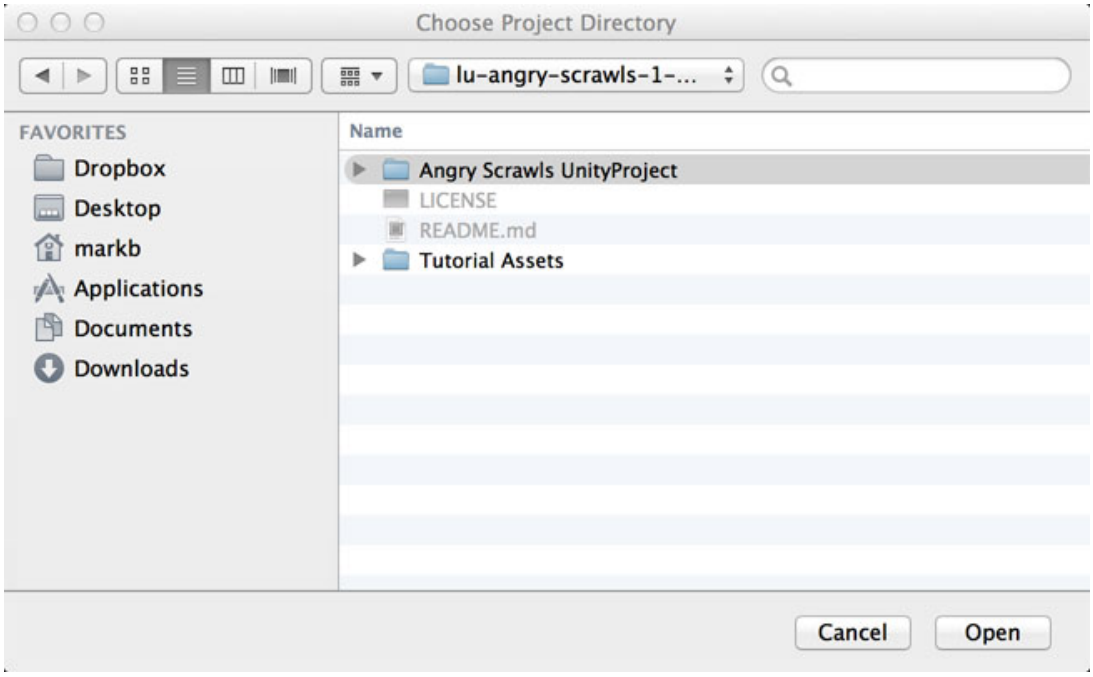
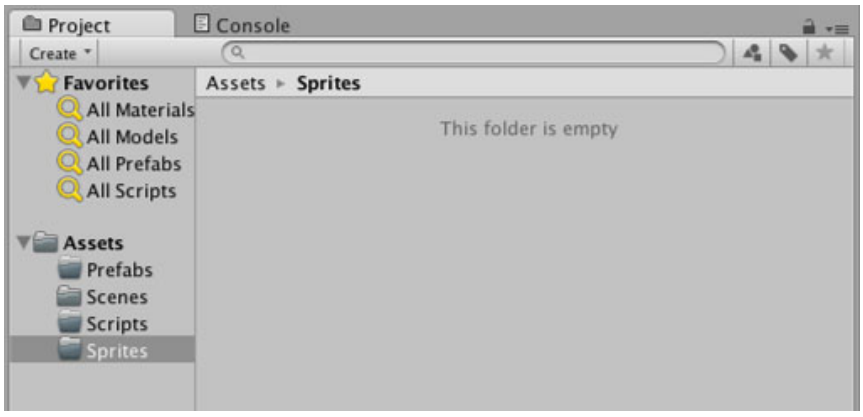
Download the project from https://github.com/mindcandy/lu-angry-scrawls-1 - you can either clone using git, or simply download a zip at https://github.com/mindcandy/lu-angry-scrawls-1/archive/master.zip

Open Unity, then open the “Angry Scrawls UnityProject” folder.



