toward aliens.
- Aliens are moving from right to left and vice-versa lowering down when they hit the edge, firing at player while they are moving.
- Player wins if he eliminate all the aliens before they reach him and loses if he gets hit 3 times by a laser, or gets in collision with some of aliens meaning that he failed to kill them in time.

Using the Code

First, we're going to add a **Player** moving controls. In the Form Event grid, I used **KeyDown** and **KeyUp** events called **Pressed** and **Released**. There are also 3 global boolean values called **moveRight** and **moveLeft** which are going to be set to **true** depending on which buttons the player pressed:

```
private void Pressed(object sender, KeyEventArgs e)
{
    if (e.KeyCode == Keys.A || e.KeyCode == Keys.Left)
    {
        moveLeft = true;
    }
    else if (e.KeyCode == Keys.D || e.KeyCode == Keys.Right)
    {
        moveRight = true;
    }
    else if (e.KeyCode == Keys.Space && game && !fired)
    {
        Missile();
    }
}
```

```

        fired = true;
    }
}
private void Released(object sender, EventArgs e)
{
    if (e.KeyCode == Keys.A || e.KeyCode == Keys.Left)
    {
        moveLeft = false;
    }
    else if (e.KeyCode == Keys.D || e.KeyCode == Keys.Right)
    {
        moveRight = false;
    }
    else if (e.KeyCode == Keys.Space)
    {
        fired = false;
    }
}
private void PlayerMove(object sender, EventArgs e)
{
    if (moveLeft && Player.Location.X >= 0)
    {
        Player.Left--;
    }
    else if (moveRight && Player.Location.X <= limit)
    {
        Player.Left++;
    }
}
}

```

If the player pressed space to fire, the value **fired** is going to get set to **true** to prevent continuous firing when the button is pressed, and there is additional check if the value **game** is set to **true** checking if the game is still active. If everything is ok, the procedure which creates bullet sprite called **Missile()** is called:

```

private void Missile()
{
    PictureBox bullet = new PictureBox();
    bullet.Location = new Point(Player.Location.X + Player.Width / 2, Player.Location.Y - 20);
    bullet.Size = new Size(5, 20);
    bullet.BackgroundImage = Properties.Resources.bullet;
    bullet.BackgroundImageLayout = ImageLayout.Stretch;
    bullet.Name = "Bullet";
    this.Controls.Add(bullet);
}

```

Now let's add some aliens to the form. For the purpose of better readability, I created a small class called **Enemies** which sets alien sprite parameters (size, images, quantity):

- **Width** and **height** presents the size of the alien sprite (40)
- **Rows** and **columns** are the total number of aliens in order 5x10
- **X** and **Y** presents the starting coordinates of the sprites in form, and the **space** is the distance between them

```

class Enemies
{
    private int width, height;
    private int columns, rows;
    private int x, y, space;

    public Enemies()
    {
        width = 40;
        height = 40;
    }
}

```

```
        columns = 10;
        rows = 5;
        space = 10;
        x = 150;
        y = 0;
    }
    private void CreateControl(Form p)
    {
        PictureBox pb = new PictureBox();
        pb.Location = new Point(x, y);
        pb.Size = new Size(width, height);
        pb.BackgroundImage = Properties.Resources.invader;
        pb.BackgroundImageLayout = ImageLayout.Stretch;
        pb.Name = "Alien";
        p.Controls.Add(pb);
    }
    public void CreateSprites(Form p)
    {
        for(int i = 0; i < rows; i++)
        {
            for(int j = 0; j < columns; j++)
            {
                CreateControl(p);
                x += width + space;
            }
            y += height + space;
            x = 150;
        }
    }
}
```

To add them once the program is started, we need to create a class within the **Form** constructor:

```
public Form1()
{
    InitializeComponent();
    new Enemies().CreateSprites(this);
    InsertAliens();
}
```

And when this is done and you start the program, it should appear like this:

Now we're moving to the alien movement part. But before that, I'm going to isolate all the sprites (`pictureBox-es`) named "`Alien`" in one public list called `aliens`:

```
private void InsertAliens()
{
    foreach(Control c in this.Controls)
    {
        if (c is PictureBox && c.Name == "Alien")
        {
            PictureBox alien = (PictureBox)c;
            aliens.Add(alien);
        }
    }
}
```

And we can go further now. As the aliens are moving from right to left and vice versa, when they touch one of the edges of the screen, they will go down and change direction so I had to create a certain logic for that purpose. First, I'm going to make a boolean function named **Touched** to check if the aliens touched the edge of the screen:

```
private bool Touched(PictureBox a)
{
    return a.Location.X <= 0 || a.Location.X >= limit;
}
```

And then the **SetDirection** procedure. The values I'm using here are **top** and **left** for the direction of the aliens, **cnt** that counts how low aliens went, and **speed** which is going to switch the direction of the aliens at a certain point.

