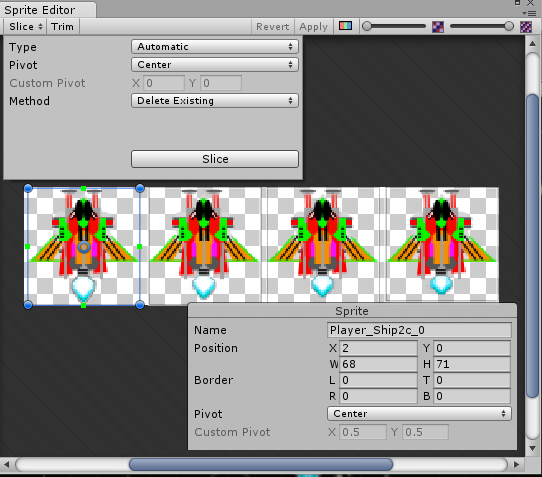
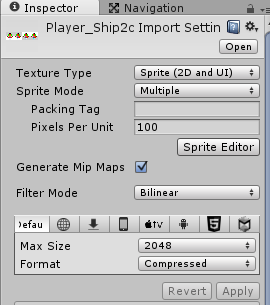
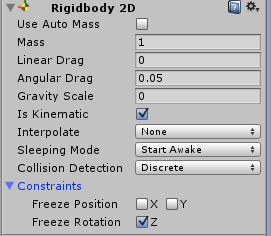
Space Invadersesque 2D shooter

So, we’re going to create another classic game here, one of space invaders, this assumes some basic 2D knowledge and is one in a beginning 2D game series of shorts. All in all, it should take a beginner no more than a few hours.

Start by creating a new 2D project and importing the assets from the site. Drag onto the scene the background image (Space texture curtesy of np4gamer, all images available on opengameart).

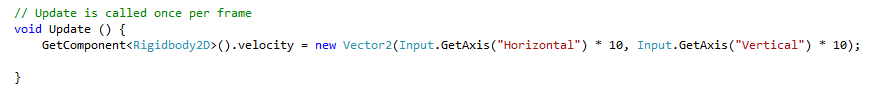
Now, find the ship, click on it in project view, this is a sprite sheet so we need to cut it up into several images; click and change sprite Mode to multiple, then, click sprite editor, slice and ensure that automatic is set as in the screenshot then hit slice to slice the ship using auto slice.

If we select all the images that have now been created (you may need to press the little arrow to expand) by ctrl clicking each, then drag them as one onto the unity hierarchy Unity should prompt you to save. It’s creating a simple animation that just flicks between them, giving us a nice simple little effect, save it and continue.

You’ll now have a gameObject on the hierarchy that’s your player ship, add a Rigidbody2D component and tick the is kinematic and freeze Z constraint. A kinematic object will not be affected by collisions, forces (like gravity) or any other bit of physics, they are “non physical” objects. That’s fine for our simple game and simple ship, we can directly influence the position of the ship using controls.

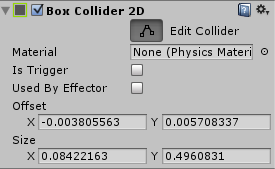
Helpfully, whilst colliders don’t work, triggers do. We need something to check if we’ve been hit so we’ll use triggers to check for them. Add a BoxCollider2D component and tick, “isTrigger”.

We now create a new C# script, shipControl that we can use to, surprise surprise, control the ship, fill the following in.

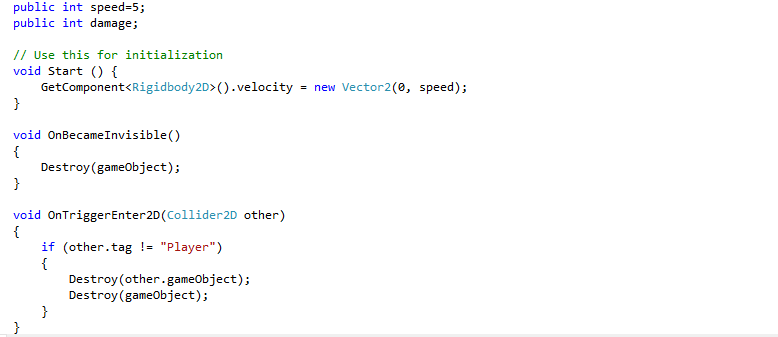
It’s a lazy all in one line solution, but effective- essentially we’re getting the velocity component of the rigidbody (this holds speed and direction of travel) then setting it to a new Vector 2 representing the value of the button pressed for the in built horizontal or vertical axis (arrow keys by default). We times it by ten to get a faster movement, you might want to vary this.