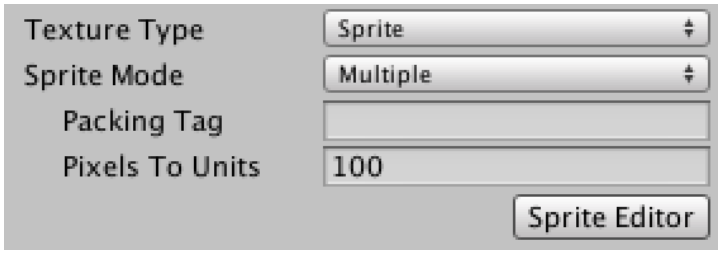
Open the “GameScene” scene in the Assets/Scenes folder in the Project view by double clicking on it.

Now in Unity, select the Assets/Sprites folder and in OS X Finder (or Windows Explorer) open the “Tutorial Assets/Sprites” folder.

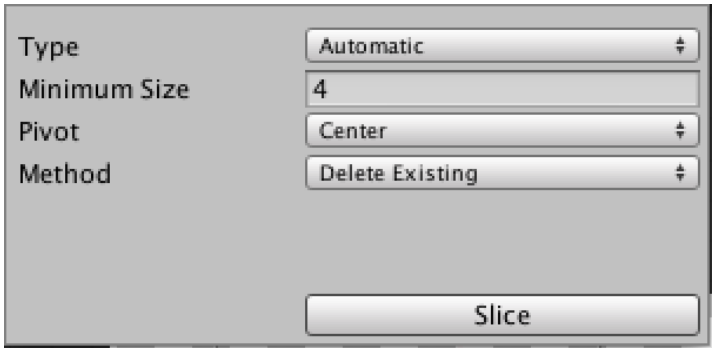


Drag the png files in to Unity and they will be imported - the files are copied into the folder within the Unity project, and some metadata is created by Unity.

By default Unity imports each png file as a single Sprite, but angryscrawls.png has many sprites. So to set them up in Unity, select ‘angryscrawls’ and then in the Inspector set Sprite Mode to Multiple, and click ‘Sprite Editor’.

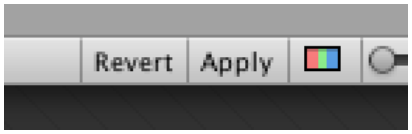


In the new window that pops up, click Slice in the top left and then click Slice again.



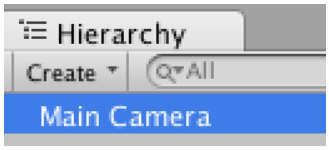
You should see that boxes now surround all the sprites - Unity has used the transparency in the PNG to detect each sprite.

To save this, just click Apply in the top right of the sprite editor. Then click Apply in the Inspector window.



If you click on the black arrow on the right of the angryscrawls Sprite in the Project window you can see the individual sprites.

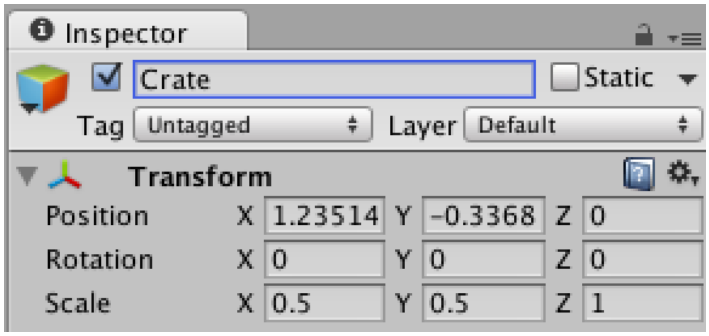
Select Main Camera in the Project hierarchy and zoom out in the Scene view - you can use the mouse wheel or two-finger scroll on a Mac touchpad.



