Unity Lander 2

Instructor Guide

An instructor's guide to
getting started with Unity

Table of Contents

Project Introduction

Overview

Create a fun physics-based game using Unity with no coding required. This lesson is the final project of the Unity Lander series!

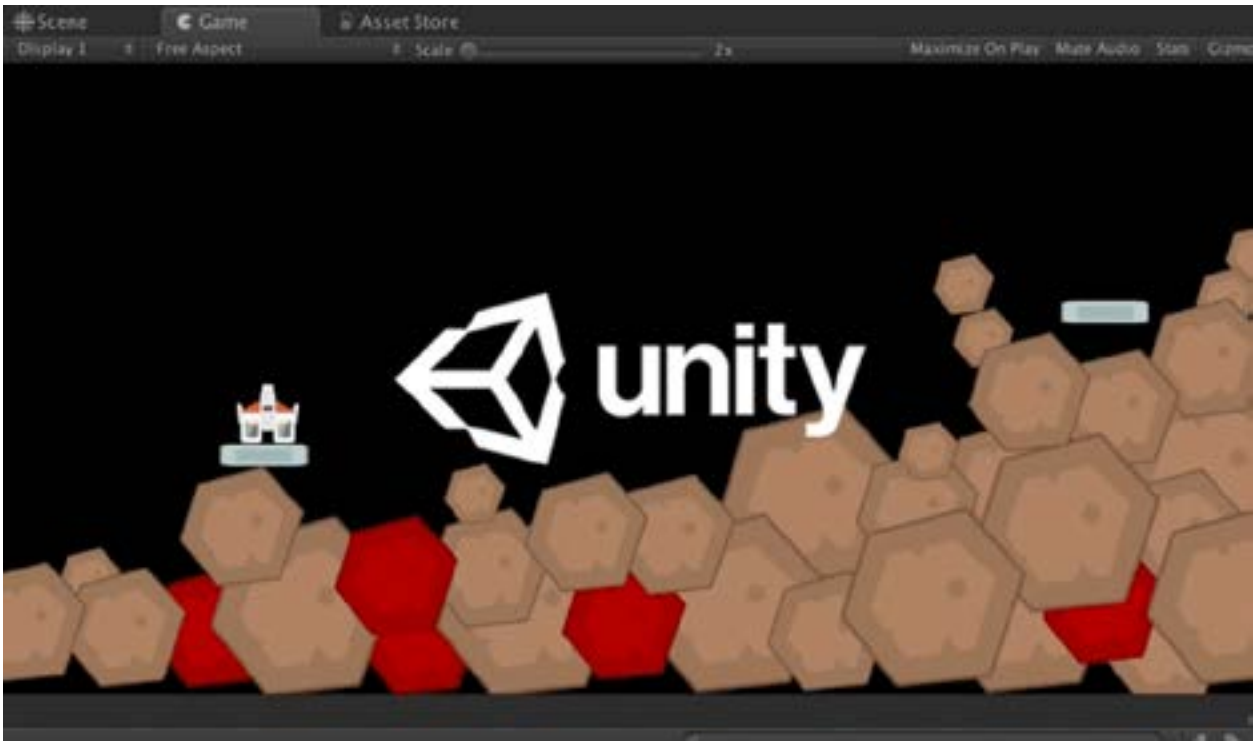

Figure1: Unity Lander Moon Lander

Introduction

Build the game you played in Unity Lander 1 all on your own. Create a world full of hazardous obstacles and set up a condition that will make your ship explode upon impact. Teach your game to tell you when you win or lose. Make sure you have completed Unity Lander 1 and have imported the Moon-Lander-Package

Educators - It is highly recommended to read through each project steps as you prepare for facilitating this project.

Intro to Unity

Background

Overview

Learners build on the Unity skills they learned in Unity Lander 1 to create a Moon Lander game from scratch. This lesson takes apporximately 35-45 minutes of instructional time.

Course

This project is part of Mouse's Serious Games course. If learners finish this project successfully, they will have completed 1 of 2 Unity projects from Mouse's Serious Games course. The full Serious Games course is a game design course focusing on creating games that try to do more than just entertain using Scratch, Inklewriter or Unity.

Mouse is a youth development non-profit with the that teaches technology with purpose. To learn more about other Mouse courses, please visit our Course Directory.

Associated Projects

The other project associated with this one is:

- Unity Lander 1

Outcomes

You will be able to:

- Create Prefabs in Unity
- Create Instances of Prefabs
- Use Unity components to detect collisions between GameObjets
- Use Unity scripts to trigger actions after two GameObjects collide
- Use Unity scripts to display a game-win and a game-lose message.
- Set a Build Target for a Unity Project
- Find the Playground Project to go further with Unity

Standards

All Serious Games activities have been aligned to Common Core, Next Generation Science (NGSS) and International Society for Technology in Education (ISTE) learning standards. You can find all of those alignments here:

- Common Core & ISTE
- NGSS

Gear

- Unity software
- Laptop or desktop computer that meets the following system requirements
- A Unity for Education license obtained by Unity prior to installing the software (if you are doing this in a classroom setting) OR a Unity Personal Edition license (if you are doing this activity on your own)
- The Mouse Lander Package available here: Moon-Lander-Package
- The Unity Manual

Activity Flow

- **Step 1: Intro (1 minute)** Introduction to the project.
- **Step 2: Creating a New Scene. (3 minutes)** Create a new blank scene to build your game.
- **Step 3: Make a MoonRock Prefab! (3 minutes)** Create another Prefab
- **Step 4: Scenery (3 minutes)** Make many instances of your MoonRock Prefab and make them all children of a new empty GameObject called Scenery.
- **Step 5: Bring Back the Player Prefab (1 minute)** Add the Player Prefab you created in **Unity Lander 1** into your current scene.
- **Step 6: Starting and Ending Platforms (3 minutes)** Create a start and end location to define the goal of our game.
- **Step 7: Making the Ship Explode (3 minutes)** Make our ship Collider trigger an explosion animation.
- **Step 8: Add Game Over Text User Interface (3 minutes)** Set up the **UserInterface** so that your game gives you feedback when you lose.
- **Step 9: Add Win Text to the User Interface (3 minutes)** Set up the **UserInterface** so that your game gives you feedback when you win.
- **Step 10: Testing, Debugging and Background Image (3-10 minutes**) Test your game both for difficulty and for technical issues. Set a background image if you want,
- **Step 11: Building Your Game (3 minutes)** Build your game so that you can share it.
- **Step 12: The Playground Project (5 minutes).** Get access to more Sprites, scripts you can use to build new games with the skills you've learned here.
- **Step 13: Beta Feedback (optional)**

Facilitators: These times are rough estimates. Be sure to look to the content of the project's steps to help you plan the timing, flow and content of the lesson for your specific group of learners.

