COMP1150/MMCC1011 Week 5 Prac

Topics covered:

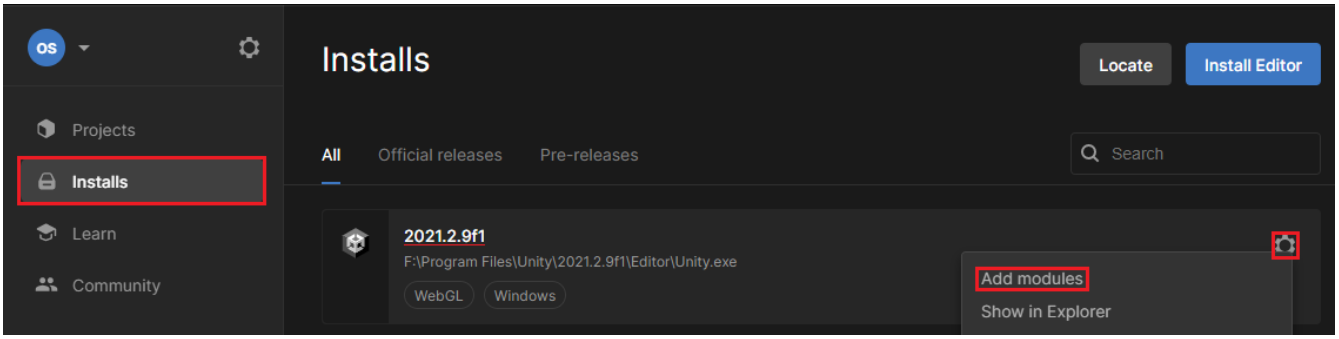
- Physics Materials
- Physics Settings
- Joints
- Building a Project

For our final 2D prac, we will be covering some advanced things you can do with Unity physics, such as making a springboard or a weighted chain. In addition, we will be showing you how you can 'build' your project, turning it into an application that you can run on your computer (or send to a friend).

Downloading the WebGL Module (Online students)

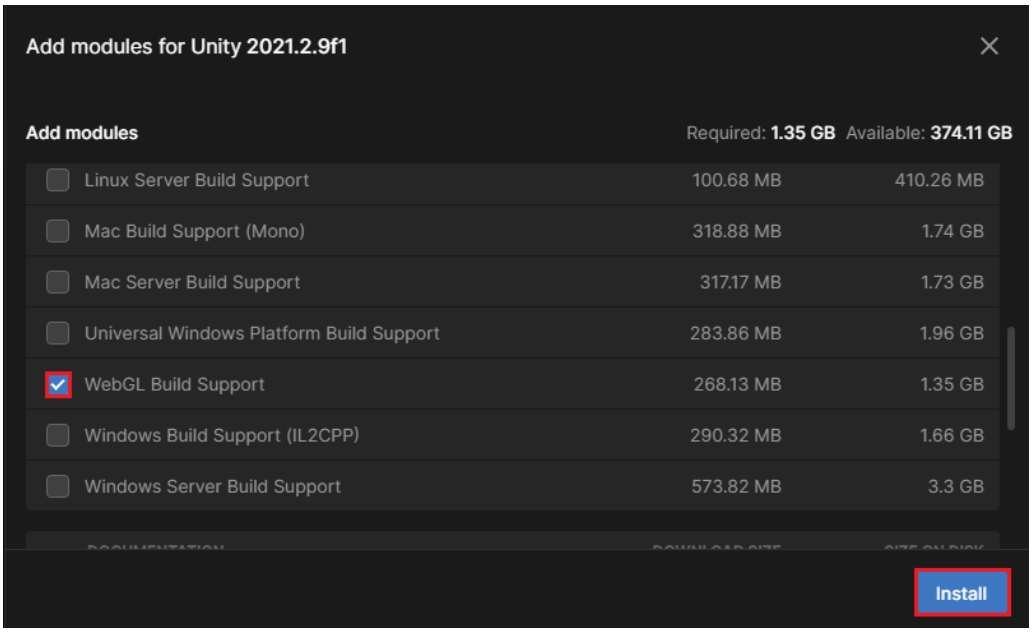
Before starting todays prac, you will need to verify that your device has the 'WebGL' module installed, which is used to build games that can be played in a web browser. If you are in the labs you will not have to worry about this, since the computers in the lab already have this installed.

If you are using your own device, go to the **Installs** tab in Unity Hub, click the cog next to 2021.2.9 and select **Add modules**.



Note: this option will be missing if you have the Apple Silicon version of Unity installed; instead will need to download and install the WebGL module package separately via this link, and then relaunch Unity Hub and Unity following installation.

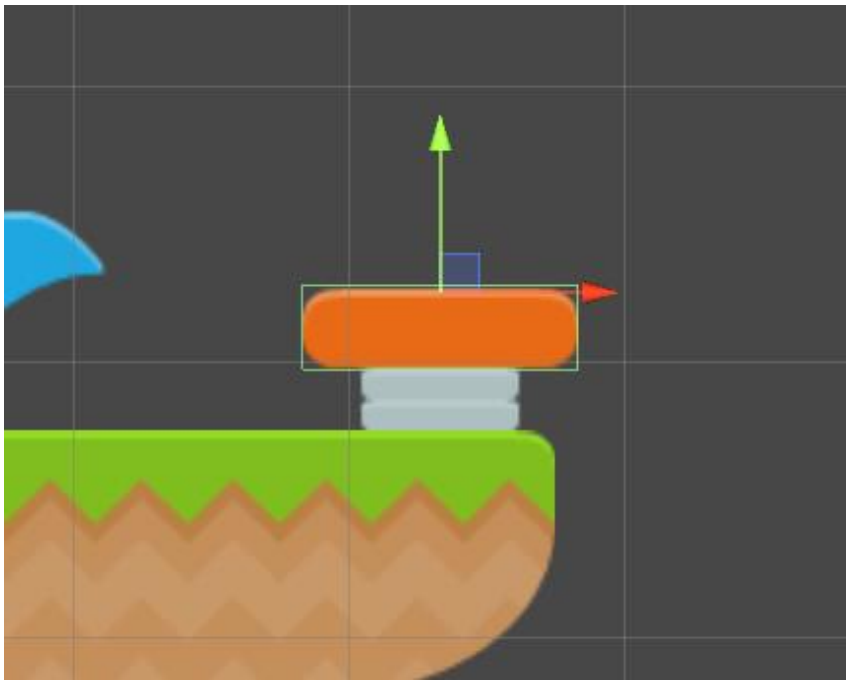
Scroll down the list of modules and select **WebGL Build Support**. Once you have done that, click the blue **Install** button on the bottom right. You can let this install in the background while you continue your prac, but you may need to restart the Unity Editor to complete the last section of the prac.