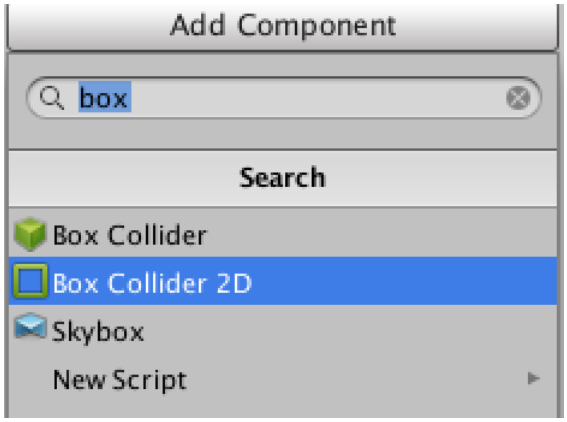
From the Hierarchy, drag the ‘background’ sprite to the Scene view. Unity will create a new Game Object and Sprite Render, so you will be able to see the background in the Scene and Game views. Move and scale the background using the manipulation widgets in the top left of Unity - you can either click on each widget or use the Q, W, E and R shortcut keys.

In the Inspector, set z=10

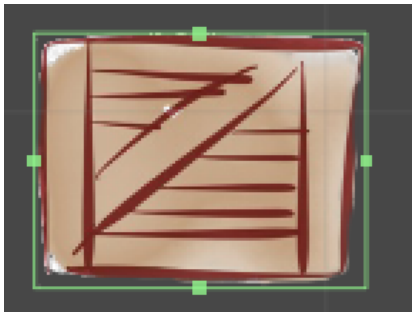
Drag in the ‘crate’ sprite from the Sprites folder, then set the X and Y Scale to 0.5 and rename it to Crate.



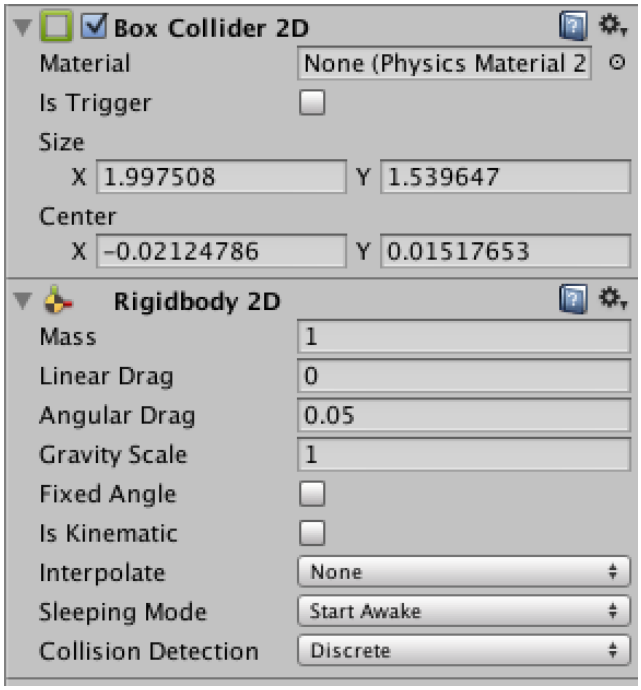
Click on the ‘Add Component’ button in the inspector and add ‘Box Collider 2D’ - the fastest way is to type ‘box’ into the search window.



The Box Collider 2D describes to Unity’s physics engine the shape of the crate. You will be able to see it drawn in a thin green line around the sprite. If it doesn’t quite match up, you can adjust it by holding Shift and then dragging the green control boxes. I recommend doing this in ‘Hand’ mode (Q shortcut key) as then if you miss the (tiny and hard to see) control box, then you don’t accidentally move or scale things.

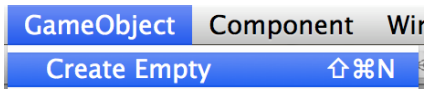


Now click Add Component again and add a ‘Rigidbody 2D’.