Prep

- Learners should have completed Unity Lander 1 before doing this project.

Glossary

- **Asset:** A file that is part of your game. For example, an image file, a code file, a sprite sheet, etc.
- **Build target:** Unity can create games playable on a wide variety of platforms from phones to browser based games to PC and Mac based games. The built target is whatever platform Unity is currently set to export games for.
- **C# (pronounced c sharp):** A programming language that uses classes.
- **Debugging:** The process of fixing technical problems in a game or program.
- **Focus:** When using Unity, bringing something into focus means it appears in the inspector.
- **GameObject:** A GameObject in Unity can be an object in your game, a part of the background, or it can be visually empty but contains information, for example text from your user interface, a menu, or a score counter.
- **Game view:** A window that shows what your game looks like to a player.
- **Hierarchy:** The active GameObjects in your scene.
- **Inspector**: Shows you details about an asset or GameObject. Allows you to set layers, add tags, and add components.
- **Instance:** An individual clone of a Prefab. Each instance appears in the Hierarchy.
- **Prefab:** A GameObject template that allows you to create many copies of GameObjects with the same specifications.
- **Project:** Your whole game is your project. Your project can be made of one or many scenes.
- **Project window:** The window that shows you the folders, scenes and assets that are in your game.
- **Scene:** A scene is a section of your game. It can be a single sequence, a level or some other slice of your game. A game can have only one scene, or it may link several scenes together.
- **Scene view:** A window in Unity that allows you to build the world of your game, where you can select, position and manipulate GameObjects. It shows you all the active objects in your Hierarchy in a visual representation.
- **Script:** A piece of code that can be attached to a GameObject in Unity.
- **Sprite:** A 2D graphic.

Evidence

What will learners do at the end that will demonstrate what they've learned?

At the end of the project, learners will have used Unity to export a basic digital game they built from scratch in which a spaceship must navigate through asteroids to reach a platform on the other side of the screen.

Evidence Checklist

What should teachers look for to know that the evidence above is a success? You may realize after writing these that you need to change/edit the learning outcomes at the top.

Learners have met target outcomes if:

- ☐ They have a game file that can be run on a Mac/PC or, if they built their game for the web, they have a folder containing game information and an index.html file that will open their game in Firefox or could be loaded to a web server.
- ☐ The game has three Prefabs, a UserInterface, Player and a MoonRock in the Prefab folder.
- ☐ The game contains a spaceship that can fly around the screen when controlled with the keyboard.
- ☐ There is a GameObject in the Hierarchy called **Scenery** that contains many **MoonRock** Prefabs.
- ☐ The spaceship explodes when it collides head-on with an asteroid.
- ☐ The game says Game Over when your spaceship explodes.
- ☐ The game says You Win when you land on the End block.

Debrief Questions

- What was the hardest part?
- Why might you want to use one kind of Collider with one GameObject and not another?
- What is different about your Moon Lander game?
- What was something that surprised you about the game now that you had to build each GameObject yourself?

Extension ideas

What can learners do if they finish the project early? Are there ways educators can level the activity up or down?

- The Moon-Lander-Package is a selection of assets taken from the Unity Playground project. The full playground project contains many more Sprites and and scripts for learners to make games. The collection of scripts are useful for allowing learners to play and experiment with common behaviors like keyboard movements and Colliders as

triggers that can be used to create a variety of different games.

- For those who want to try a tutorial featuring the newest features of the Playground Project try this tutorial on new features.
- Additional Unity tutorials can be found here

Project Steps

Introduction

Instructor note: Repeat the Introduction and the first step.

Build the game you played in Unity Lander 1 all on your own! Create a world full of hazardous obstacles and set up a condition that will make your ship explode upon impact. Teach your game to tell you when you win or lose. Make sure you have completed Unity Lander 1 and have imported the Moon-Lander-Package before doing this project.

Creating a New Scene

We're going to create a new scene so we can create our game from the ground up. Scenes are parts of your game. Some games have only one scene, some games have several levels and each level is a new scene. You can only have one scene open at a time when editing a game in Unity.

Unity should be open with the **Moon-Lander-Package** of assets imported. If you closed Unity in between the last activity and this one, open the Moon Lander project (or whatever name you gave it) by going to the File menu and selecting Open Project.

Once you are back in the Moon Lander project go to the File menu in the upper left and select New Scene.

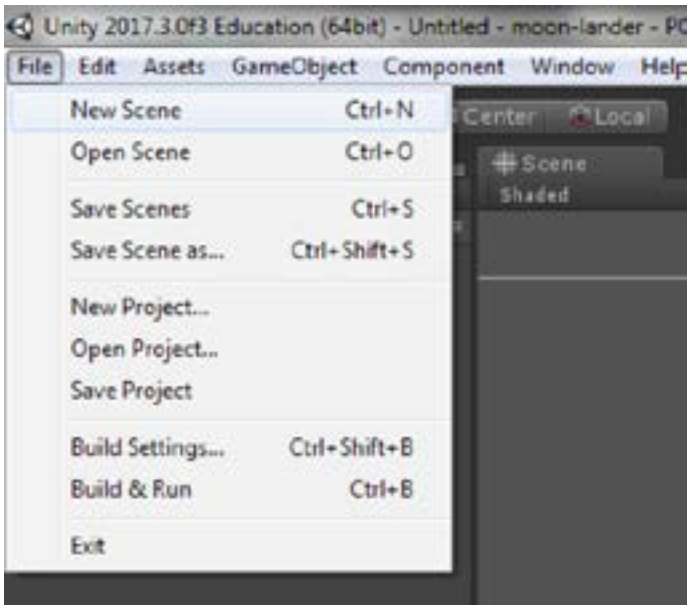

Figure 2: File menu open and new scene highlighted

Now we have a new blank scene. This is where we are going to build our new Moon Lander game from scratch. Games in Unity are made up of one or more scenes, and scenes contain all the information for your game. The main things you will add to these scenes are GameObjects and Scripts.