Test the game. Testing is good.

Now for some pew pew, find and slice the beams image in the assets, in a way similar to above. This one we’re not going to use as an animation, instead, choose the laser you want. I went with beams\_51 for the enemy and beams\_43 for the player’s beams. Once you’ve chosen, drag the image (singular) you want onto the scene and then into a folder in your project called “Prefabs”. You don’t strictly need the folder but it helps- everything you do now to the objects in this folder will be applied to all others. You can see if they are prefabs, their name will be blue in the hierarchy. The Laser pack is by Rawdanitsu.

Add a BoxCollider2D to the laser beam, clicking edit collider to adjust it to be smaller better fitting to the laser shape and click IsTrigger. Now add a Rigidbody2D and once again make kinematic.

# Player beams

Create a new script in C# called shotScript, setup as below.

Let’s go through this, the first two lines setup global variables, these are values that are available across the whole script. In Start (remember, this is run whenever the script is first loaded) we set the velocity of the beam to 0 in the horizontal direction and “speed” in the vertical. This lets us easily change the speed of the beam later.

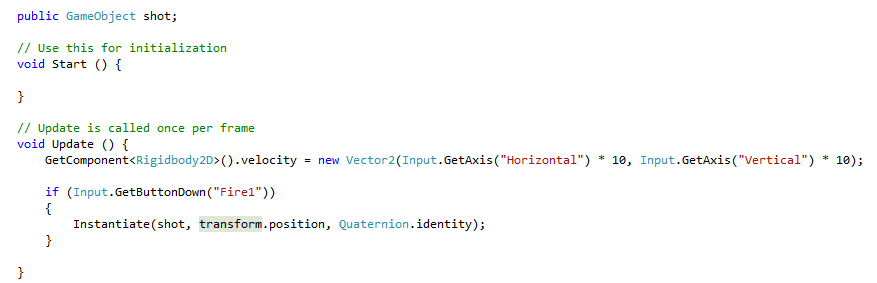
voidOnBecameInvisible() is a built in Unity function, it triggers whenever the player is no longer visible by the Camera (technically when it’s no longer being rendered, there are occassions when this is isn’t the same). When it’s no longer visible, we destroy it (requires it to have been visible at one point).

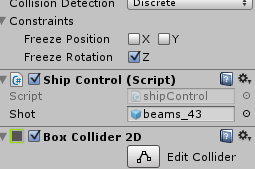
OnTriggerEnter2D(Collider2D other) is another built in, it triggers whenever a collider enters a trigger. A message is sent to both the object with the trigger and the one triggering, a trigger is only triggered is one of the colliders also have a rigidbody attached to it. Here, we check if we’ve hit a player, if we haven’t, we destroy the object we hit and the beam, no mercy here.

This script will be used for the player’s beams, in order to recognise the player (so we don’t accidentally hit ourselves) we use something called a tag, which is just a name attached to the object. We need to find the player object and, after clicking on it, tag it as Player by clicking the drop down next to Tag (Player is a pre existing tag).

Attach your new beam script to the player beam prefab you created earlier. Drag the beam prefab onto the scene and test, they should fly forwards automatically now.

# Shooting back

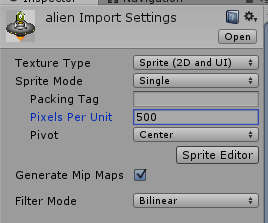
Open shipControl script and create new variable “shot” which will be used to hold the GameObject that represents the beams. Below the GetComponent line from previous and in Update() we add a condition, “if the fire1 button is pressed, create a beam, at this objects position and straight on”.

Drag the prefab you made for the beam onto the script in the inspector, assigning it’s value to the variable shot above.

Test once again, you should be able to shoot beams now.