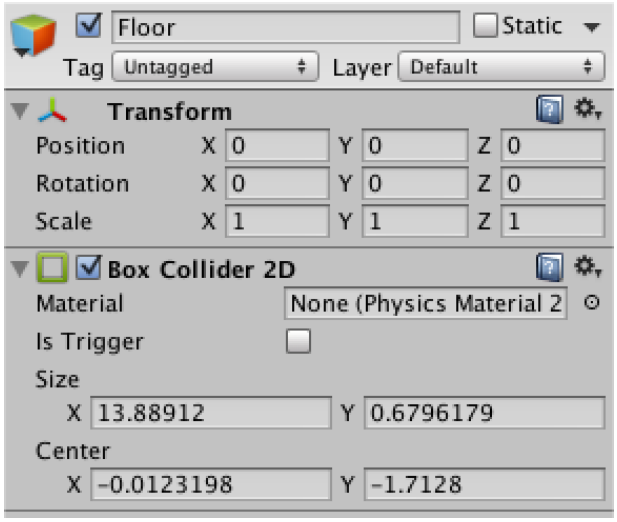
Save the scene (File->Save or Cmd-S) and then press the large Play button in the middle and top of Unity's editor window. You should see the box falls down under gravity and passes through the floor!

So, we need to create a floor. To do this, choose 'Create Empty' from the 'Game Object' menu.

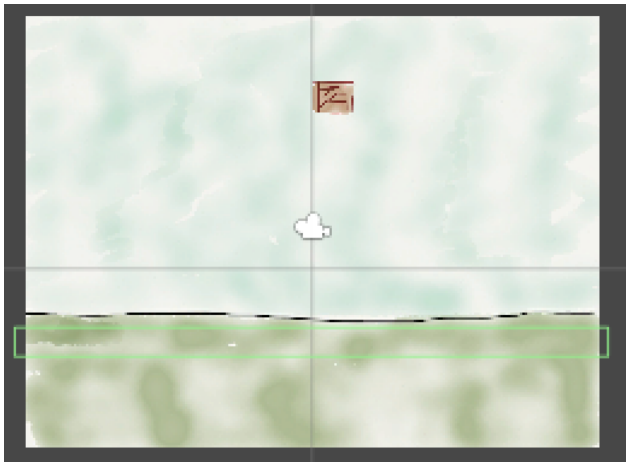


In inspector window, change the name from GameObject to Floor and set the Position to (0,0,0).

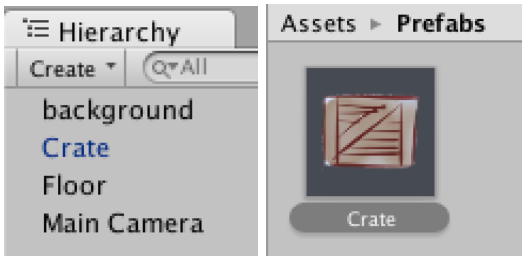
Add a Box Collider 2D component to the Floor, then scale it so that the green rectangle matches where the floor is visually.



Now Save the scene and Run again - you should see the crate won't pass through the floor any more.

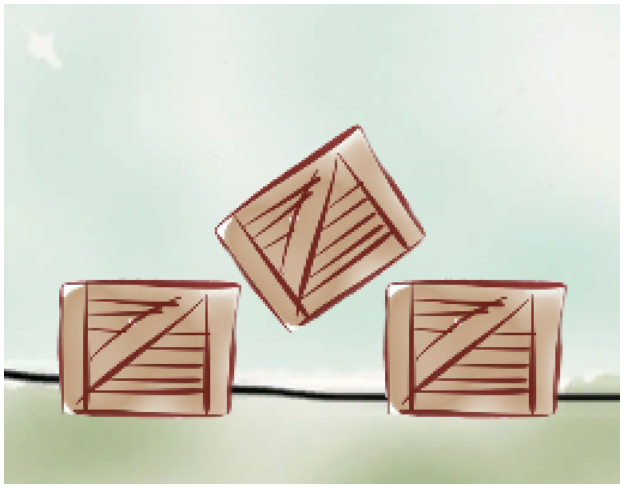


Change the folder in the Project view to Assets/Prefabs, then drag the Crate from the Hierarchy to the Prefabs folder. You will see the Crate prefab is then created and the Crate in the Hierarchy window will turn blue to show that it is now a Prefab.