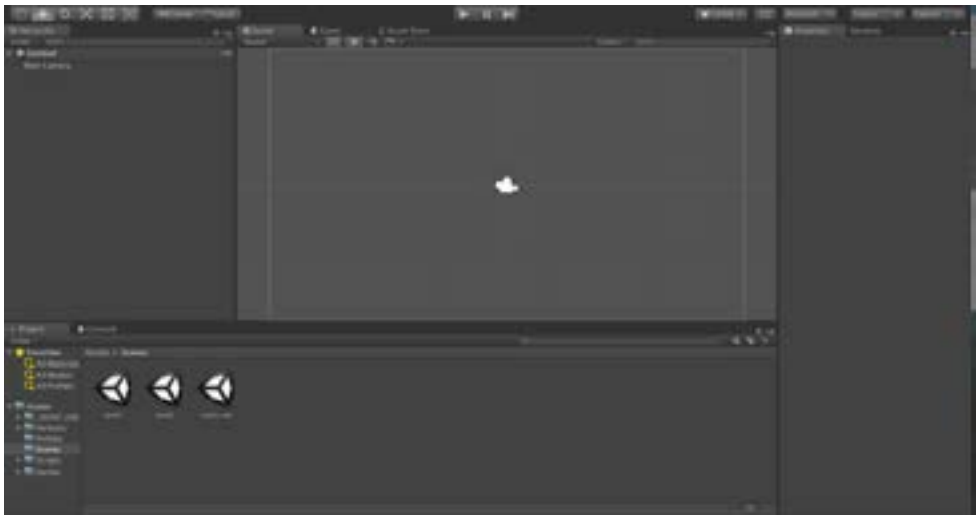

Figure 3: A blank scene

Remember, GameObjects are the objects that interact in your game. This can be your game characters, collectible items, enemies, anything you can think of. GameObjects can also be used in a more abstract way as an organizational tool for your game. For example, it's common to have empty GameObjects that just act as containers for sets of smaller GameObjects. GameObjects can be made out of assets you import into Unity. Every game in Unity is made out of GameObjects. We'll make some in the next step.

Scripts are pieces of code that control your GameObjects and make them do things like respond to your controls or appear and disappear. These will be discussed later in this project.

Now go to the File menu and select Save Scene As and name this scene "moon-lander." Save the scene in the **Assets** folder.

In order to stay organized, we're going to put our new scene in the same **Scenes** folder that has all our other scenes:

Select the **Assets** folder on the left side of the Project window and drag and drop the new moon-lander scene into the **Scenes** folder.

Figure 4: Drag the scene into the scenes folder

Congratulations, you've created and saved your first new scene. You'll start adding GameObjects to it in the next step.

Make a MoonRock Prefab!

Let's start by making an obstacle for your ship to crash into or land on: an asteroid. You might want to deactivate your Main Camera while building your asteroid. (Remember: this means bringing the Main Camera into focus by clicking on it in the Hierarchy, then unchecking the box next to the "Main Camera" title in the Inspector).

- Go to the **Asteroids** folder inside the **Sprites** folder and select any of the asteroids. Drag it into Hierarchy to create a new GameObject. Rename this GameObject "MoonRock."
- Select it in the Hierarchy so that it is in focus and set both the X and Y in the Scale area of Transform to 2. Now our asteroid is a decent size!
- Click Add Component in the Inspector and select Polygon Collider from the Physics 2D menu. You should see a green line outlining the asteroid. Now our asteroid can collide with our ship!

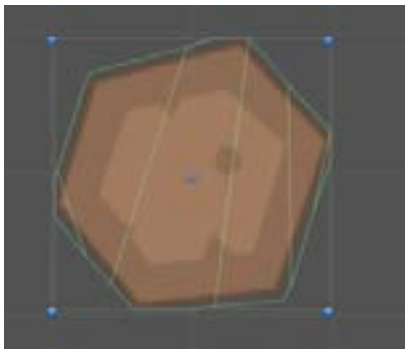

Figure 5: Asteroid selected with a green Polygon Collider 2D around it

Now that we've rescaled our MoonRock and created a Collider, we've set it up to be a building block for our game that we can use over and over and over again, otherwise known as a Prefab.

You may remember from the last activity that a Prefab is a type of GameObject that lives in the Project view and can be added to many different scenes without having to build it all over again. Each copy of a Prefab is called an 'instance'. When you create an instance of a Prefab it will create a new GameObject with all the exact settings and scripts as the Prefab. If you change the Prefab it will alter every instance of that Prefab so that it matches the Prefab. If you change an instance it will only change that one instance and not the Prefab. This can be very useful when you are working with many copies of a similar GameObject and you want to change them all or if you want to have a number of GameObjects that share the same basic properties but have a few individual differences.

Let's turn the asteroid on the screen into a Prefab like the **Player** prefab we created in Unity Lander 1.

- Go back to the **Assets** folder at the top level of the folder tree in the Project window.
- Drag the **MoonRock** GameObject from the Hierarchy into the **Prefabs** folder. The **MoonRock** GameObject in your Hierarchy is now the first instance of your Prefab in the Hierarchy!

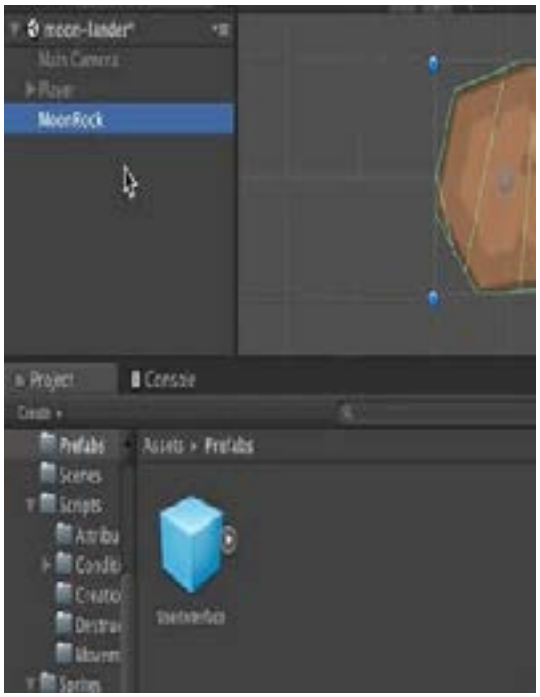

Figure 6: Dragging the MoonRock GameObject from the Hierarchy to the Prefab folder

Select the **MoonRock** Prefab in the Project window and it will appear in the Inspector as a Prefab

Scenery

Drag a **MoonRock** Prefab onto the Hierarchy to create a new instance of MoonRock. This new instance should be named “MoonRock (1)”. You won’t see the new MoonRock right away in the Scene window because it will appear on top of the first MoonRock.

Click on the MoonRock and move it to one side. You should see the exact same MoonRock right underneath.

You can adjust the size, color and position of the Prefab in the Transform and Sprite Renderer components.

- Move one **MoonRock** somewhere to the left side of your screen using either the Move tool (the crossed arrow located on the upper right side of the window) or by changing the numbers next to the X and Y in the Position area of the Transform component.
- After you move one **MoonRock**, move the other so that it is positioned next to it.
- Try changing the color by clicking on the box next to Color area in the **Sprite Renderer** component (underneath the Transform component) and select a new color from the color picker.
- Then try changing the scale of one **MoonRock** by adjusting the numbers next to the X and Y in the Scale section of the Transform area.

