**Unity Tips & Tricks**

By Mike Riley

Table of Contents

[Purpose of this Document 5](#_Toc93000906)

[Unity Interface Tips 7](#_Toc93000907)

[Setting Game Window to be Side-by-Side with Scene Window 7](#_Toc93000908)

[Setting the IDE to use for C# Scripts 7](#_Toc93000909)

[Extensions to use for Visual Studio Code 7](#_Toc93000910)

[Moving between 2D and 3D Project Views/Modes 7](#_Toc93000911)

[Adding Background Image to Fill Canvas 7](#_Toc93000912)

[Adding Scenes to the Project 8](#_Toc93000913)

[Making Canvas Scale with the Screen Size 8](#_Toc93000914)

[Anchoring Screen Elements Relative to the Screen Size 10](#_Toc93000915)

[Changing the Size of a Sprite Image Equally in all Dimensions by Clicking & Dragging the Mouse 11](#_Toc93000916)

[Aligning Sprite Images Easily 11](#_Toc93000917)

[Parent, Child & Animation 11](#_Toc93000918)

[Useful Web Pages for Use with Unity 12](#_Toc93000919)

[GitHub.COM – Shared Project Database 12](#_Toc93000920)

[DoFont.COM – Fonts for use with Your Projects 12](#_Toc93000921)

[OpenGameArt.ORG – Downloadable 2D & 3D Art, Music, Textures, and Sound Effects 12](#_Toc93000922)

[Freesound.ORG – Website with Free Sound Files for Download 12](#_Toc93000923)

[App.Diagram.Net – Online Flowchart Drawing Program 12](#_Toc93000924)

[Adobe Color Wheel 12](#_Toc93000925)

[Share My Game 12](#_Toc93000926)

[RenderDoc – Graphics Debugger 13](#_Toc93000927)

[Unity Answers Discussion Group 13](#_Toc93000928)

[Las Vegas Unity User Group 13](#_Toc93000929)

[Mirror Networking 13](#_Toc93000930)

[Game Programming Patterns 13](#_Toc93000931)

[Krita – Free Paint Program 13](#_Toc93000932)

[2D Game Art Assets 13](#_Toc93000933)

[Webinars & YouTube Instruction Videos 14](#_Toc93000934)

[Text Mesh Pro – Using Fallback Font Settings 14](#_Toc93000935)

[Useful Unity Manual Pages 15](#_Toc93000936)

[Order of Execution for Event Functions 15](#_Toc93000937)

[Useful Packages from Unity 16](#_Toc93000938)

[Device Simulator 16](#_Toc93000939)

[Useful Assets from the Unity Asset Store 17](#_Toc93000940)

[Text Mesh Pro – Comes with Unity 17](#_Toc93000941)

[Agora Video SDK for Unity – Text & Video Chat Plug-in 17](#_Toc93000942)

[Generally Useful Code Tricks 18](#_Toc93000943)

[Exit Game in Unity Editor 18](#_Toc93000944)

[Laser Defender from Udemy Complete Unity Game Developer 2D Course 19](#_Toc93000945)

[Trigger Sound Effects (Explosions, dropping a Bomb, Firing a Laser, etc.) 19](#_Toc93000946)

[Delay for Loading a Scene 19](#_Toc93000947)

[Music Player with Singleton (Lesson #114) 19](#_Toc93000948)

[Spinning Effect on a Sprite 20](#_Toc93000949)

[Glitch Garden from Udemy Complete Unity Game Developer 2D Course 21](#_Toc93000950)

[Startup Audio for Loading Game 21](#_Toc93000951)

[Setting Canvas to Match World Units Scaling and Scaling Camera to Match 21](#_Toc93000952)

[Placing Sprite Images (Trees) on Canvas Easily 21](#_Toc93000953)

[Slicing a Sprite Sheet for Animation 22](#_Toc93000954)

[Adding Animated Sprite to Game 23](#_Toc93000955)

[Playing with Animation Playback 23](#_Toc93000956)

[Basic Animation Transition 24](#_Toc93000957)

[Moving a Sprite Animation using Transforms 24](#_Toc93000958)