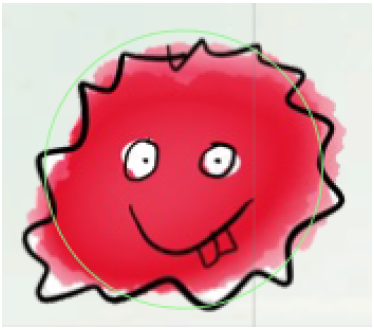


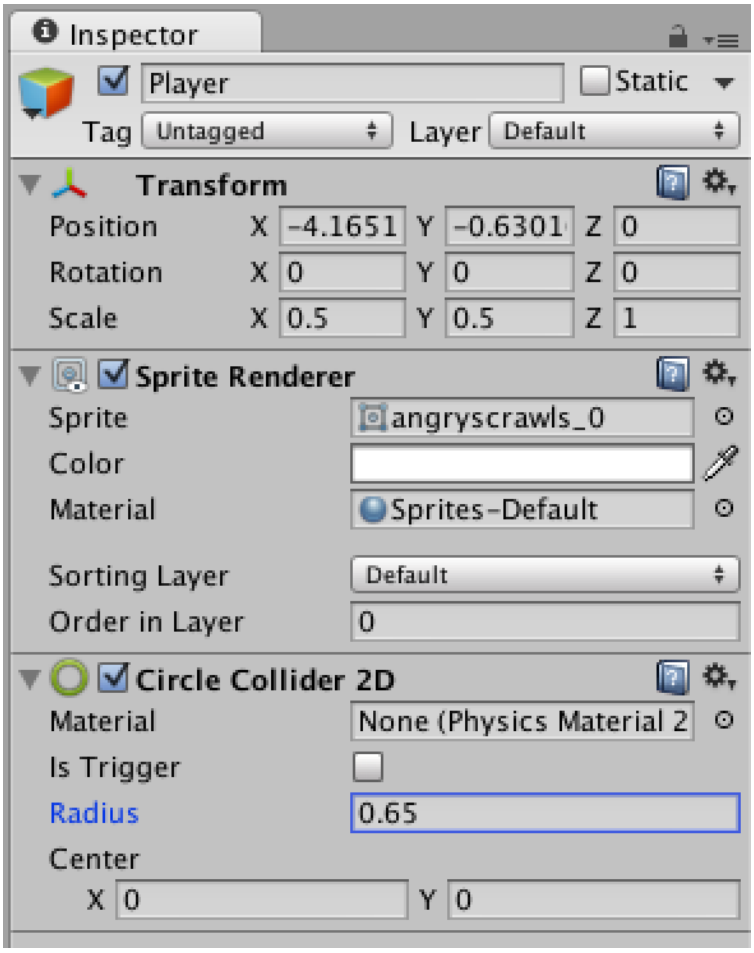
Unity allows us to use a Prefab as a template, so we can create multiple instances of the same item and keep them updated with changes. To add some more crates, just drag them out of the Prefabs window and into the Scene. Make yourself a stack of crates and hit Play to see if they

stay up :)



Now we'll add a player - choose the red angry scrawl and drag it on to the Scene, scale in X and Y to 0.5 and rename to 'Player'.
Add a 'Circle Collider 2D' component and a 'Rigid Body 2D' component. You can see a green circle, you can resize it by holding Shift (as before) or simply change the Radius to 0.65 in the inspector.

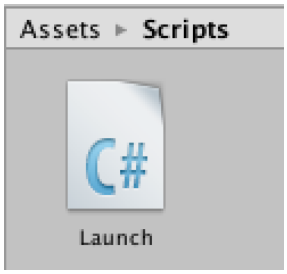




If you hit Run you will see the player drop to the floor. In the rigid body properties, set the Sleeping Mode to ‘Start Asleep’.