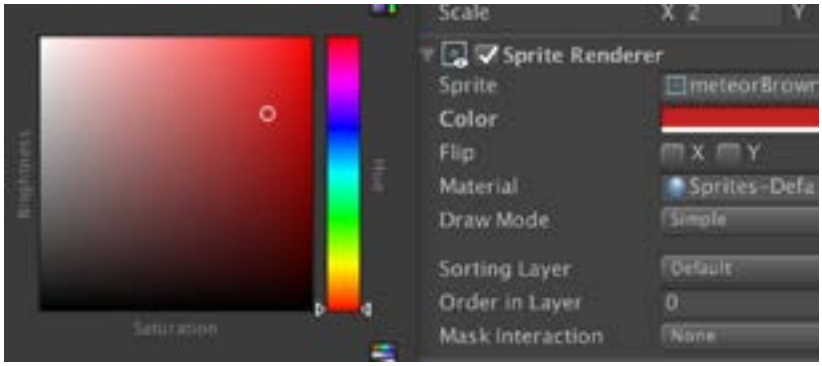

Figure 7: Color picker open next to the sprite renderer component

Now that you’ve gotten some practice experimenting with position, scale and color, drag a new **MoonRock** Prefab from the Project window into the Hierarchy to create another instance. This should be our third MoonRock (it will have the name “MoonRock (2).”) Move this MoonRock somewhere new on your screen.

Create 5 - 7 more instances of **MoonRock** in your game. You can make them look similar or you can make each instance look unique. Experiment to learn how various decisions impact the look of the **MoonRock**.

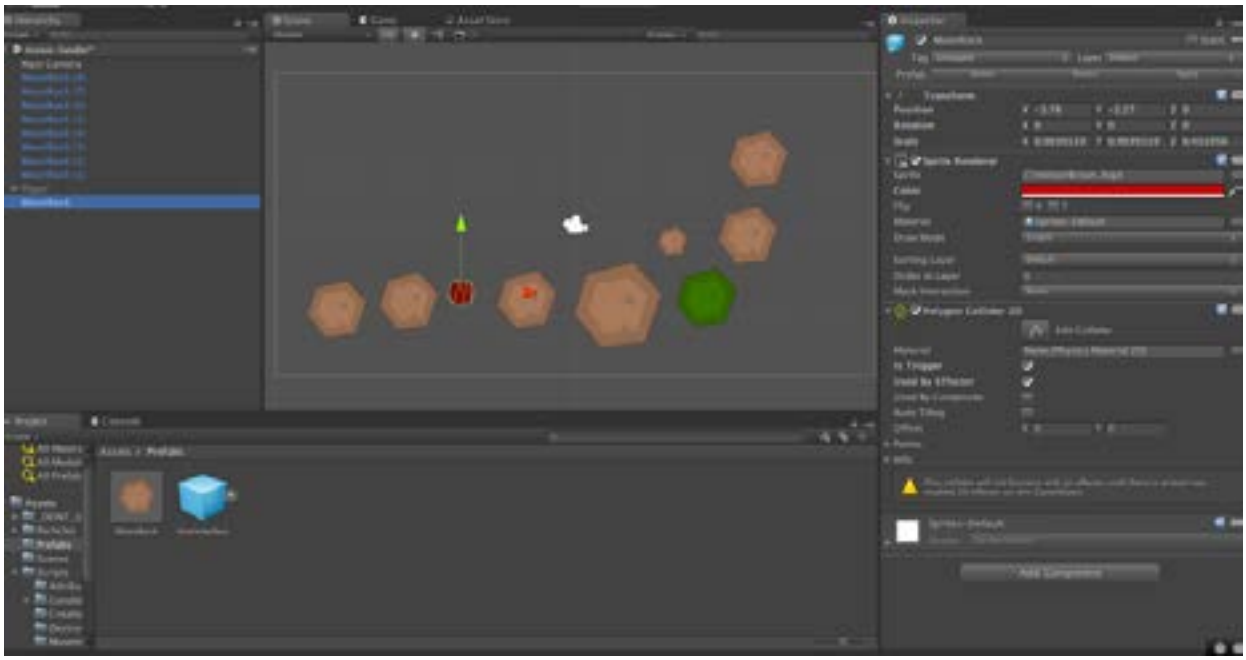

Figure 8: Screen with several instances of **MoonRock** *spread out across the scene*

Now let's create a parent folder for all of the Moon Rocks:

- Create a new empty GameObject by going to the Create menu in the Hierarchy and selecting "Create Empty".
- Change the name of the new empty GameObject to "Scenery" the same way we renamed **MoonRock**.
- In the Hierarchy select all of the **MoonRock** instances and drag them onto the **Scenery** object. This makes the **Scenery** GameObject the parent to the **MoonRock** GameObjects.

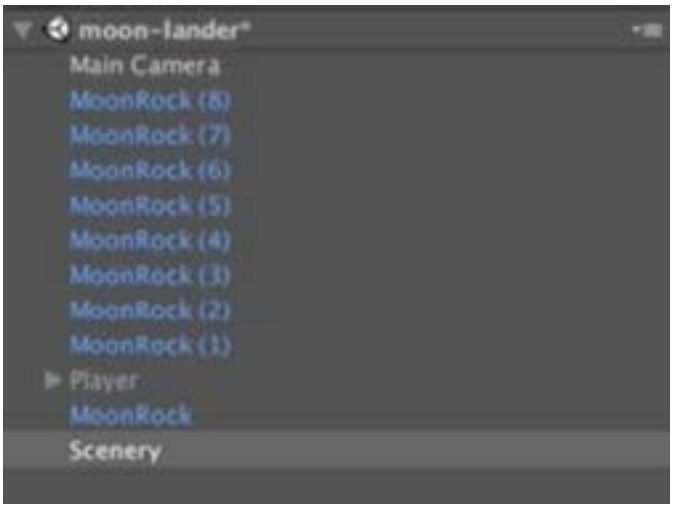

Figure 9: Dragging MoonRock onto Scenery

Not only does this make it so we can open or close this list of **MoonRock** GameObjects, but all the **Scenery** children have the properties of the **Scenery** parents, including their location. For example, if you select the Parent GameObject “Scenery” and move it, you will see all the MoonRock children objects will move as well.

For the next few steps we’re not going to need to see our scenery, and in fact, it might get in the way. So let’s deactivate it to make it invisible.

Make sure Scenery is selected in the Hierarchy. Go to the Inspector and click the check mark in the upper left corner of the Inspector. The Scenery object (along with all of the MoonRock objects should disappear).