[Calling Code Functions using Animation Events 24](#_Toc93000959)

[Keyframe Animations 25](#_Toc93000960)

[Performing an Animation using a Single Image 26](#_Toc93000961)

[Spawning Animations at Random Intervals 26](#_Toc93000962)

[Parent & Child Animation 27](#_Toc93000963)

[Collision Detection in 2D 27](#_Toc93000964)

[Combining a Rigid Body Component with a Collider 27](#_Toc93000965)

[Triggered Particle Video Effects (VFX) for Explosions 28](#_Toc93000966)

[Mouse Click Detection Using Collider and OnMouseDown() 29](#_Toc93000967)

[Determining Where a Mouse Click Happened using World Coordinates 29](#_Toc93000968)

[Spawning a Defender via Instantiate and Storing the new Object 29](#_Toc93000969)

[Creating a Row of Buttons to Select Defenders to Spawn 30](#_Toc93000970)

[Set and Place Selected Defender 31](#_Toc93000971)

[Complete C# Unity Game Developer 2D (2nd Version) Notes 32](#_Toc93000972)

[New (and some Old) User Interface Controls in Unity 2021 32](#_Toc93000973)

[The Mouse & QWERTY Control Keys for Objects 32](#_Toc93000974)

[Additional Shortcut Keys for use on Objects 33](#_Toc93000975)

[Duplicating an Object 33](#_Toc93000976)

[Adding, Disabling, or Removing a Component from Multiple Objects at Once 33](#_Toc93000977)

[Rotating an Object using Code in a Script 33](#_Toc93000978)

[Simple Rotation 33](#_Toc93000979)

[Moving Rotation 33](#_Toc93000980)

[Camera Tricks 33](#_Toc93000981)

[Making a Camera Follow an Object using a Script 33](#_Toc93000982)

[Using Tags to Identify Objects 34](#_Toc93000983)

[Using GetComponent() to Change Values on Components of a GameObject 35](#_Toc93000984)

[Snow Boarder Game from Complete C# Unity Game Developer 2D 36](#_Toc93000985)

[Creating a Sprite Shape for the Snow Surface 36](#_Toc93000986)

[Creating a Custom Sprite Shape Profile for a Sprite Shape 36](#_Toc93000987)

[Creating More Control Points for Modifying the Sprite Shape 36](#_Toc93000988)

[Deleting Control Points for a Sprite Shape 36](#_Toc93000989)

[Changing the Spline Shape Around a Control Point for a Sprite Shape 36](#_Toc93000990)

[Adjusting the Collider Offset from the Sprite Shape 36](#_Toc93000991)

[Adding a Sprite Image as a Texture to a Sprite Shape 36](#_Toc93000992)

[Adjusting the Height of a Sprite Shape Texture Image 37](#_Toc93000993)

[Creating Sprites with Collider 2D and Rigid Body 2D Components Pre-Installed 37](#_Toc93000994)

# Purpose of this Document

This document is to help explain how to best use features of Unity from the Udemy courses on Unity. So it will include references to specific lessons and lectures. Later I will be adding references to other documentation, including web pages and videos that I have also found useful for Unity.

The hope is that this will provide a quick way to look up how I did something for a particular project, so it can be used in future projects.

# Unity Interface Tips

## Setting Game Window to be Side-by-Side with Scene Window

Left-click on the Game tab and drag it to the right past the Scene window and drop it once it creates a separate Game Window. You can then drag the middle bar to make it the size you want for both the Scene and Game windows.

## Setting the IDE to use for C# Scripts

You can set this in the Unity preferences under the Edit 🡪 Preferences menu dialog box. Select *External Tools* and you will see a drop-down list for setting the *External Script Editor*. If already set earlier, it will show what is currently set and when clicked will show a list of known programs. You can also browse to editors you would like to use that are not known to Unity.

In the example for using Visual Studio Code, it also shows a line called External Script Editor Args, which is set to “$(ProjectPath)” -g “$(File)”:$(Line):$(Column), but for Visual Studio 2019 this line is not shown. So not sure if this is simply due to the early version of Unity 2021.1 being used, or if this line only shows for Visual Studio Code.