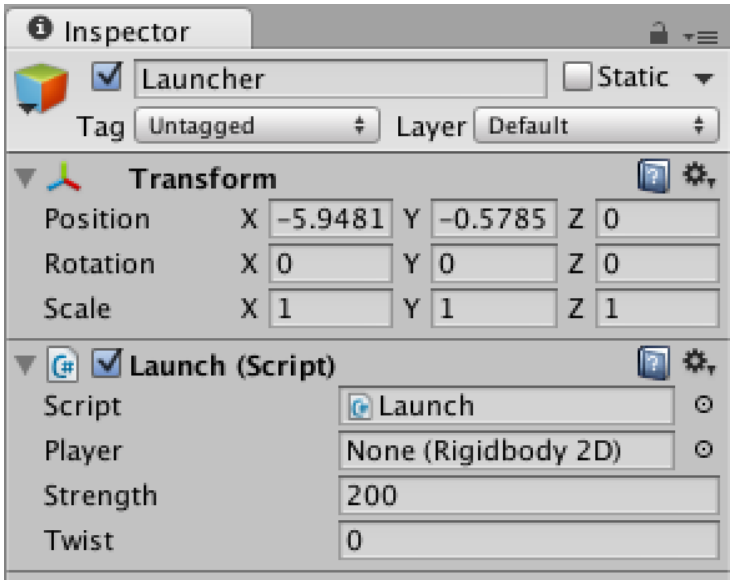
If you Run the game again you will see the player no longer falls.

Now we want to create an aiming arrow to launch the player. Create another empty Game Object, name it Launcher and move it over to the left of the scene. Change to the Assets/Scripts folder and then in Finder/Explorer, find to the ‘Tutorial Assets/Scripts’ folder and drag in Launch.cs.

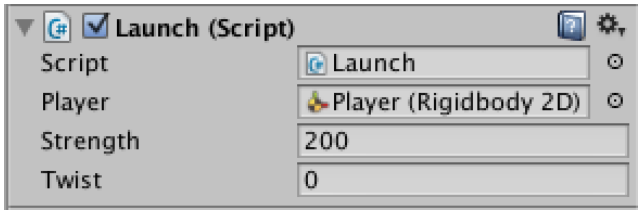


You can then add the new ‘Launch’ component on to the Launcher in Unity. You can either add it the same way as before, or simply drag the script from the asset window on top of the inspector window.

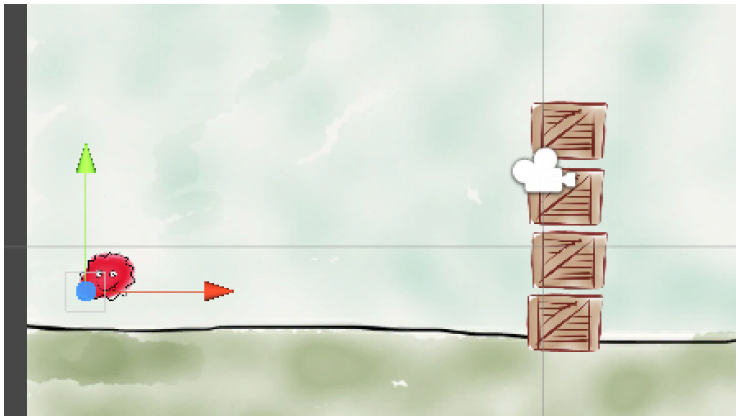
You will see the Launch component has several properties that can be edited:



For now we want to set the Player parameter so that is uses the ‘player’ game object we created earlier. You can drag this from the Hierarchy window, or click on the dotted circle to the right of the Player parameter and then select it from the window that appears. Either way it should look like this:



Now if you Run the game and then click the left mouse button, you will see the Player is launched at the crates! It’s not very strong, so try setting the Strength parameter higher and then try again.