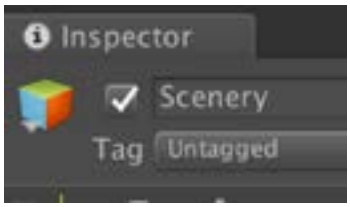

Figure 10: Active checkmark on the Inspector for the Scenery GameObject

Bring Back the Player Prefab

We've created obstacles for our game but we don't have a player character anymore. We need to take the **Player** Prefab we created in Unity Lander 1 and add it back into our Hierarchy. This is as easy as it was to create our **MoonRock** PreFab.

- Find the **Prefab** folder in the Project window.
- Click and drag the **Player** Prefab into the Hierarchy.

Figure 11: Dragging the **Player** *PreFab from the* **Prefab** *folder into the Hierarchy*

Once you've added your Player character reactivate your Main Camera and try entering Play Mode to make sure your ship still flies!

Starting and Ending Platforms

Now we'll add starting and ending platforms to our game so that our ship has a place to start as well as a destination.

- In the Project window, find the **Sprites** folder again and go to the **Blocks** folder inside of that.
- Choose a different **Material** folder than you chose for the landers double-click that folder in the Project window.

Figure 12: Blocks folder, containing three folders called Glass, Metal Elements and Stone

- Choose a wide flat block from the material of your choice and drag it into the Hierarchy or the Scene Window.
- Find it in the Hierarchy and rename this GameObject "Start"
- With **Start** in focus, click on Add Component in the Inspector.
- Select Box Collider from inside of Physics 2D.
- Make sure Start is a platform your ship can rest on. If you need to, make it wider by increasing the X Scale in Transform.

Figure 13: A wide flat block

- Select the Move Tool and place the Start block somewhere on the screen where you would like Player to begin the game.
- Move **Player** so that it is right above Start, so that when it falls it will land squarely on start.

Now let's make **Start** a Prefab so we can create an ending block that's exactly the same.

- Select **Start** from the Hierarchy and drag it into the **Prefab** folder in the Project window.

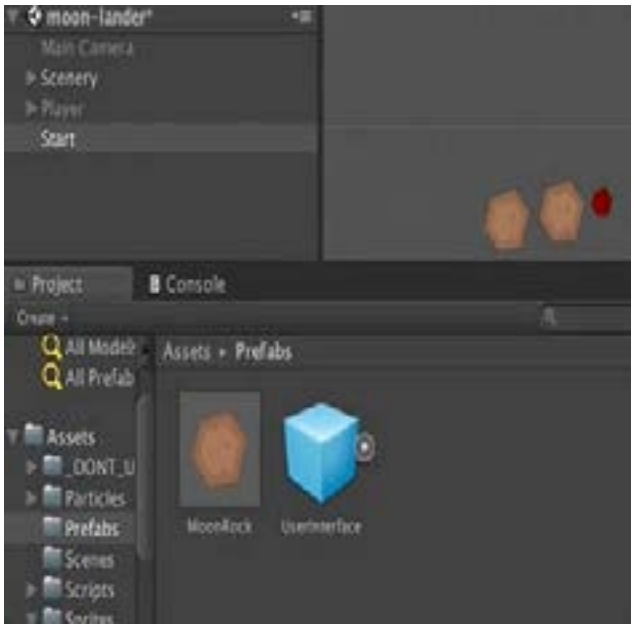

Figure 14: Dragging the Start GameObject from the Hierarchy into the Prefabs folder

- Then select the **Start** Prefab that is in the Prefab folder and drag it back into the Hierarchy to create a new instance of it.

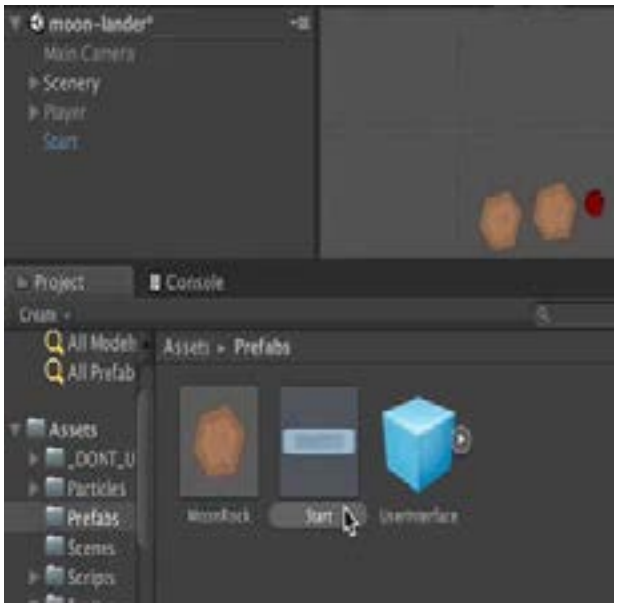

Figure 15: Dragging the start Prefab back into the Hierarchy

- Rename this instance “End.”
- Select the Move Tool and drag End somewhere on the other side of the screen from Start so your Player will have to move across the screen in order in order to win the game.

Making the Ship Explode

Right now we have a game with an objective, but the obstacles don't do anything but stop the **Player** from moving through them. Let's make our asteroids cause our **Player** to explode if it runs into them. While we want the landers (**LeftLander** and **RightLander**) to be safe, we want the ship to explode if we crash face-first into an asteroid, forcing the player to start over.

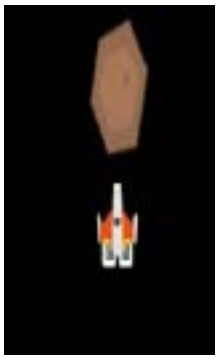

Figure 16: Ship running into an Asteroid and then exploding

- Go to the **Scripts** folder, then go into the **Conditions** folder.
- From there, go into the **Actions** folder and find the **DestroyAction** script. This script causes GameObjects to become destroyed when it is triggered by some condition.
- Select the **Player** GameObject (the parent one), to make sure it is in focus.
- Drag the **DestroyAction** script onto the Inspector (you might have to scroll to the bottom of the Inspector window). Alternatively, you can also select Add Component from the Inspector, select Scripts and then choose the **DestroyAction** script from that menu.
- Look at the **Destroy Action** section in the Inspector and change the Target to "This Object." You want your ship to explode when it crashes, not to eat the asteroid.

Next we want to create an explosion effect when our ship is destroyed. Fortunately we've already got an explosion animation for us to use.

- Click the little circle next to the words None (GameObject . . .). This brings up the Object Picker.