### Extensions to use for Visual Studio Code

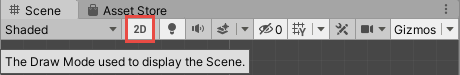
You need to install at the minimum the *C#* by Microsoft and *Unity Code Snippets* by Kleber Silva extensions for Visual Studio Code. You can view the installed extensions by selecting the button in the left panel that looks like four boxes with the top right box pulled away from the other three boxes.

Enter *C#* or *Unity* in the search box at the top to bring up a list of matching extensions and choose the ones you want from it. Once you select an extension you can click the *Install* button to install it. For existing extensions you can click on them and there will be an *Uninstall* button if you ever wish to remove an extension.

## Moving between 2D and 3D Project Views/Modes

In case you accidentally create a project in the wrong mode you can change it in Unity using the settings for the editor.

You can click on the 2D button as shown to switch between 2D and 3D views:



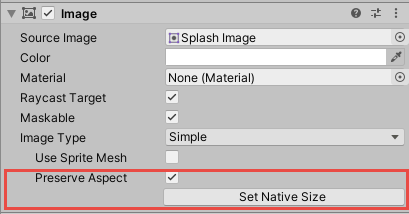
Next you must choose the Edit 🡪Project Settings 🡪 Editor from the menus. As of Unity 2019, selecting Project Settings will open a dialog box from which you can choose Editor. You then need to change Default Behavior Mode to be the 2D or 3D setting you selected with the Scene button.

## Adding Background Image to Fill Canvas

With the image already in a folder under Assets, right-click on Canvas and select UI 🡪 Image to put an image on it. Be sure to set the correct aspect ratio or resolution setting you need to use. You can one to the list by clicking the plus (+) sign as I did here to add 1080P with 1920x1080 resolution:



Click on the *Image* item in the hierarchy to show it in the Inspector. In order, select *Preserve Aspect* and then click on the *Set Native Size* button. This will make the image the same size as the canvas, assuming it is the exact same resolution.

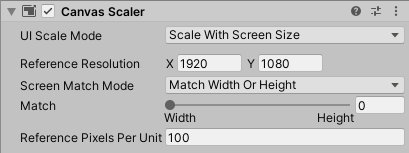


## Adding Scenes to the Project

Click on the *File 🡪 Build Settings* menu selection in the Unity menus. Clicking on the *Add Open Scenes* button will add your current open scene. You can drag any other scenes, or all of them, into the *Scenes in Build* text box. Each scene has a check box next to the name that must be selected for the scene to be enabled. You can drag them into whatever order you wish to have them kept, which will change the scene numbers.

## Making Canvas Scale with the Screen Size

Select the Canvas in the Hierarchy and in the Inspector open the **Canvas Scaler (Script)** section for viewing. Set **UI Scale Mode** to **Scale with Screen Size**. For **Reference Resolution** set it to the native resolution you want as a default to scale from. This should be a size that will hold your largest image, which is usually your background image, say 1920x1080.



Now when you look in the Scene mode you should see the blue dots for the corners matching your background image.



You can also set **Screen Match Mode** to **Match Width or Height**, **Expand**, or **Shrink**. Shrink will cut of content near the edges, which is okay if that scene has nothing important there.

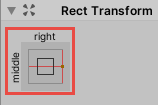
These modes can leave your canvas in a letterbox with areas on the size or above and below that are outside your canvas area. By default, these appear blue, but the convention is to make them black. You can set this color by selecting the **Main Camera** and setting the **Background Color** to black (RGB 0,0,0).

## Anchoring Screen Elements Relative to the Screen Size

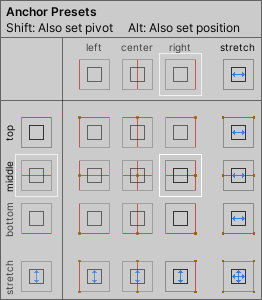
A good example of this is when you want text or buttons to be relative to the edge of the screen. Since the screen size may change on you, depending on the display device being used, this keeps important screen elements visible regardless of the aspect ratio. See **Making Canvas Scale with the Screen Size** for an example of how to set your canvas so it is independent of the screen size and aspect ratio. This is typically true if you have set the **Screen Match Mode** to **Shrink** for the Canvas.