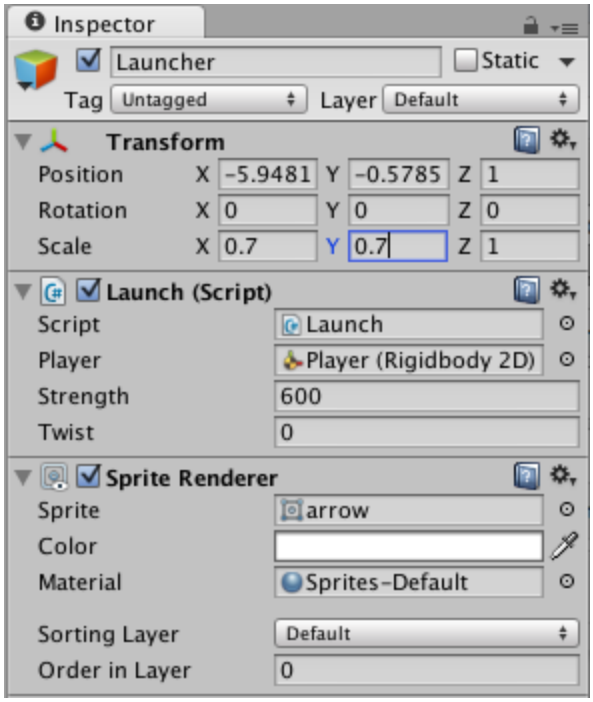
If you double click on the Launch script, it will open in Mono Develop. Do this now so you can look at the code:

```
public class Launch : MonoBehaviour {

    public Rigidbody2D player;

    public float strength = 200.0f;
    public float twist = 0.0f;

    bool launched;

    // Use this for initialization
    void Start () {
        launched = false;
    }

    // Update is called once per frame
    void Update () {
        if (!launched) {
            // launch?
            if (Input.GetMouseButtonDown(0)) {
                Vector2 launchForce = new Vector2(0.5f, 0.5f) * strength;
                player.AddForce(launchForce);
                player.AddTorque(twist);
                launched = true;
            }
        }
    }
}
```

You can see we have three ‘public’ variables at the top of the script, which then appear in the Inspector and can be changed within the editor. Each frame, if we have not yet launched, then we check for a mouse button press with Input.GetMouseButtonDown. To launch, we have to build a force which is a Vector2, holding the X and Y dimensions of the force. We add this to the player using AddForce() which also ‘wakes up’ the Rigidbody2D. To add visual interest we twist the player on launch using AddTorque().

Now we’ll add the aiming Arrow. Go to the Assets/Sprites folder and select the arrow, then set its Pivot = Left and hit Apply.

Now select the Launcher in the Hierarchy and add a Sprite Renderer in the Inspector (by clicking Add Component). Drag in the arrow sprite, and set the X & Y scale to be about 0.7 and the Z position to be 1.



So now we can see we have a nice arrow attached to the launcher. But it isn't launching in the direction that the arrow is pointing in. So open up Launch.cs and change the line where launchForce is set to:

```
Vector2 launchForce = transform.right * strength;
```

Now if you hit Run and click, you can see the player is launched in the direction of the arrow. If you change the Z Rotation of the Launcher, you will see the arrow point up (e.g. set it to 30). Then if you run again, you will see the player is still launched in the direction of the arrow.

So now, we want to change the rotation of the arrow according to where the user's mouse is. Find Launch-2.cs in the Tutorial Assets/Scripts folder and drag it into Mono Develop. Then select all of it and copy/paste it over the older Launch.cs

You will notice that actually only the Update() function has changed and is now:

```
// Update is called once per frame
void Update () {
    if (!launched) {
        // find where mouse hits the world
        Vector3 hitPoint = Camera.main.ScreenToWorldPoint(Input.mousePosition);

        // get relative direction
        Vector2 target = hitPoint - transform.position;

        // rotate launcher to point at target
```

```
            float angle = Vector2.Angle(Vector2.right, target);
            if (target.y < 0.0f) angle = -angle;
            transform.eulerAngles = new Vector3(0.0f, 0.0f, angle);

            // launch?
            if (Input.GetMouseButtonDown(0)) {
                Vector2 launchForce = transform.right * strength;
                player.AddForce(launchForce);
                player.AddTorque(twist);
                launched = true;
            }
        }
    }
```

In the Update function, we now take the Input.mousePosition and map it to point in the world.

```
    Vector3 hitPoint = Camera.main.ScreenToWorldPoint(Input.mousePosition);
```

Then work out the vector between that and the position of the Launcher

```
    Vector2 target = hitPoint - transform.position;
```

Finally figure out the Z Angle from that vector -- this is the same as the ‘Rotation’ in the Inspector.

```
    float angle = Vector2.Angle(Vector2.right, target);
    if (target.y < 0.0f) angle = -angle;
    transform.eulerAngles = new Vector3(0.0f, 0.0f, angle);
```

Now if you Run the game, the launcher arrow follows the mouse cursor!