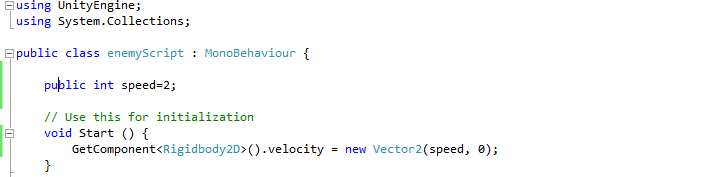
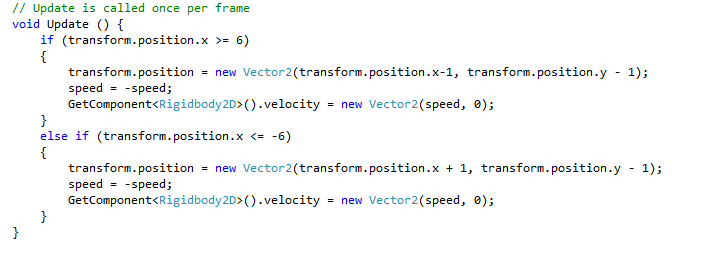
# Enemies

We’re using another free art asset, an alien by Ctoy- <http://c-toy.blogspot.pt/> configure it using the settings below. This ones not a sprite sheet, so just a single, but the pixels per unit (how many pixels in the image refer to one unit in Unity) does need setting (it’ll be too big otherwise).



Drag alien onto the scene, then from the scene into the prefab folder. To the prefab, as before, add a Rigidbody2d, make kinematic, add a BoxCollider2D, make Trigger and test if you can shoot and kill the enemy.

Add a new script enemyScript, this controls your enemies, “public int speed=2;” is a variable to holw the speed of the enemy.



The by now familiar velocity code in start, then the update code which may be slightly confusing.

The “if” bit, checks the objects x position, if it’s greater than or equal to 6, the position is equal to it’s current position -1 and reverse it’s speed before passing this to velocity. A similar thing happens if the position is less than -6. This means if it’s partly off the camera on either side, we reverse it’s speed, or make it go the same speed, in the other direction. We also bring it down a line.

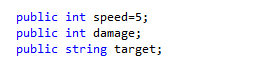
Drag a prefab onto the scene, position it at -5.5,5 then duplicate them (right click it, duplicate), move this one to -3.5, 5, then again and -1.5,5 repeating up to 4.5,5.

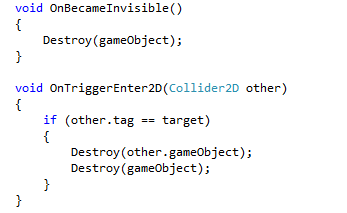
Highlight these and duplicate again, drag them down and find their speed in EnemyScript attached to these ones and set it to -2 for this entire row. Now, with the first one starting at -4.5,5 and going up to 5.5,4 (x going up by 2 each time, y staying the same) position them evenly. Test.

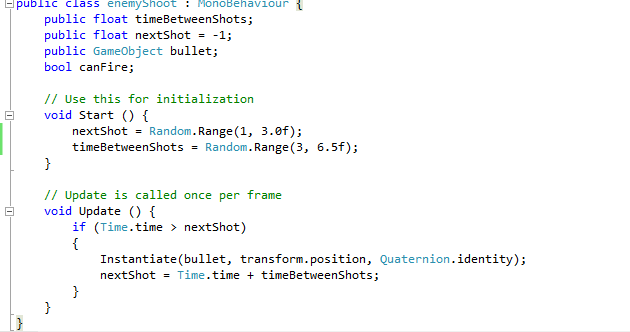
# Enemies shooting

Create new script, enemyShoot

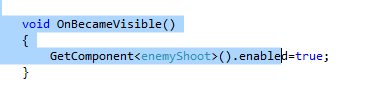
Modify shotscript for target







Add to enemy script



Keep duplicating off screen, waves of two, each moved up by 2

Can make it harder by increasing the speed of the ones above

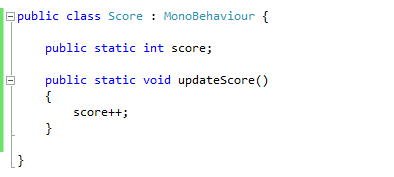
Keep score

Add UI -> Text

Double click canvas

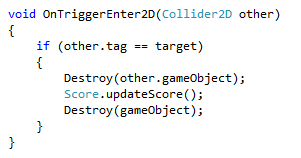
Click text, best fit and change to white

Position bottom left

Change text to Score:

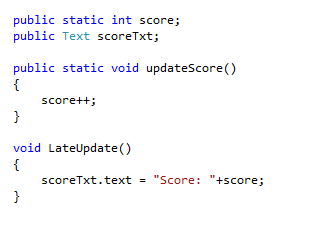
New script- score

Update shotscript



Back to score

Using unityengine.ui



Drag text from UI to field

Test