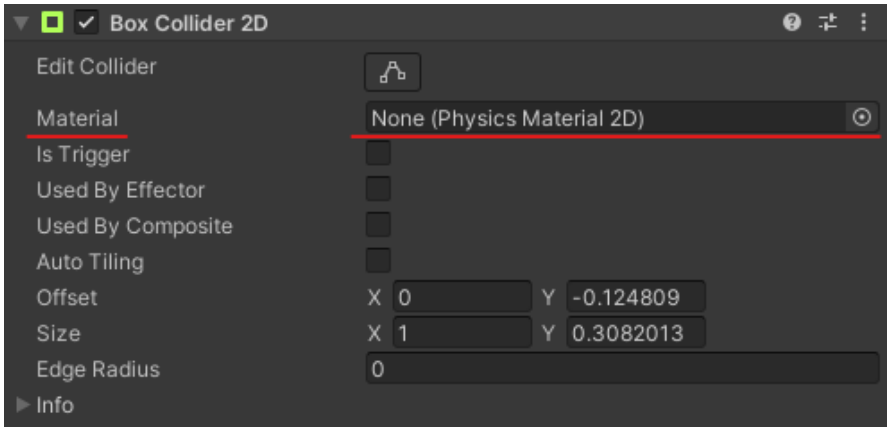
Physics Materials

Go to 'Assets/platformerGraphicsDeluxe/Items' and place one of the springboard sprites (**springboardUp** or **springboardDown**) into your scene (preferably on a Platform).

Add a **Collider2D** component (whichever collider you think is appropriate) and edit the collider so it snugly wraps around the top/orange part of the spring, as pictured below.

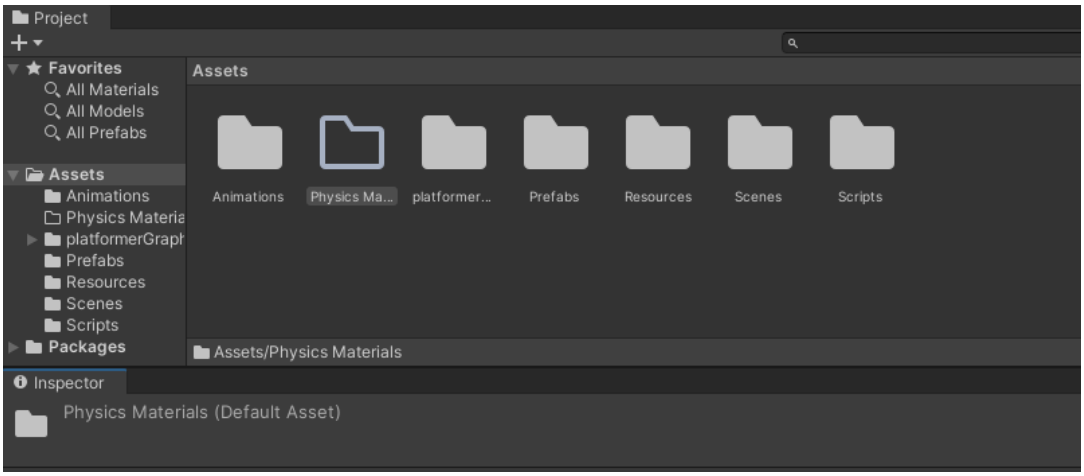


Take a look at the collider in the inspector. You will notice a field marked **Material**, which is short for “Physics Material”.

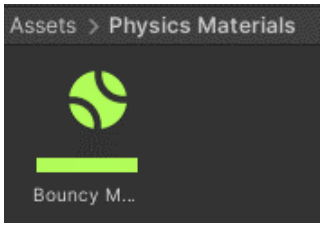


In the context of colliders, a material refers to the kind of physical properties a surface has. Physics materials control things like friction or bounciness of an object. Currently it is set to None, which means that it is using the default values. We can change the physical properties of the surface by creating a new ‘Physics Material 2D’ and putting it in this slot.

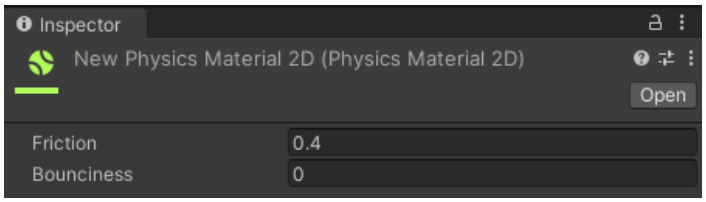
Just like we did for our animation files, in the ‘Project’ panel, within ‘Assets’, create a new folder called “Physics Materials”.



Open the folder, and add a Physics Material 2D (**Assets > Create > 2D > Physics Material 2D**), and call it ‘Bouncy Material’.

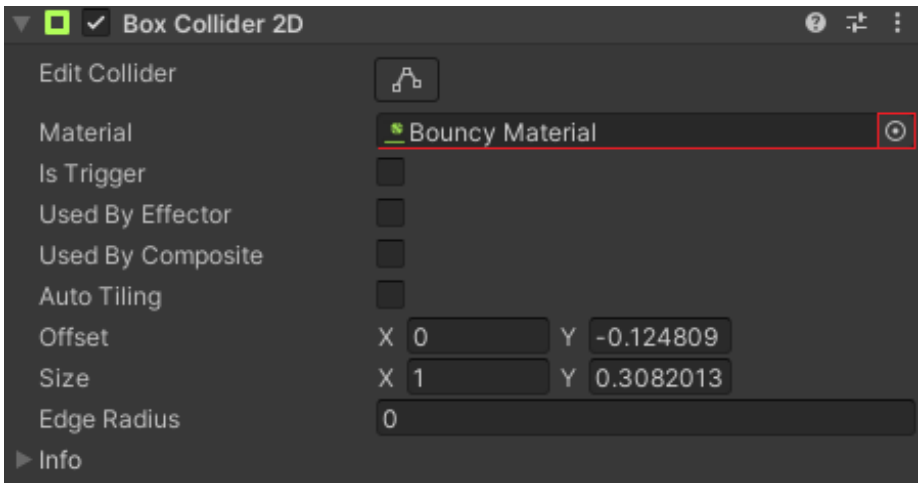


Select this to view and edit it in the Inspector.