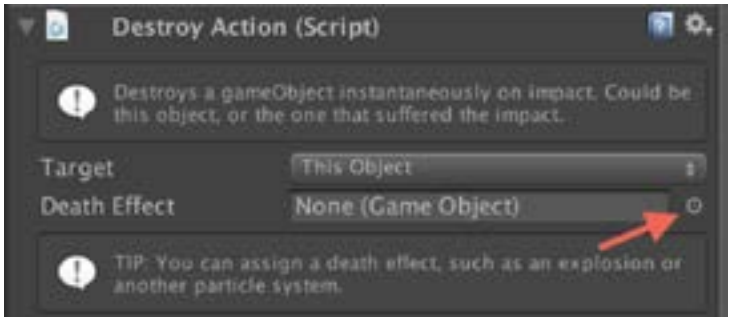

Figure 17: **DestroyAction** *script with an arrow pointing to the Object Picker button*

- There are two tabs on the Object Picker: one for objects that are currently in your Scene's Hierarchy (called "Scene") and another that shows you all the available objects in your Assets folder (called "Assets"). Make sure the Assets tab is selected on the Object Picker and type the word "spaceship" into the search area.
- Double click on **SpaceshipExplosion** from the Asset Picker window. It should appear in **Death Effect** area of the Inspector.

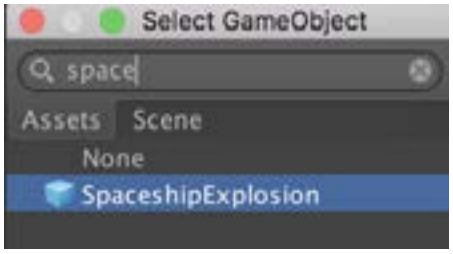

Figure 18: Object Picker with **SpaceshipExplosion** *selected*

Figure 19: **DestroyAction** *script with* **Spaceship Explosion** *selected as the death effect*

Now when our ship is destroyed it will trigger the **SpaceshipExplosion** animation.

Go back up to **Conditions** folder in the Project window and find the Script called **ConditionCollision**. This script creates a condition that triggers other scripts once a collision takes place. It allows us to trigger actions like destroying a block, modifying its size, adding to a Score, and displaying a User Interface message.

We will use this script to make our ship trigger the **DestroyAction** script when Ship's Collider is activated (i.e. when it runs face-first into an asteroid). Without **ConditionCollision** our ship would simply bounce off our asteroids, which is not what we want it to do. If you are familiar with Scratch or programming you can think of **ConditionCollision** as an "If statement" that asks if the attached GameObject has collided with anything.

- Select the **Ship** child GameObject in the Hierarchy so that it is in focus (you only want the ship selected, not the landers).
- Drag the **ConditionCollision** script onto the bottom of the Inspector.
- In the Condition Collision (Script) section, click the + icon under where it says **Gameplay Actions**

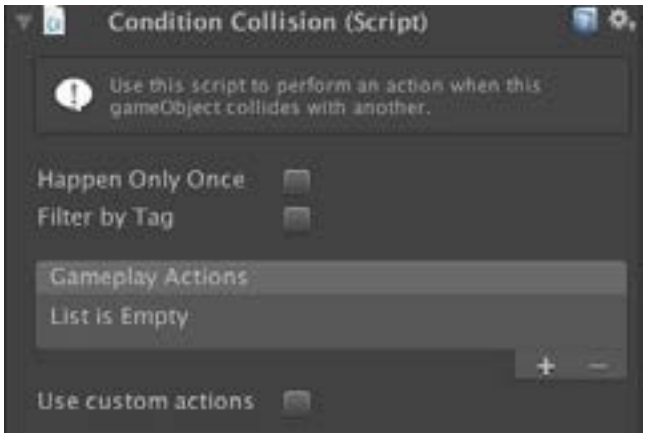

Figure 20: Condition Collision script in the Inspector

- Then click the circle next to the “None (Action)” line to launch the Object Picker.
- Select the **Player** GameObject from the Scene tab of the Asset Picker, then close the Asset Picker.

Reactivate your scenery GameObject and make sure none of your asteroids are on top of your ship, your Start/End platforms, or on top of your Player.

Now reactivate your camera and try playing your game. If you run straight into an asteroid you will explode, but if the landers touch it you’ll be fine.

Add Game Over Text to the User Interface

We have the basic parts of a game, but let's add some messages so we know when the game is over.

- Make sure you have exited Play Mode.
- Then in the Prefabs folder the Prefab that comes standard with the Playground project called **UserInterface**.
- Drag it from the Prefab folder into the Hierarchy.
- You should see a new GameObject in the Hierarchy with a parent and a few children (You might have to expand the GameObject by clicking the arrow button to see the children). You might notice that if you enter Play Mode, you'll see some text that says "3 Health" and "Score 0". This is just default text from the **UserInterface** Prefab. If you do, find the the child object of **UserInterface** called **StatsPanel** and deactivate it.

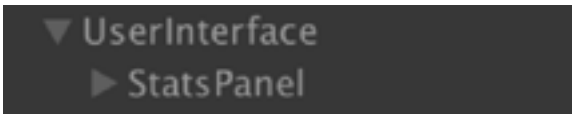

Figure 21: **UserInterface** *GameObject as a parent of the* **StatsPanel** *GameObject*

Now let's set it up so that we activate the Game Over screen at the same time we trigger our spaceship explosion.

- Select Ship in the Hierarchy so that it is in focus.
- Go to the ConditionCollision component that appeared when we added our script.
- Click the checkbox that says "Use Custom Actions."

Figure 22: Checked Use Custom Actions box

- Now click the + arrow at the bottom right of the custom action box. It looks like this:

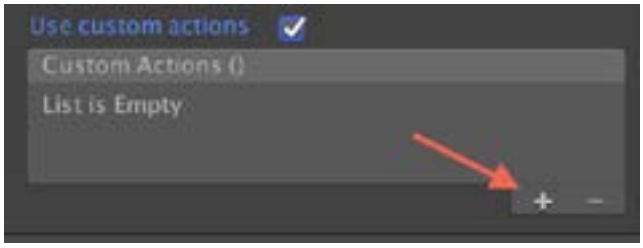

Figure 23: The + on the Custom Actions section

- Make sure “Runtime Only” is selected from the drop down menu on the left.
- Select the circle just to the right of the words “None (object)” and another Object Picker will appear.
- Select **UserInterface** from the Scene tab of the Object Picker.

Figure 24: **UserInterface** *in the Asset Picker*

Now there will be a drop down box to the right that says No Function.

- Click on it and select **UIScript** and from that menu
- Then select “GameOver (int).”

Now when you run into an asteroid and explode it will also say “Game Over.” Go into Play Mode and try it out. Once you disable Play Mode we will add our Win Text as well.

Add Win Text to the User Interface

Now that the game tells us when we've lost, we also want it to tell us when we win. To add our win text:

- Select the End GameObject from the Hierarchy. It already has a Collider on it, so we just want to make sure when it has a collision that some win text will appear.