As an example, once you have placed some text relative to your default screen size, you should test it with different aspect ratios to see how it looks in 16:9, 4:3, etc. If elements would fall outside the visible, you should consider anchoring them to be relative to some place on the screen. The places are the center, the corners, or the sides of the screen.

You select the anchor point for an item by opening the **Rect Transform** properties in the Inspector and clicking on this element in it:



Clicking on this button (for lack of a better word) will bring up a sub-window element that lets you choose to what place on the canvas you wish to anchor this object. The currently selected choice is highlighted with a white box around it. In the example below the element will be relative to the middle of the right side of the canvas:



Once you have chosen anchors for the elements you wish to have them, test with the different aspect ratios and resolutions that are defined. You can add your own to the list that Unity comes with, if you have specific ones you know you will need to support that do not fit the standard list.

## Changing the Size of a Sprite Image Equally in all Dimensions by Clicking & Dragging the Mouse

This can be easily done using the ‘R’ key, which selects the Scale button in the editor: 

This puts a vertical line from the center going up and horizontal line from the center going right, plus a white square to the upper right of center of the Sprite image. By clicking on the square and dragging the mouse left or right you can shrink or expand the Sprite image equally in all dimensions. If you use the squares at the end of either line you will only change the vertical or horizontal size of the image.

## Aligning Sprite Images Easily

Clicking on a Sprite image and dragging it into position is not an accurate way to place images that need to line up. There are two basic ways to do this.

* Set the position using the X, Y, Z numeric fields in the Inspector for the object. This gives you extremely fine control, but you may have to do some math based on the size of the Sprite image to determine the coordinates to use.
* Enable *Grid Snapping* by toggling on the button for it (). You need to have the *Tool Handle Rotation* button () set for *Global Positioning* to enable Grid Snapping. Once you have enabled Grid Snapping you can click the down arrow on the button to bring up a dialog box that allows you to set the grid size:
  + 
  + Note that although there are buttons and values for setting which axes to snap to, they do not seem to allow for anything other than X to be changed right now. Changing X sets all axes grid size to the same number. The buttons highlight when clicked on, but do not change the behavior.

## Parent, Child & Animation

As Dr. Unity pointed out to me, it is often useful to put an animation as a Child of a GameObject parent. This lets you instantiate the Parent and the Child objects easily. This is shown in lesson #282 of Glitch Garden in Complete C# Unity Game Developer 2D from Udemy using a Star Sprite image by making another Star Sprite a child of the parent. The neat thing about how the Child position works is that it is relative to the Parent. So, the position of the Parent is considered 0,0 for the Child and its relative position to the Parent will always be the same. Dragging the parent around in the Scene will cause the Child to move with it.

Another way that the two are tied together is relative size. If you resize the Parent, the Child will resize by the same relative percentage.

An Animator placed on the parent can animate the Renderers of the Parent and all the Children, but an Animator placed on a Child can only animate the Renderers of the Child. When you create an Animator in the Parent (go to the Animator window and click the Create button) you will see an Add Property button. Clicking on that button you will see properties for the Parent and a folder-like name for any Children. Opening a Child folder will expose its properties that can be added. The properties should be Transform and Sprite Renderer (assuming there is a Sprite Renderer for the Parent or Child). Children will inherit position changes from the Parent Transform changes, while their Position Transform changes will all be relative to the Parent.

# Useful Web Pages for Use with Unity

## GitHub.COM – Shared Project Database

The website is: <http://github.com>

This is a great place to store your projects, especially is using them with a development team. You can have different branches for experimenting with features in your project. It also gives you a way to go back to known good version in case you get lost going down a rabbit hole later.

It also allows you to work on the same project using multiple computers, by simply downloading the latest one onto the system you are working on at the time.

The GitHub Desktop utility gives you an easy GUI interface to use to create and maintain your projects. You can download it from the GitHub website.

## DoFont.COM – Fonts for use with Your Projects

The website is: <http://www.dafont.com>

Many fonts are free to use here. They work great with Text Mesh Pro text fields.