You will see two variable names, ‘Friction’, which determines how rough or slippery the surface is, and ‘Bounciness’, which determines how much objects bounce when they hit it. Since we want objects to bounce off it, change **Friction to** 0 and **Bounciness to** 2.

Reselect the springboard in your Scene, and in the ‘Collider2D’ click the **Material’s object picker** button (the button that looks like a circle with a dot inside it, highlighted in the image below). This popup menu will scan your Project for all file types that are, in this case, ‘Physics Materials 2D’. Double click **Bouncy Material** in the results to add it as the material for your collider.



Note: you can also apply elements like this by simply dragging them from the Project panel to their intended **Object Field** (the area underlined in the image above) in the Inspector (i.e. in this case dragging the ‘Bouncy Material’ into the **Material** field on your collider). You might do this if you were just working on the item and have it visible in the Project panel, since you know then that you are selecting the right material.

However, you need to hold and drag; if you select the material by clicking it first then the Inspector will change to show the details of the clicked item. There is a lock button you can click at the top of the Inspector panel that will prevent it from changing to make this process easier (just remember to unlock it again when you’re done).

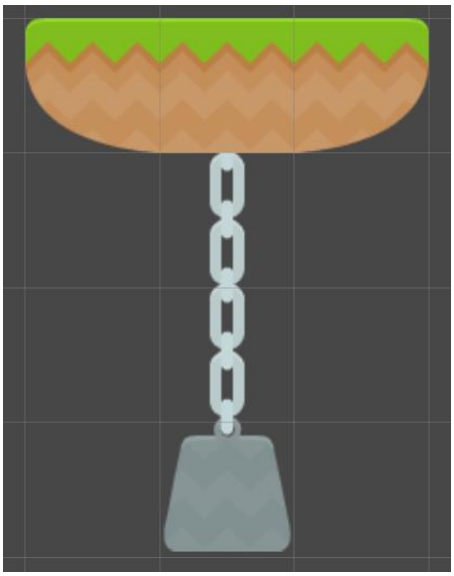
Try having the player jump onto the platform, and you should see that it makes the player bounce quite high. In what other ways do you think you could use these physics materials? Maybe you could make a slippery ice platform, or make it so that the player can jump off enemies (sort of like Mario)?

Hinge Joints

Unity provides a variety of Joints to allow us to connect physics objects together. Today, we'll look at two, the Hinge Joint and the Spring Joint.

The Hinge Joint is used to connect two objects together at a point, while allowing them to turn about that point. It is useful to create doors, chains, rotors and other such things.

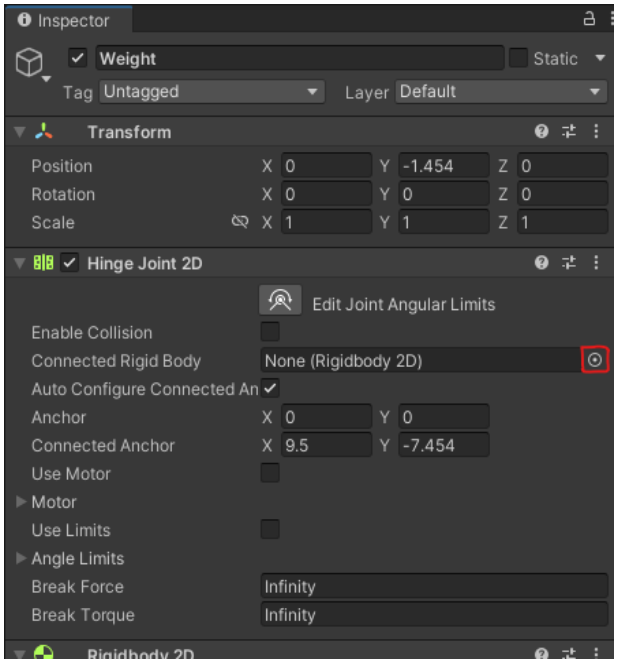
Use a Platform prefab (or a duplicate of a Platform if you haven't made it a prefab yet) alongside the weight and two chain sprites (in 'Assets/platformerGraphicsDeluxe/Items') to create something like the image below.



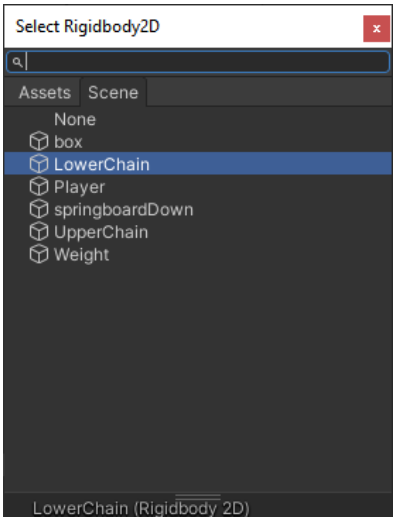
Make a new empty GameObject and make the sprites children of that parent. Also rename the bottom chain sprite to "LowerChain", and the top chain to "UpperChain" so that you can differentiate between the two in the Hierarchy.

Add **Rigidbody2D** components to the weight and the chain sprites so they have physics. Also add an appropriate **Collider2D** (which type?) to the weight so it can collide with other things. We won't put colliders on the chain, so they don't hit objects.

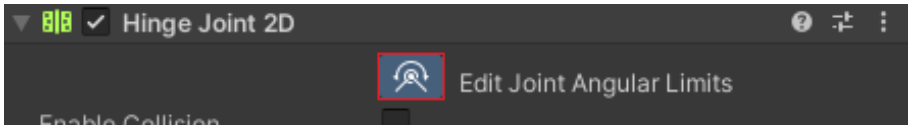
Now to connect them together. Add a **Hinge Joint 2D** component to the weight object (**Add Component > Physics 2D > Hinge Joint 2D**), as shown below.



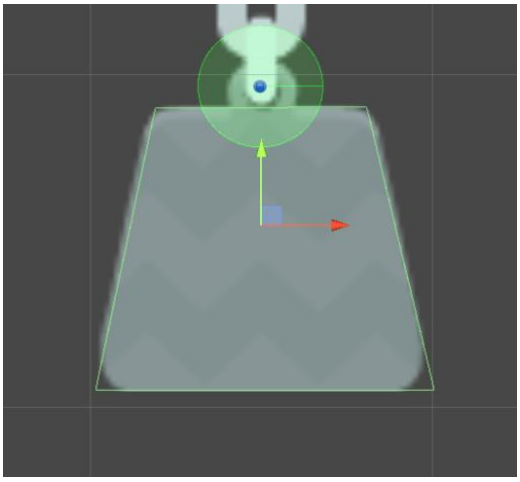
We want to connect this to the chain link above it. Click the object picker button for **Connected Rigid Body** (highlighted in the above image). A popup should appear, containing a list of objects in the scene that have rigidbodies. Click the **Scene** tab, and select **LowerChain**. This will have connected the two objects together.