Now open Launch-3.cs and copy/paste it into Launch.cs. The main changes are the following lines:

```
    float magnitude = Mathf.Clamp(target.sqrMagnitude / maxStrengthAtDistance, minMagnitude, 1.0f);
    float scale = magnitude * maxScale;
    transform.localScale = new Vector3(scale, normalScale, normalScale);
```

We take the distance between the Launcher and the mouse cursor (target.sqrMagnitude)

Also when we launch the player, we use this magnitude to affect the force:

```
    float strength = maxStrength * magnitude;
    Vector2 launchForce = transform.right * strength;
```

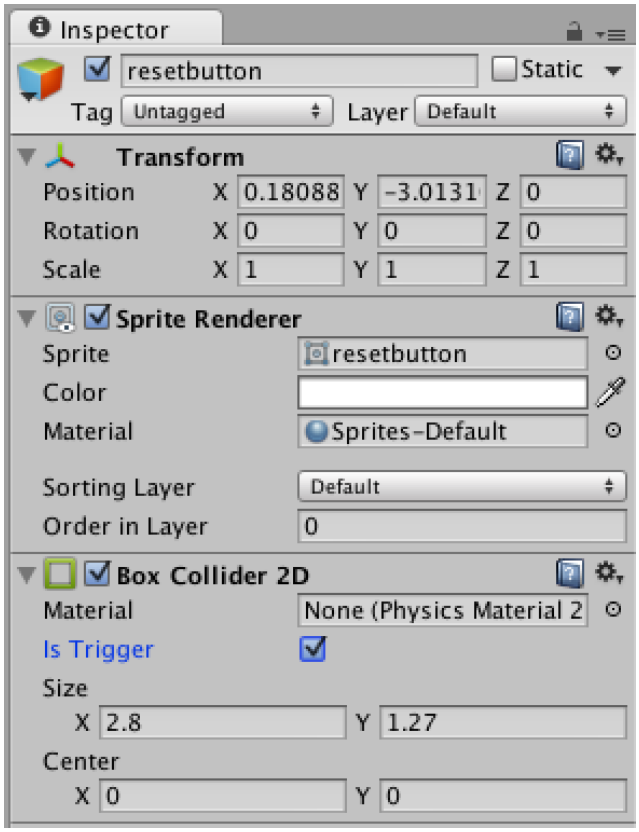
Now when you Run you can see the arrow gets smaller when the mouse gets closer to it. This also affects the speed that the player is launched.

One last change to the launcher - grab Launch-4.cs and copy/paste again!

```
if (!launched) {
    ...
} else {
    if (Input.GetMouseButtonDown(0)) {
        Reset();
    }
}
```

This adds a new behaviour - when the mouse is clicked after the player is launched, the player is reset back to their original position.

The last thing we want to add is a Reset button, to replace all the boxes. To do this grab the reset button from the sprites folder and drag it onto the Scene. Add a Box Collider 2D and set it as a Trigger.



To make the reset button work, find Reset.cs in Finder/Explorer and drag it in to the Assets/Scripts within Unity. Once this is done, drag the Reset script on to the resetbutton in the Inspector.

You can see that there aren't any parameters for that script. If you double click on the Reset script it will open in MonoDevelop - you can see it just reloads the Game Scene:

```
public class Reset : MonoBehaviour {

    void OnMouseDown () {
        Application.LoadLevel("GameScene");
    }
}
```

That's it! You can always add more type of object by just dragging in the sprites, adding collider + rigid body and then dragging them to the Prefab folder to act as a template.