Once you download a font (pay attention to license requirements!) you can extract it under Assets. Using a Fonts folder with a subfolder of the font name is best. On Windows, double click on the TTF file and click on the Install button of the window that opens.

To add a font, first select the TTF file from your assets (icon is: “Aa”), then use Assets 🡪 Create 🡪Text Mesh Pro 🡪 Font Asset to create the Text Mesh Pro asset in your project from the TTF file. The icon for the is a capital “F”. Now it will appear in the list of fonts for your Text Mesh Pro object.

## OpenGameArt.ORG – Downloadable 2D & 3D Art, Music, Textures, and Sound Effects

The website is: <http://opengameart.org/user>

This is used in the Laser Defender project in the Unity 2D class by Udemy in lesson #114: Music Player With Singleton. While you may use a free download, please put an attribution about anything you may use in a game. The license terms will vary based on what the artist that posted it wants applied to it. You can use what is there, even for commercial projects. Be sure to read the FAQs about the different license types.

## Freesound.ORG – Website with Free Sound Files for Download

This is mentioned in the Laser Defender lessons #111 – Trigger Sound Effects.

## App.Diagram.Net – Online Flowchart Drawing Program

This is a useful website for drawing State Diagram flowcharts. It is completely free to use. The extension is called .drawio for the files it creates.

## Adobe Color Wheel

The website is: <http://color.adobe.com/create/color-wheel>

This is a free website from Adobe that gives you a selection of colors that go together for use in your game design. Colors are shown with the hex values, which makes them easy to use in Unity.

## Share My Game

The website is: <http://sharemygame.com>

This is a location where you can share your game for others to see and try out for free. It is only there for a limited time, but you can post updates to it at any time.

## RenderDoc – Graphics Debugger

RenderDoc is a free MIT licensed stand-alone graphics debugger that allows quick and easy single-frame capture and detailed introspection of any application using Vulkan, D3D11, OpenGL & OpenGL ES or D3D12 across Windows 7 - 10, Linux, Android, Stadia, or Nintendo Switch™.

## Unity Answers Discussion Group

The website is: <http://answers.unity.com/index.html>

This is a general discussion group where you can ask questions about Unity and search the list for possible answers. The search capability isn’t that great though. You can sign up to have questions emailed to you so you can see them and even answer them. I like to file useful ones away for future reference.

There are other links there for Forums, Unity User Groups

## Las Vegas Unity User Group

The website is: <https://www.meetup.com/Las-Vegas-Unity3D-Meetup/>

Their Facebook group is: <https://www.facebook.com/groups/lasvegasunitymeetup/>

They also have this website: <https://bitbucket.org/LasVegasUnityMeetup/meetupprojects>

## Mirror Networking

The website is: <https://mirror-networking.com/>

They provide a high level networking API for Unity that provides several different types of low level transports. Supposedly you can make it so both the server and client run from the same code base.

## Game Programming Patterns

The website is: <https://gameprogrammingpatterns.com/>

This is someone that wrote a book on patterns used to program games. The book is available for free via the website, or you can buy one.

Here is the online version: <https://gameprogrammingpatterns.com/contents.html>

## Krita – Free Paint Program

The website is: <https://krita.org/en/>

## 2D Game Art Assets

The website is: <http://www.2dgameartguru.com>

Tutorials on creating game 2D art using lots of software for creating the art. He uses Inkscape primarily, which is free. By using vector drawing tools it is easier for non-artists to create images, he says.

# Webinars & YouTube Instruction Videos

## Text Mesh Pro – Using Fallback Font Settings

<https://create.unity3d.com/thank-you-text-mesh-pro-webinar>

# Useful Unity Manual Pages

## Order of Execution for Event Functions

This is described on <http://docs.unity3d.com/Manual/ExecutionOrder.html> and has a state diagram chart that is very useful. It has references to many other methods that can be useful in the text after the chart.

# Useful Packages from Unity

## Device Simulator

There is a simulator that will simulate all sorts of phone and device screens. In the case of some Apple phones, you have a notch, so it simulates that.