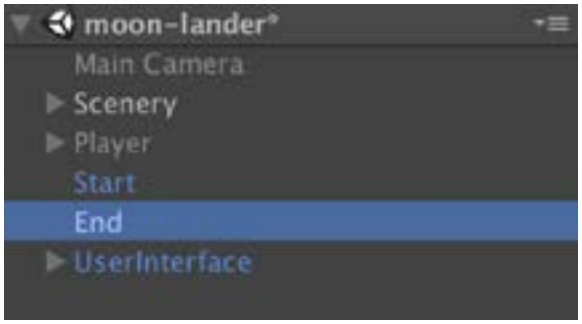

Figure 25: End selected in Hierarchy

- With End in focus click "Add Component"
- Go to Scripts and select **ConditionCollision**. This is the same script we added to our **Ship**.
- Again click the checkmark to use custom actions.
- Click the plus sign on the custom action area.
- Make sure runtime only is still selected and click the little circle next to None (Object).
- Select **UserInterface** from the Object Picker.
- On the right hand side, where it says No Function, click the drop down menu.
- Select **UIScript** and then **Game Won (int)**. This will make it so that when you touch the End platform it will say "Player 1 wins!"

Figure 26: The Game in Play Mode, it says "Player 1 Wins!" on the screen

Enter Play Mode and see if you can win.

Testing, Debugging and Background Image.

We've almost finished making the game. First we need to do some debugging. Debugging is when we look for errors in the game, such as the ship not exploding when it hits an asteroid.

Try playing your game and see if you need to do anything to improve the it. Don't just look for bugs, also think about the balance of the game. Is it too easy? Too hard? Do you need to add more MoonRocks or change the gravity level? Make your game feel right for you.

As you test your game make sure that everything is working properly and spend some time debugging your game. Try playing it a few times and make sure everything happens the way you want it to.

Here's a summary of how the game should be set up in order to work properly. You you should have the following GameObjects in your Hierarchy:

- **Ship**
- **LeftLander**
- **RightLander**
- **Start**
- **End**
- **Scenery**
- **User Interface**

Figure 27: GameObjects that should be in Hierarchy which include the Main Camera, the parent GameObject with children **Ship**, **LeftLander** *and* **RightLander**, *as well as the GameObjects* **Start, End** *and the parent GameObjects* **Scenery** *and* **UserInterface**.

- **Ship**, **LeftLander**, **RightLander**, **Start**, **End** and each **MoonRock** should have a Collider of some kind.
- You should have a **DestroyAction** script attached to **Player**.
- You should have a **ConditionCollision** script attached to the **Ship** GameObject that points to **Player(DestroyAction)** and also has a custom action selected that runs the UserInterface's GameOver message.
- The **End** platform should have a **ConditionCollision** script also that points to the

UserInterface's GameWon message.

As a reminder, here are our recommended settings for the movement scripts on the **Player** GameObject. Don't feel like you have to use our settings. Give your game the feel that you want it to have:

- Mass is 1
- Angular Drag is 2 (If you are having a really hard time steering you can try turning this to 4 for easy mode)
- Linear Drag is 2
- Gravity to .5
- Push with Button Push Strength is 10
- Rotate with Arrows speed is 2

As a finishing touch you can add a background to your game if want more than a plain color. There are a lot of great space photos from NASA you can use.

- Search for Galaxy, Star or anything you want for your space background.
- Find a large image and download it.
- Drag and drop the file into the Unity Project Window to add it to your game as an asset, and move it to the Sprites folder. (You can also import the file by going to the Assets Menu, selecting import New Asset and selecting your space picture. Make sure you put it in your Sprites folder to stay organized.)
- Then, drag your background image from your Sprites folder into the Hierarchy and you'll see your background in the Scene View with the rest of your game.
- Set the Background image's layer (in the Sprite Renderer component) to -1 and it will appear behind everything else in your game.

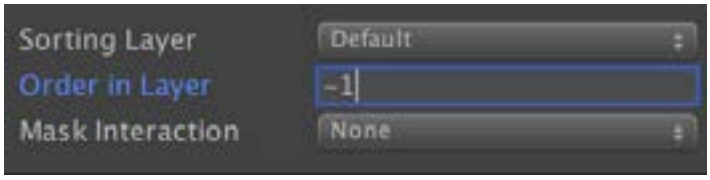

Figure 28: Order in Layer with a value of -1

Building Your Game

Congrats, you finished making a working Moon Lander scene! Now make sure to save it.

- Go to File and select Build Settings.

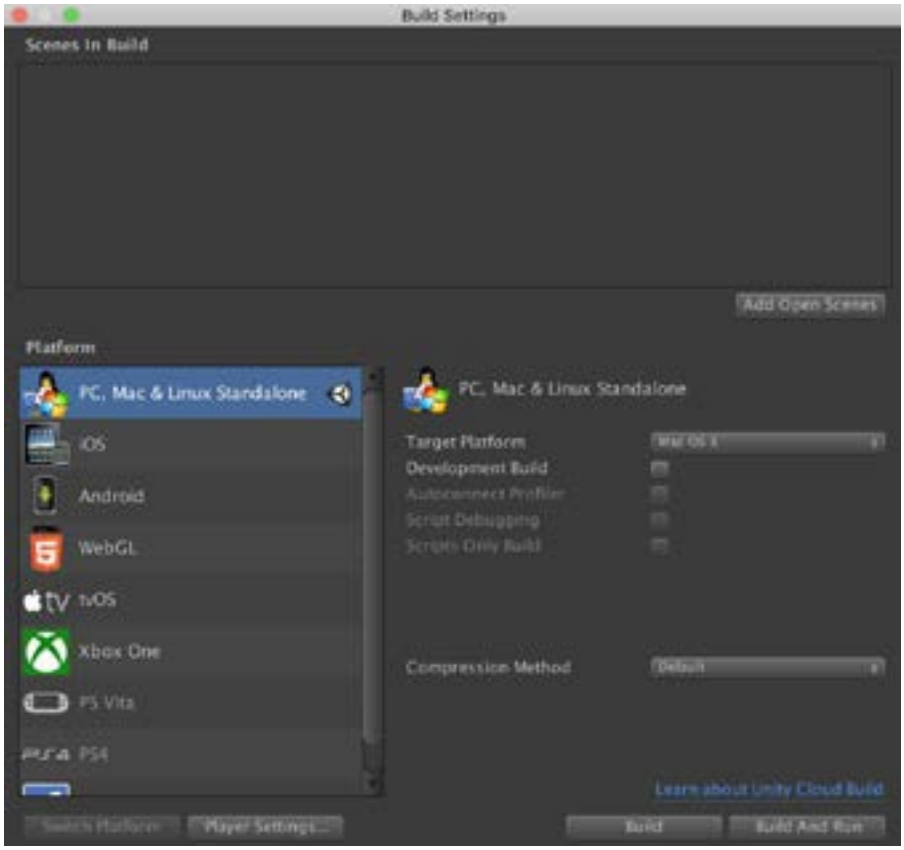

Figure 28: Build settings set to PC, Mac and Linux Standalone

- Make sure you have the right kind of build settings selected. To play your game right now, or share it with a teacher, you'll want the Windows or Mac build settings. If you want your game to be playable in a web Browser (which is required for you to submit a game to the Games for Change student challenge) you'll need to select the WebGL build settings. You can tell what your current build target is because it has a Unity logo next to it in the "Platform" list pictured.