You will notice that there is now an Anchor point shown in a green circle (technically two, but they are linked together by default for hinge joints). Clicking the **Edit Joint Angular Limits** button, found in the 'Hinge Joint 2D' component, will allow you to drag this Anchor point.



You can also move the anchor point by changing its values in the Inspector. These values are the **Anchor** and **Connected Anchor** values, but the **Connected Anchor** can't be edited right now (since **Auto Configure Connected Anchor** is enabled by default for hinge joints). Make the **Anchor** point line up at the link at the top of the weight sprite, as shown below.



Now 'Weight' and 'LowerChain' will pivot at this location. Do the same process to join this link 'LowerChain' to 'UpperChain', by adding a **HingeJoint2D** to the 'LowerChain', setting its connected **RigidBody2D** to be **UpperChain**, and setting the anchor to the point at the location you want them to join.

Finally, add a **HingeJoint2D** to the topmost chain link, but in this case, leave the Connected Rigid Body as **None**. This means that it is connected to a fixed point in the world. Set the anchor point to indicate where that connection point is (at the top of the chain).

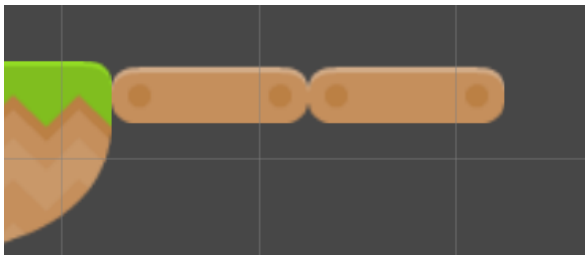
Run the game. If you've set everything up correctly, the weight should just hang in space. Try having the player run into the weight, and it should get knocked around. Alternatively, put the **DragObject2D** script on the weight and see what happens when you drag it around with your mouse.

Try adjusting the **Mass** and **Drag** of the weight's Rigid Body 2D until it behaves in a way that looks appropriately heavy.

REMINDER: Save, Commit and Push to GitHub

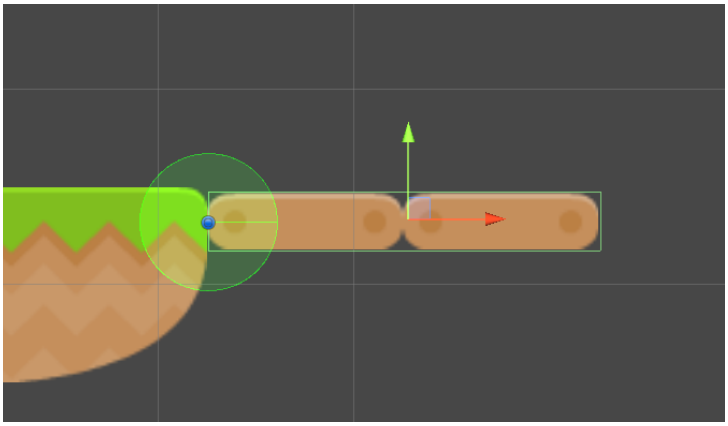
Creating a trapdoor with motors and angle limits

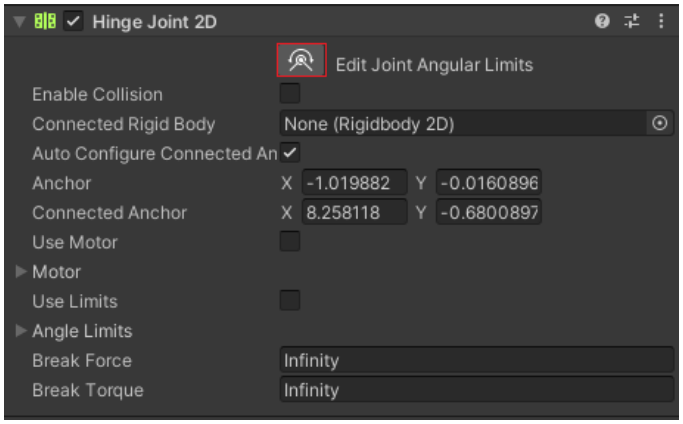
Let's try another hinge joint, this time to create a spring-loaded trapdoor. Lay out some bridge sprites (from 'Assets/platformerGraphicsDeluxe/Tiles') side by side and group them into a single Trapdoor object using an empty parent.



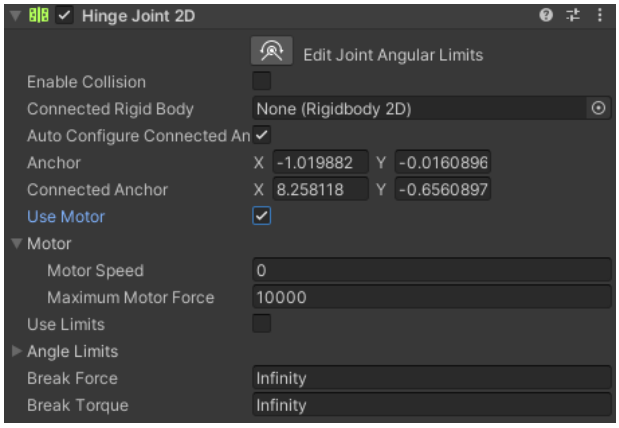
Add a **Rigidbody2D** and a **BoxCollider2D** to the Trapdoor parent object, so that it becomes a solid, physical object in the world.