Go to Windows 🡪 Package Manager and when it is up select Advanced and enable Preview, since it is currently a preview. You enable it by going to Window 🡪 General 🡪 Device Simulator.

Note: After you install it, you must restart Unity for the Device Simulator option to be shown.

<https://docs.unity3d.com/Packages/com.unity.device-simulator@2.2/manual/index.html>

# Useful Assets from the Unity Asset Store

## Text Mesh Pro – Comes with Unity

There is one in the standard API included with Unity now, so you don’t need to download it from the Asset Store. It is better to use this for any text items than the older Text class.

There are two types for this type of text object. The TextMeshPro type can be used in 3D space. The TextMeshProUGUI type is what you use on a canvas, which is the UGUI environment that it requires. You might use the TextMeshPro type for showing some health info or other text that needs to follow something within a scene, for example.

## Agora Video SDK for Unity – Text & Video Chat Plug-in

Their website is : <https://www.agora.io/en/unity/>

I set up a login here: <https://sso.agora.io/en/login/>

This was recommended to me. I still don’t understand how the licensing would work for it, but it looks very capable to allow for Teams-type video conferencing for games.

You can find it under Home -> Tools -> Video -> Agora Video SDK for Unity

# Generally Useful Code Tricks

## Exit Game in Unity Editor

To exit a game while in the Unity editor you can’t use the same method as you would for a standalone game. Note that the logic must be disabled, so it doesn’t even compile, when you build it for a standalone game. If you don’t, you will get an error that prevents it from running as a standalone program.

Here is the code I use:

public void ExitGame()

{

if (Application.isPlaying & !Application.isEditor)

Application.Quit(); // We may return from this, but the program will terminate at the end of the frame

#if true

else

UnityEditor.EditorApplication.isPlaying = false; // Handle being in the editor, but set #if to true to use it

#endif

} // ExitGame()

You simply change #if true to be #if false when you build for a standalone game, and it will run just fine. This is useful for testing that your exit game logic works while in the Unity Editor.

# Laser Defender from Udemy Complete Unity Game Developer 2D Course

## Trigger Sound Effects (Explosions, dropping a Bomb, Firing a Laser, etc.)

While most audio formats are supported, in Laser Defender he used OGG files for these effects. Here is example code used to create a bomb and play the sound. Note that the position in 3D space for the sound to play is that of the camera. Changing that position would change the volume and possibly where the sound seems to come from (assuming Surround Sound is being used).

Here is an article on what audio formats are supported by Unity: <http://support.unity.com/hc/en-us/articles/206484803-What-are-the-supported-Audio-formats-in-Unity->

private void Fire()

{

GameObject bomb = Instantiate(projectile, transform.position,

Quaternion.identity) as GameObject;

bomb.GetComponent<Rigidbody2D>().velocity = new Vector2(0, bombSpeed);

AudioSource.PlayClipAtPoint(bombDropSFX, Camera.main.transform.position,

bombSoundVolume);

} // Fire()

## Delay for Loading a Scene

You sometimes need to delay a bit before continuing, like loading the next scene. Calling a method like this will delay for the number of seconds you specify in the variable delayInSeconds:

void SomeMethod()

{

yield return StartCoroutine(WaitToLoad());

}

IEnumerator WaitToLoad()

{

yield return new WaitForSeconds(delayInSeconds);

SceneManager.LoadScene("GameOver");

} // WaitToLoad()

You can just have the WaitForSeconds() call in here to have a more general delay method and pass the number of seconds to delay as a parameter to the method and it will return to the caller once the delay period is over. This example is one that was designed to load the Game Over scene using a value you can set in the Unity Editor. Since WaitToLoad() is running in a separate thread, whatever is to be delayed **must** be in the method.