First, it will check if one of the aliens touched the edge of the screen, and if it did, the variable **cnt** will increment. When the **cnt** reaches the height size of a single alien, it will stop going down and change direction. The same thing will happen when the **cnt** reaches double size of the previous value meaning that the aliens collided with the edge from the other side and the direction will be changed again. After that, the **cnt** value will be set to **0** in order to do the same procedure in the next row:

```
private void SetDirection(PictureBox a)
{
    int size = a.Height;

    if (Touched(a))
    {
        top = 1; left = 0; cnt++;

        if (cnt == size)
        {
            top = 0; left = speed * (-1); Observer.Start();
        }
        else if (cnt == size * 2)
        {
            top = 0; left = speed; cnt = 0; Observer.Start();
        }
    }
}
```

The function of the timer called **Observer** will be explained later.

Now I will post the code for procedure called **AlienMoves** which loops through the list of aliens (called **aliens**) and moves the sprites by the coordinates that are being set in the **SetDirection** procedure. The **AlienMoves** procedure also contains another procedure which checks if the aliens collided with player in which case the game is over:

```
private void AlienMove()
{
    foreach(PictureBox alien in aliens)
    {
        alien.Location = new Point(alien.Location.X + left, alien.Location.Y + top);
        SetDirection(alien);
        Collided(alien);
    }
}
```

Collided procedure:

```
private void Collided(PictureBox a)
{
    if (a.Bounds.Intersects(Player.Bounds))
    {
        gameOver();
    }
}
```

And the timer **MoveAliens** that calls the **AlienMoves** procedure to run things:

```
private void MoveAliens(object sender, EventArgs e)
{
    AlienMove();
}
```

Now we arrived at the part of the aliens firing towards player. Just like in case of player, first we need to write a procedure which creates a sprite of the laser. It's called **Beam**:

```
private void Beam(PictureBox a)
{
    PictureBox laser = new PictureBox();
    laser.Location = new Point(a.Location.X + a.Width / 3, a.Location.Y + 20);
    laser.Size = new Size(5, 20);
    laser.BackgroundImage = Properties.Resources.laser;
    laser.BackgroundImageLayout = ImageLayout.Stretch;
    laser.Name = "Laser";
    this.Controls.Add(laser);
}
```

Before I post the code, I will explain how the firing works. We have two timers where one presents the time span at which the lasers are going to be fired called **StrikeSpan** and its time interval is set to 1000 (1s) meaning that after each second, a laser is going to be fired. The other one called **DetectLaser** finds the laser that is created and hurls it down the ground by setting its **Top** property to 5. That timer is set to interval of 1ms.

StrikeSpan:

```
private void StrikeSpan(object sender, EventArgs e)
{
    Random r = new Random();
    int pick;

    if (aliens.Count > 0)
    {
        pick = r.Next(aliens.Count);
        Beam(aliens[pick]);
    }
}
```

DetectLaser:

```
private void DetectLaser(object sender, EventArgs e)
{
    foreach (Control c in this.Controls)
    {
        if (c is PictureBox && c.Name == "Laser")
        {
            PictureBox laser = (PictureBox)c;
            laser.Top += 5;

            if (laser.Location.Y >= limit)
            {
                this.Controls.Remove(laser);
            }
            if (laser.Bounds.Intersects(Player.Bounds))
            {
                this.Controls.Remove(laser);
                LoseLife();
            }
        }
    }
}
```

```

    }
}
}

```

As you can see, the code checks if the laser left the field in which case it's going to be removed, or if laser collided with player when the player loses life.

That was a part of the aliens striking. Now it's time for the player.

We have a single timer called **FireBullet**, but things are a bit more complicated as we have to perform multiple different checks. I'll separate the one important part:

```

if (bullet.Bounds.Intersects(alien.Bounds) && !Touched(alien))
{
    this.Controls.Remove(bullet);
    this.Controls.Remove(alien);
    aliens.Remove(alien);
    pts += 5;
    Score(pts);
    CheckForWinner();
}
else if (bullet.Bounds.Intersects(alien.Bounds) && Touched(alien))
{
    this.Controls.Remove(bullet);
    this.Controls.Remove(alien);
    delay.Add(alien);
    pts += 5;
    Score(pts);
    CheckForWinner();
}

```

While testing the code, I figured out that when the aliens touch the edge of the screen and I destroy them, the condition from procedure **SetDirection** returns **false** and the procedure fails to switch direction properly, as there are no more aliens touching the edge, so they would just keep moving down which presents the logical issue. I solved that problem by creating an additional global list called **delay** and a timer called **Observer**, so when I destroy aliens when they touched the edge, the bullet and alien controls are being removed, and instead of deleting the picture from the list **aliens** (from which they are moving on the screen), I add all destroyed pictures to **delay** list first so they still exist in **aliens** list to be able to switch direction. Once they switched direction, I call **Observer** timer which then removes all the aliens from the **aliens** list and clear the **delay** list. So when they reach the edge of the same side I destroyed the aliens from before, there won't be an empty space between aliens and the screen, and the sprites will change the direction correctly. Timer also checks if the bullet left the screen in which case it gets removed, or if it collided with some of the lasers. The full code is as given below:

```

private void FireBullet(object sender, EventArgs e)
{
    foreach (Control c in this.Controls)
    {
        if (c is PictureBox && c.Name == "Bullet")
        {
            PictureBox bullet = (PictureBox)c;
            bullet.Top -= 5;

            if (bullet.Location.Y <= 0)
            {
                this.Controls.Remove(bullet);
            }
            foreach (Control ct in this.Controls)
            {
                if (ct is PictureBox && ct.Name == "Laser")
                {
                    PictureBox laser = (PictureBox)ct;

                    if (bullet.Bounds.Intersects(laser.Bounds))