Now add a **HingeJoint2D** component, so that the bridge is anchored to the world at one end:



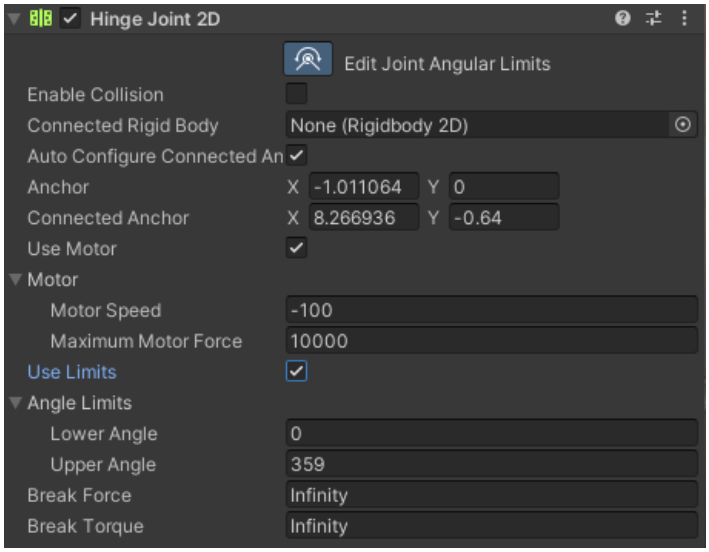


If you play the game now, the trapdoor should immediately fall open, which is not what we want. Go back to the **HingeJoint2D** component and click the **Use Motor** checkbox. Then press the dropdown triangle next to **Motor**. You should see two fields: **Motor Speed** and **Maximum Motor Force**, as shown below.

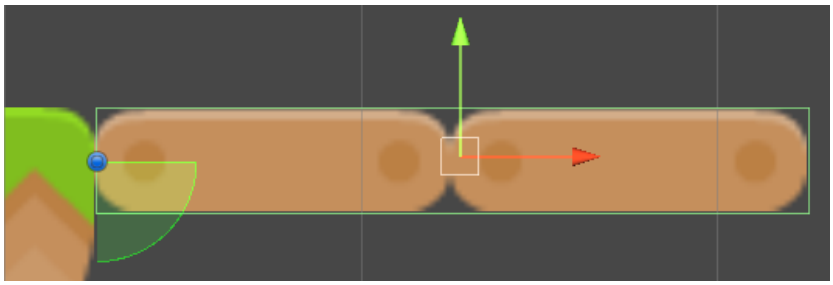


Set the **Motor Speed** to 100 (this value is in degrees per second). Play the game. The trapdoor should start spinning around the hinge joint. This is getting closer to the behaviour we want, but it is still not quite there. First, the rotation is in the wrong direction. Change the **Motor Speed** value to -100 in order to make the hinge rotate anticlockwise.

We also want the trapdoor to stop rotating when it is horizontal (i.e. at 0 degrees). We can do this by putting **Angle Limits** on the hinge. Tick the **Use Limits** checkbox and select the **Angle Limits** dropdown triangle.



The green circle on our hinge joint indicates the range of movement for the trapdoor. We can change this range using the **Lower Angle** and **Upper Angle** values. Leave **Lower Angle** as 0 degrees and set **Upper Angle** to 90 degrees (these are the correct values if you setup your trapdoor the same as ours, to the right of a platform). Your hinge joint's green circle in the Scene view should update to show the constrained angle, so you can use that as a guide.



If you play this, nothing will seem to happen. The **Angle Limits** prevent the motor from rotating the hinge beyond horizontal (0 degrees). If your player character jumps on top of trapdoor, you might see a little movement, but mostly it is pretty solid. This is because our motor is too strong.

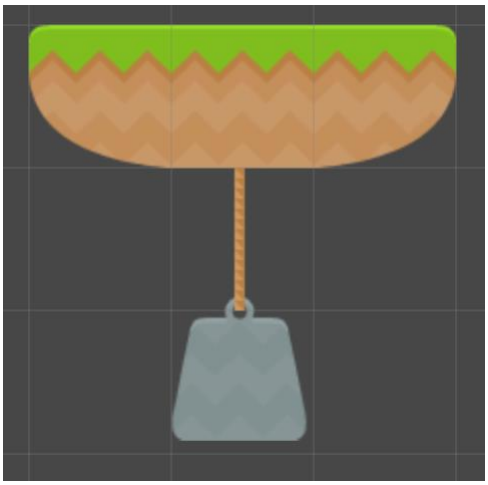
Go back to the Inspector and change **Maximum Motor Force** to 10. This makes the motor much weaker. Play it again. Now when your player character jumps on the trapdoor it will give way, as the mass of the player is stronger than the motor. When the player falls through or is off the trapdoor, the motor will kick in and close the trapdoor again.

REMINDER: Save, Commit and Push to GitHub

Creating a Spring Joint 2D

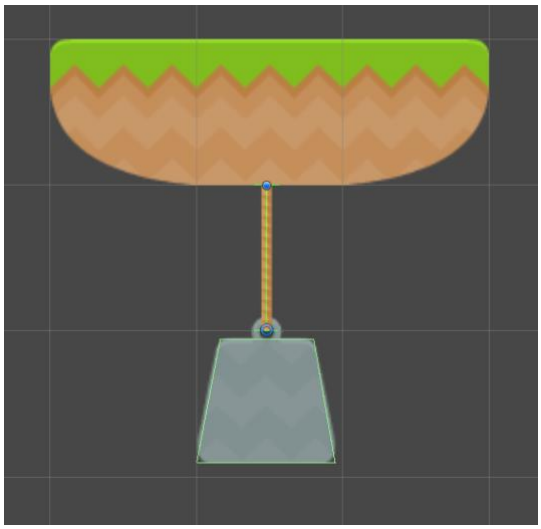
The Spring Joint allows us to connect two objects together with a virtual spring in between. If the spring is stretched, the objects are pulled together. If it is compressed, the objects are pushed apart.

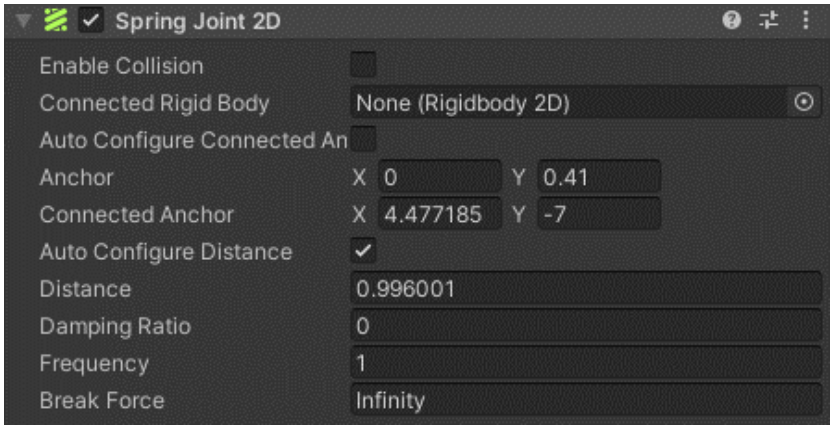
We're going to create a weight on a stretchy rope. Lay out some sprites as pictured (remember to make an empty parent object for the rope and the weight).