Facilitator Note:

You may not be able to play a game that is meant for the web directly from your computer. You may need to host your files on a web server in order for a web Unity game to work properly. For more details about building Unity games for the web see the Unity Manual.

- If you want to change your Build Target, select a new one from the list of Platforms. Then click the Switch Platform button to switch.
- Make sure you also press the Add Open Scenes button as well.
- Once you have the correct build settings selected, hit “Build” and Unity will start making your game. This may take a long time depending on your computer’s speed and will take up most of your computer’s processing power so make sure no one needs to use your computer for while.
- Make sure to name your build file and to put it in a folder where you can find it.

Try playing your game. Congratulations!

Want to host your game online so that others can play it? You need to make sure you have the WebGL build target selected and follow the steps from this video tutorial.

The Playground Project

The Mouse Moon Lander package was created with assets from the Unity Playground Project. The Playground Project is a free collection of sprites, scripts and more that are premade so you can get started making games in Unity without knowing any code.

If you want to use more of those assets, download the Playground Project from Github by clicking the Clone or download button and selecting Download ZIP.

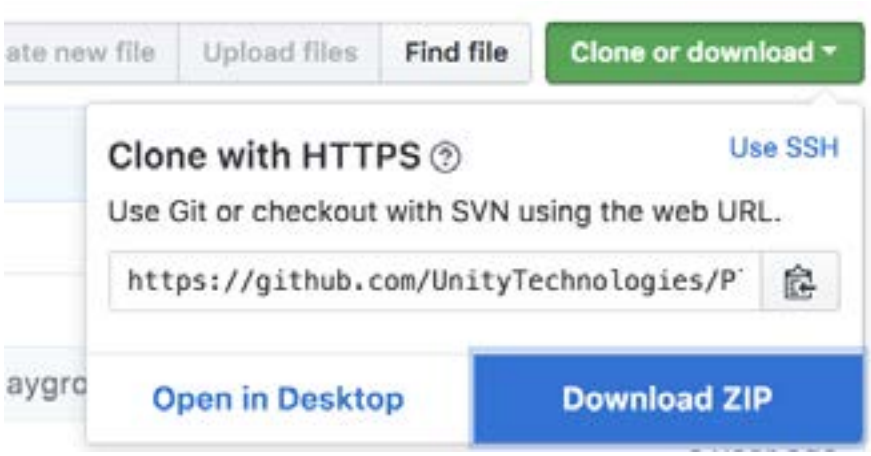

Figure 30: Clone or Download Selected and Download ZIP selected

This will download the Playground Project as a .zip compressed file, which you will need to unzip in order to use.

- On a Mac you can double-click on the zip file, which will create a new folder with the contents of the .zip file.
- On a PC (if you have Windows 7 and up) press and hold (or right-click) the folder, select **Extract All**, and follow the instructions.

Once you have unzipped the Playground Project you can open it by

- Going to Unity and selecting Open Project
- Double-click on the **PlaygroundProject-master** folder
- Select the folder called **PlaygroundProject** and click the open button.

There are many more scripts you can experiment with in the Playground Project like the **MoveWithArrows** script that allows you to control a player's movement with the four arrow keys, each key moving the player in a different direction. If you want to experiment with this movement script, bring the ship into focus and deselect the existing script components in the Inspector to temporarily disable them.

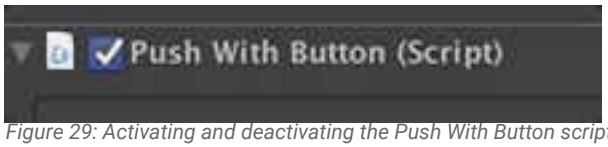

Figure 29: Activating and deactivating the Push With Button script by clicking the checkmark

The scripts PushWithButton and RotateWithArrows can be removed by clicking the small gear icon in the upper right corner of the Script's component box in the Inspector and selecting Remove Component.

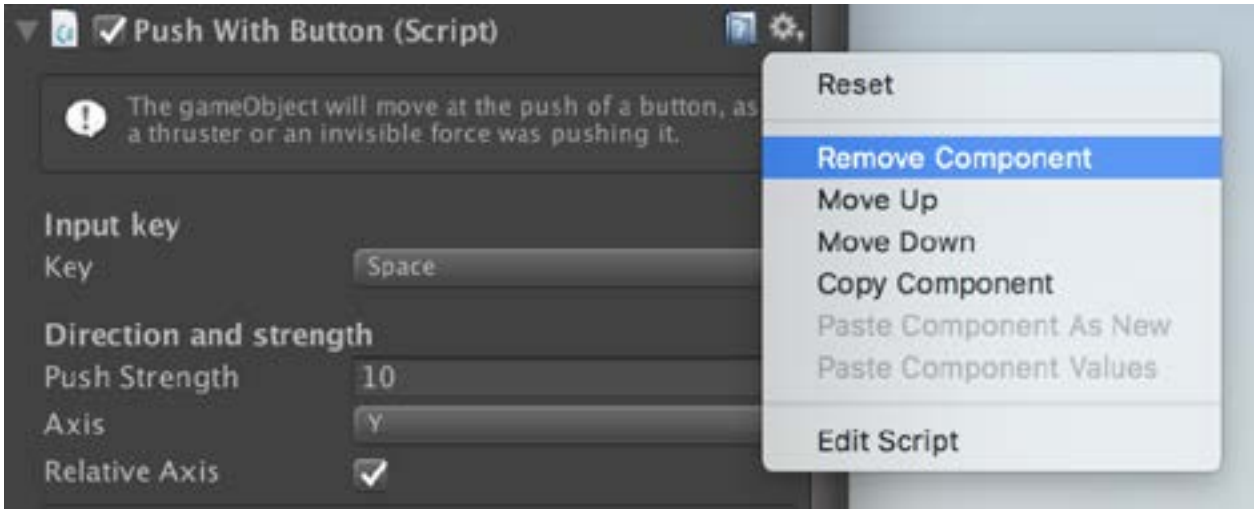

Figure 30: Remove Component selected on the Push With Button Script]

Continue to play with Unity and don't be afraid to make mistakes. Making mistakes is how you learn what you can and can't do. Most importantly, have fun!

Feedback

We'd like to hear from you! Please take 5 minutes to fill out our survey!