```

```
        {
            this.Controls.Remove(bullet);
            this.Controls.Remove(laser);
            pts++;
            Score(pts);
        }
    }
}
foreach(Control ctrl in this.Controls)
{
    if (ctrl is PictureBox && ctrl.Name == "Alien")
    {
        PictureBox alien = (PictureBox)ctrl;

        if (bullet.Bounds.Intersects(alien.Bounds) && !Touched(alien))
        {
            this.Controls.Remove(bullet);
            this.Controls.Remove(alien);
            aliens.Remove(alien);
            pts += 5;
            Score(pts);
            CheckForWinner();
        }
        else if (bullet.Bounds.Intersects(alien.Bounds) && Touched(alien))
        {
            this.Controls.Remove(bullet);
            this.Controls.Remove(alien);
            delay.Add(alien);
            pts += 5;
            Score(pts);
            CheckForWinner();
        }
    }
}
}
}
```

Observer timer:

```
private void Observe(object sender, EventArgs e)
{
    Observer.Stop();

    foreach (PictureBox delayed in delay)
    {
        aliens.Remove(delayed);
    }
    delay.Clear();
}
```

And for the end, I left procedures which check whether the player has won, lost life, if the game is over and add a score in certain situations.

If the bullet collides with a laser, the player gets 1 point, and if the alien is destroyed, he gets 5. To write the result:

```
private void Score(int pts)
{
    label2.Text = "Score: " + pts.ToString();
}
```

When the player gets hit by a laser, he loses life, and at that point, the small picture of tank in the lower left screen gets removed and the player is centered at his starting position in the form. It also checks if the game is over:


```
private void LoseLife()
{
    Player.Location = new Point(x, y);

    foreach(Control c in this.Controls)
    {
        if (c is PictureBox && c.Name.Contains("Life") && c.Visible == true)
        {
            PictureBox player = (PictureBox)c;
            player.Visible = false;
            return;
        }
    }
    gameOver();
}
```

And if the game is over, we stop all of the timers, loop through the form finding the label with the name "Finished". Once it's been found, we're writing the text "Game Over" and all the other controls visibility is set to false:

```
private void gameOver()
{
    timer1.Stop(); timer2.Stop(); timer3.Stop(); timer4.Stop(); timer5.Stop(); Observer.Stop();

    foreach (Control c in this.Controls)
    {
        if (c is Label && c.Name == "Finish")
        {
            Label lbl = (Label)c;
            lbl.Text = "Game Over!";
            game = false;
        }
        else
        {
            c.Visible = false;
        }
    }
}
```

Finding a winner contains two small procedures. The first one called `CheckForWinner()` counts the number of the `pictureBox`-es with the name "Alien". if the count is 0, it calls `YouWon()` procedures that again finds the control named "Finish" writing the text "You Won" and presenting the score player achieved. It also sets `game` value to false preventing the player from creating sprites of the bullet when the space is hit:

```
private void CheckForWinner()
{
    int count = 0;

    foreach(Control c in this.Controls)
    {
        if (c is PictureBox && c.Name == "Alien") count++;
    }

    if (count == 0) YouWon();
}
```

`YouWon` procedure:

```
private void YouWon()
{
    game = false;
}
```

```
foreach(Control c in this.Controls)
{
    if (c is Label && c.Name == "Finish")
    {
        Label lbl = (Label)c;
        lbl.Text = "You Won!" + "\n"
                + "Score: " + pts.ToString();
    }
    else
    {
        c.Visible = false;
    }
}
}
```

Points of Interest

Next time, there could be a lot of potential updates to the game.

History

- 6th December, 2019: Initial version

License

This article, along with any associated source code and files, is licensed under [The Code Project Open License \(CPOL\)](#)

About the Author




NickNs1991

Employed (other) G4S

Serbia 

My name is Aleksandar, I'm 27. I'm currently working in G4S secure solutions. I went to college, studied IT but quit after short time. Still my love for programming remained. Now I'm writing small-trivial projects in C# and posting them as a Tips.

Comments and Discussions

 **8 messages** have been posted for this article Visit <https://www.codeproject.com/Articles/5252990/Space-Invaders-in-Csharp-WinForm> to post and view comments on this article, or click [here](#) to get a print view with messages.

- [Permalink](#)
- [Advertise](#)
- [Privacy](#)
- [Cookies](#)
- [Terms of Use](#)

Article Copyright 2019 by NickNs1991
Everything else Copyright © [CodeProject](#),
1999-2020

Web05 2.8.200414.1