Add a **Rigidbody2D** and an appropriate **Collider** to the weight so it becomes a physical object.

Add a **SpringJoint2D** to the weight. As with the ‘HingeJoint2D’, we need to specify the **Anchor** for the spring, which should be at the bottom of the rope, as well as the **Connected Anchor,** which should be at the top of the rope, as pictured below.

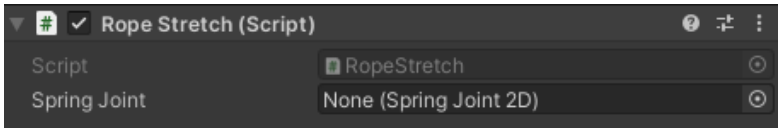




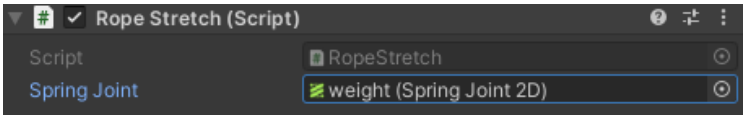
As with our ‘UpperChain’, we want the **Connected Rigid Body** to be left as **None** so that the weight is anchored to a point in the world, rather than to another object in the scene.

If you run your game, you should see the weight magically bob up and down on its spring without the rope moving, since we haven’t connected it to anything. There is no built-in way to visualise the spring, so we have included a little script which moves and resizes the rope to match up to the ends of the spring.

Add the **RopeStretch.cs** script to the rope object (**Add Component > Scripts > Rope Stretch**).



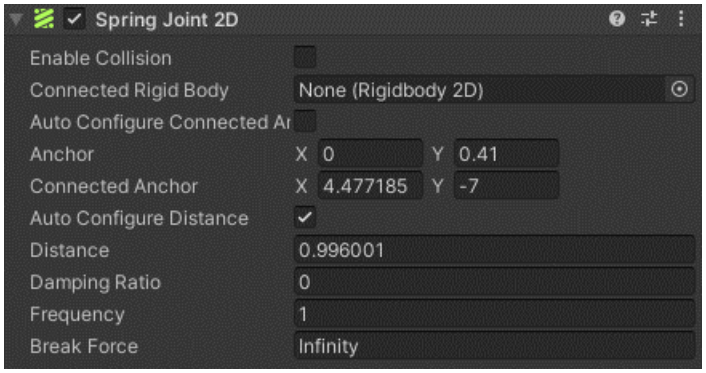
The rope needs to know which spring joint it is connected to. Click **Spring Joint's** object picker button to get a list of objects with a 'SpringJoint2D' in the scene. Select the **weight** object.



That's all you need to do. The rope should now automatically resize itself to match the spring joint. Run your game to verify that it works. Again, you might like to add the **DragObject2D** script on the weight so that you can play with it directly with your mouse.

It's worth noting that 'SpringJoint2D' has several parameters that affect how it behaves:

- **Enable Collision:** If this is deactivated, the weight will not register collisions with the object it is attached to.
- **Distance:** This defines the resting length of the spring.
- **Damping Ratio:** This controls the damping of oscillations in the spring.
- **Frequency:** This controls the frequency of oscillations (which corresponds to the strength of the spring).



Other Joints

Unity also includes other joints, such as:

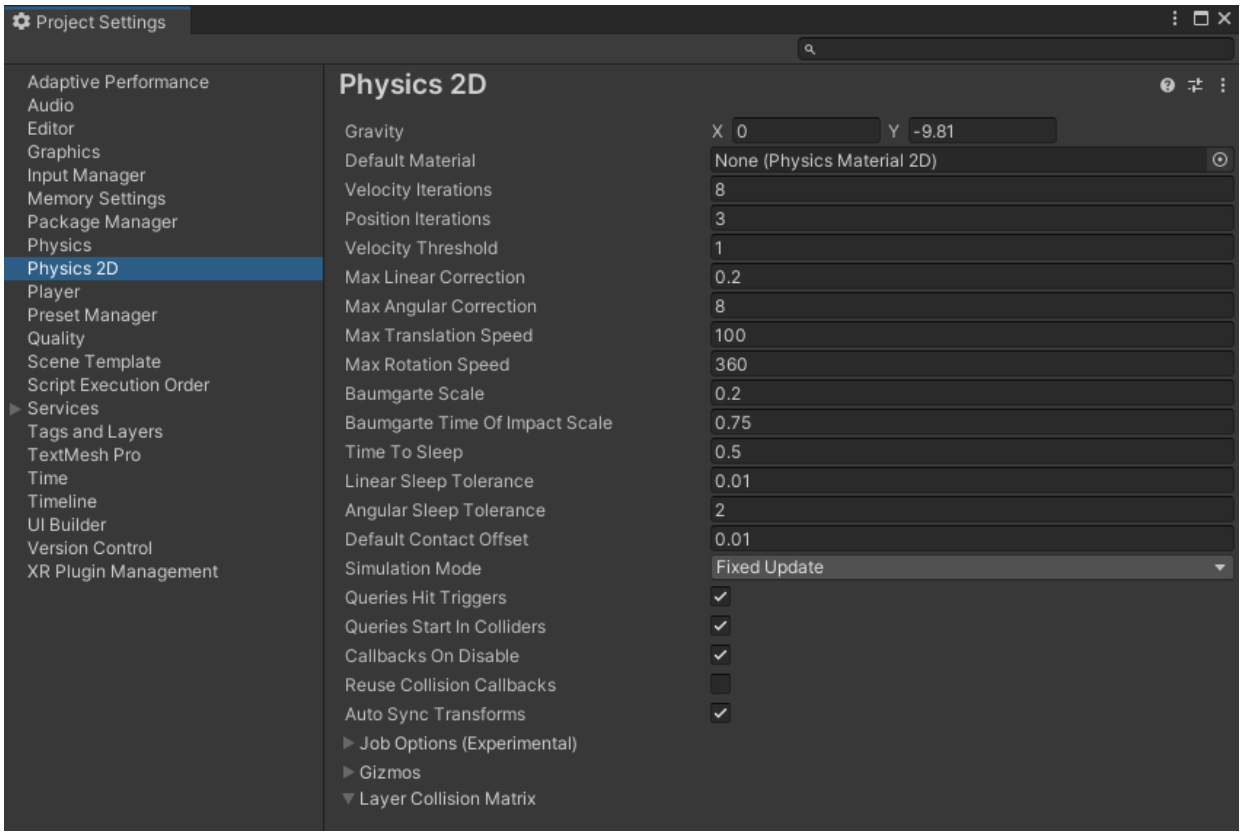
- The **Distance** Joint, which keeps two objects a specified distance apart.
- The **Slider** Joint, which restricts an object to move along a line.
- The **Wheel** Joint, which combines a hinge and a spring to make a spinning wheel with a suspension.

You can read more about these joints in the 2D Physics Reference section of the Unity Documentation.

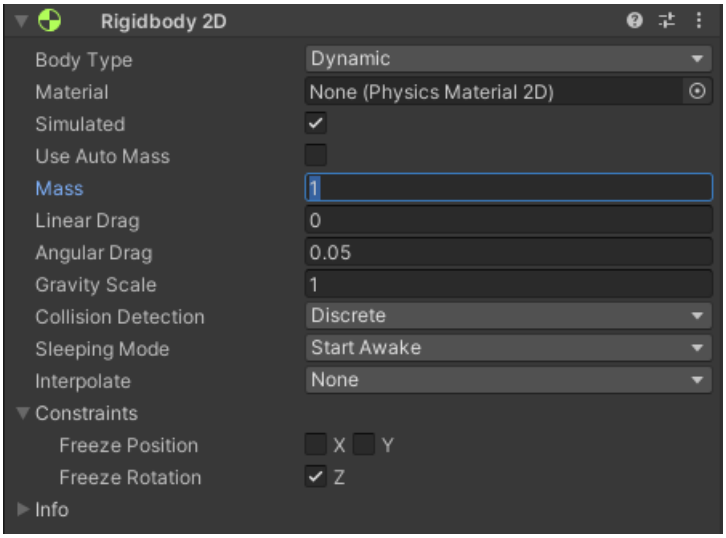
Adjusting Physics Parameters

There are multiple parameters we can adjust to fine tune the physics of our simulation and get it feeling the way we want. Some of these are global, and affect all objects in the game, while others are local. Have a look through the outlined parameters below and briefly experiment with changing these values to fine tune the physics of your game (there's still a couple more sections of the prac to get through after this, so don't spend too much time here).

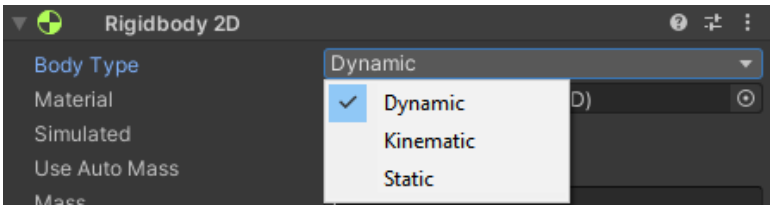
Gravity (global): From the menu bar, select **Edit > Project Settings > Physics 2D**. A window will pop up, showing a list of parameters for the physics system (you can dock this window if you like, just like the 'Animation' and 'Animator' windows from last week). Most of these parameters are very technical, but the top one, Gravity, is easy to understand. It has two components X (horizontal) and Y (vertical). By default, X is 0 and Y is -9.81 (which is equivalent to Earth's gravity). Experiment with changing these values and see what effect it has on the game.



Object mass (local): The Rigidbody2D component contains a number of different parameters. The first is **Mass**, which determines how heavy an object is. **Mass** affects how objects respond to collisions, such that a heavier object has greater inertia than lighter ones. You already played with this property for your weight on the chain, so make sure you're happy with these values and that both your weights feel appropriately heavy.



Body Types (local):



- ***Dynamic*:** The default body type for a Rigidbody2D. It's the most interactive of the types, being able to collide with all other body types. It also allows access to all variables. It is also the most performance-intensive body type, because of its dynamic and interactive nature.
- ***Kinematic*:** Moves only under explicit input from the player/designer, this body type is typically reserved for animated or script-driven objects. Only collides with dynamic Rigidbodies (unless **use full kinematic contacts** is enabled). Behaves like an immovable object until 'activated'. This can be used to make an animated platform.
- ***Static*:** Functions as an immoveable object, which can have simulation of physics toggled. For example, an object that previously moved but is now locked in place.

Linear Drag and Angular Drag (local): These variables control friction on the object, i.e. its tendency to slow down. Objects with increased linear drag slow down more quickly. Angular drag functions the same but affects rotation. These properties are useful to dampen the movement of an object. For example, setting the player character's **Linear Drag** to 1 prevents it from continuously gaining height when using the springboard (if **Bounciness** of the springboard's material is set to 2), but this will also affect the player charcter's jump arc/height.

Gravity scale (local): This attribute lets you reduce, remove, or even negate the effect of gravity for a particular object. Experiment with different values to see what it does, for example, try and tweak it so the player character feels like they're on the moon, or a bomb sprite is actually a helium balloon.

The other parameters are more technical, and we will ignore them for now. Check the Unity documentation if you want to see more details.

Other Physics Ideas

If you have time or want to learn more in your own time, try making additional changes to your scene applying physics in new ways. Here are a few ideas:

a) A bouncy pool of gems
b) A Newton's cradle
c) A catapult
d) A floating island

Tidy Up Your Project

Since this is the last time we will be working in this project, make sure that it is organised in a professional manner:

- Make sure everything in your Project and Hierarchy has **appropriate/meaningful names** describing its purpose (e.g. not "Empty" or "grassMid (5)"). Have a consistent naming style (i.e use capitalisation and spacing consistently).
- Use **sub-folders** to organise your Assets folder. At the top level assets should be grouped by type (i.e. "Animations", "Sounds", "Physics Materials", etc.). Lower levels could be grouped by game objects, depending on the complexity of your scene (e.g. "Player", "Enemies" and "Environment" folders underneath "Animations").
- Use **empty GameObjects** to organise your Hierarchy into meaningful groups (e.g: "platforms", "Coins", "Environment", etc).
- Use **Prefabs** where it makes sense, such as coins, platforms, weighted chains, etc.

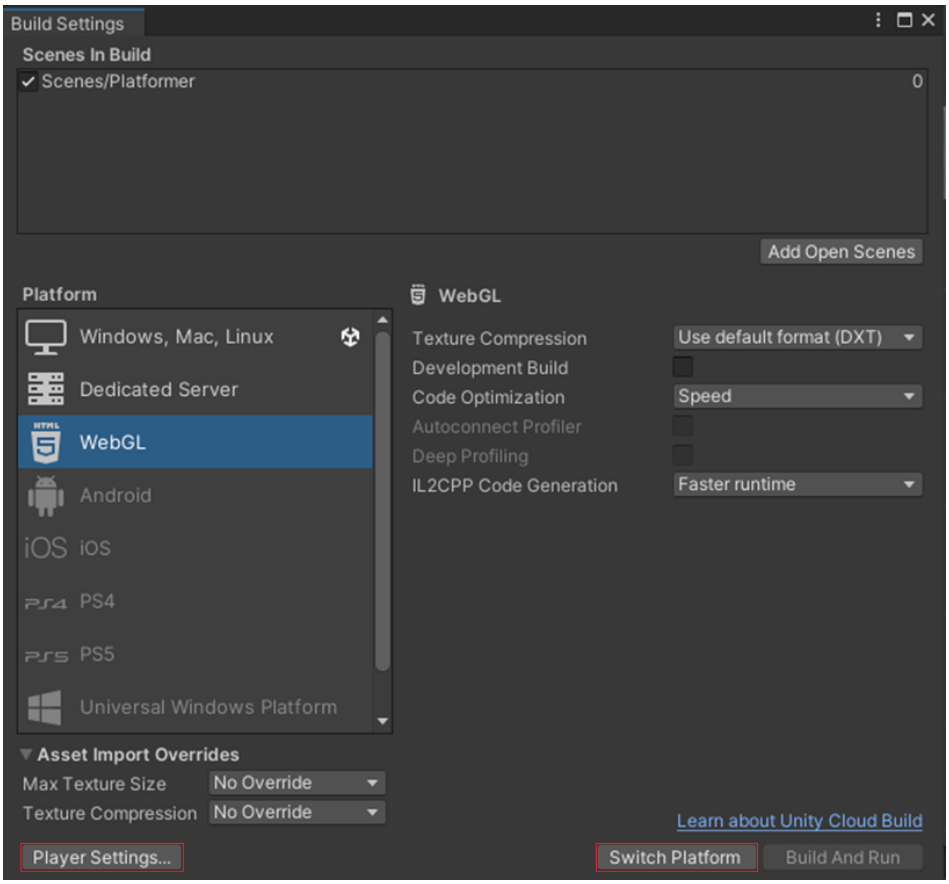
For more specific advice, check out this article for Unity best practices, specifically focusing on the section at the end on naming. Good developer practices like these will make your life much easier in the long run.

REMINDER: Save, Commit and Push to GitHub

Note: if you started installing the WebGL build module at the start of this prac, you will need to make sure that is finished and relaunch your Unity Editor at this point.

Building a Game

To make your project a game that other people can play, you will need to ‘Build’ your project. Select **File > Build Settings** and a new window will appear.

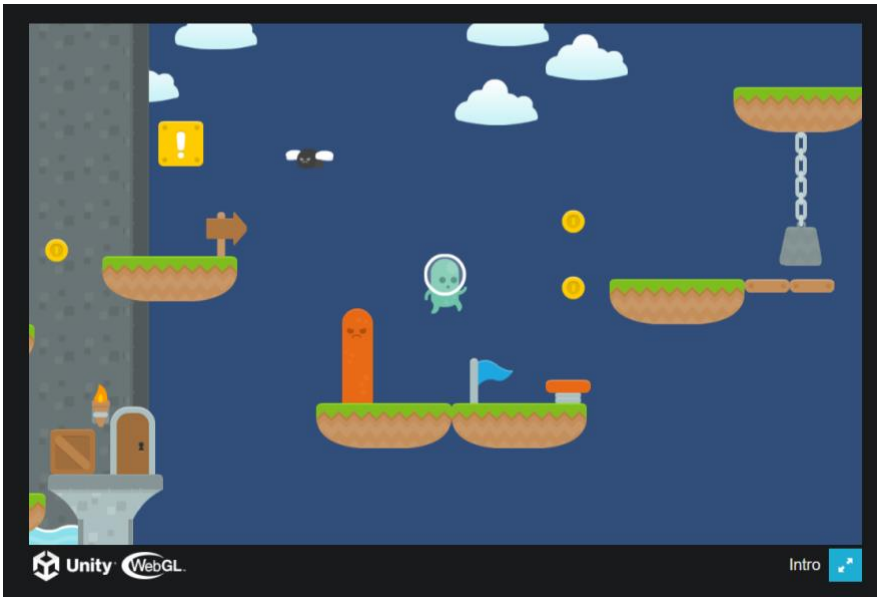


Unity can build to a variety of different platforms, as shown in the list on the left. For now we are going to make a **WebGL** build. Select **WebGL** in the list of Platforms and press the **Switch Platform** button. Once you have done that, click the **Add Open Scenes** to ensure your scene will be added to the build. Before building a game, it is always recommended that you double-check your **Player Settings. Player Settings** can be used to change things like the resolution of the game, but allow you to modify other attributes of your game executable, such as the icon and version info (but most of this can be ignored for now).

Once you have checked the settings, select **Build And Run**. You will be prompted to select a folder to place the game data. **Do not build it inside your project or Assets folder**, instead**:**

- **On PC:** create a new folder called “COMP1150 2D game” in your GitHub or another documents folder and build your game in there.
- **On Mac:** the folder containing the build will be created automatically, so just save as “COMP1150 2D game” in your GitHub or another documents folder.

Unity will build a folder structure (including a html and other necessary files) for your game and automatically open it in your default browser. You may need to modify your browser's security settings to enable WebGL content to run. Once the WebGL build loads, your game should be playable within your browser.



You can also create an executable build of your game for **Windows/Mac/Linux** if you prefer. This can be done by selecting the intended platform in the Platforms list in Build Settings, and clicking the **Switch Platform** button. It is again advised that you double-check your **Player Settings** before building your game (especially the **Resolution and Presentation** settings).

PC Build note: by default Windows builds are set to run full screen, and since we haven't programmed a way to quit the game you will need to press Alt+F4 to close the application, or use Alt-tab to switch back to the Unity Editor.

Mac Build note: by default the security settings for MacOS do not allow you to run 'unsigned' applications without deploying them through Xcode (Apple's development platform). There are workarounds to bypass this and get Unity Mac builds to run, however these involve using terminal to override security settings and so won't be covered here.

Show your Demonstrator

To demonstrate your understanding of this week's content to your prac demonstrator you might show:

- Your springboard with its bouncy material
- An object with a working Hinge Joint
- An object with a working Spring Joint
- An object with Mass or Drag values changed from the default
- That you know how to build and run project
- How cool your game is!