## Music Player with Singleton (Lesson #114)

* Download a music track and put it in an *Asset* folder called *Music*.
* Create an Empty Game Object (reset the Transform info, as usual) and add an Audio Source component.
  + Be sure to click on the Loop checkbox to play it in a continuous loop.
  + Add a new script called MusicPlayer to the object.
* Drag it into your Prefab folder to make it a prefab and put it in your other scenes.
* Change the Start() method in the MusicPlayer scripts to be Awake() and use code like the following to make it a Singleton object:

void Awake()

{

if (FindObjectsOfType(GetType()).Length > 1)

Destroy(gameObject); // Destroy any that come afterwards

else

DontDestroyOnLoad(gameObject); // Make the first one immortal

} // Awake()

## Spinning Effect on a Sprite

This is lesson video #118 and is implemented in the Spinner.cs script. I changed the name to specify how many times it would spin per second. You simply add this line in the Update() method to spin on the Z axis:

gameObject.transform.Rotate(0, 0, degreesOfSpinPerSecond\* Time.deltaTime);

# Glitch Garden from Udemy Complete Unity Game Developer 2D Course

## Startup Audio for Loading Game

In the Splash Screen scene do a *Create Empty* to create a new *Game Object* and add an Audio Source component to it. Recommend doing a Reset of the Transform position of the Game Object, because sound output had a position. Then simply add the audio file to the Audio Source component. Select Play On Awake checkbox to enable playing when the scene starts.

## Setting Canvas to Match World Units Scaling and Scaling Camera to Match

* We want 1 Grass Square = 1 World Unit.
* Canvas width is 1920 pixels.
* Our Squares are 160x160 pixels.
* Number of Squares = 1920/160 = 12.
* We want 12 World Units as our width.
* The canvas needs to be scaled by 12 / 1920 = 0.00625.
* Scale the canvas so 1 World Unit = 1 Square.
* Resize and align your camera.
* Repositioned canvas to (5,3), but not sure why those numbers were chosen yet.
  + Position the camera to the same location to center it on the canvas.
  + Set the camera Size to be (CanvasHeight / SquareHeight) / 2 = (1080/160) / 2 = 3.375

Create a new Sprite in the hierarchy (knob is a default one for Unity and works well). Positioning the sprit at (1,3) you should see it in the center of the leftmost column, right in the middle of the middle square.

## Placing Sprite Images (Trees) on Canvas Easily

* Add tree images to Canvas (they will appear near 0,0).
* Change *Order in Layer* to be 10 (Canvas is 0 and Defender and Attacker images will be 5, so this will cover them).
  + You normally only have a Sorting layer in the *Sprite Renderer* section for the image.
* Resize if needed after positioning the image, but be sure not to completely cover any playing area square (partial is okay).
  + If necessary, you can leave part of the image off the edge of the camera view.

Note that instead of numbers you can name your layers, which will be easier to keep track of if you have a lot of layers. You do this by clicking on Sorting Layer, which is Default when using just numbers, and selecting *Add Sorting Layer*. This opens a different panel in the Inspector area with the list of layers. *Default* is the first layer (at the top) normally. You can add new layers by clicking on the plus sign (+), which creates a new place to put the name of your layer. Layers at the top of the list are “higher” than ones lower in the list. You can move a layer name by clicking on the word layer and dragging it to a new position in the list. To delete a named layer drag it to the end of the list and click the minus sign (-). You can get out of the layer list by clicking on anything in the hierarchy.

## Slicing a Sprite Sheet for Animation

A sprite sheet is a PNG with multiple images placed on a transparent background (PNG images support transparency, but JPEG does not). To animate the sprites, you need to “slice” the images up into individual images.

* Click on the icon for the sprite sheet image you need to slice.
* In the Inspector, change *Sprite Mode* to be *Multiple* (default is *Single*).
* Click on the *Sprit Editor* button to bring up the Sprite Editor.

|  |  |
| --- | --- |
|  |  |

The Sprite Editor is a pop-up dialog box that you close once you have completed your editing. The image above shows the menu when you click on the *Slice* button. You may need to widen the dialog slightly to see all the buttons as shown.

* Click on the *Slice* button to open that sub-dialog option.
* Leaving it set for *Automatic* and click on the *Slice* button at the bottom of the sub-dialog box.
  + If using *Grid by Cell Size* figure how many cells and divide width and height to get *X* and *Y* values.
* This will put a white outline around each image in the sheet.
* Click the *Apply* button (which was greyed out before doing the slice operation) to apply the changes.
  + Clicking on the *Revert* button will revert the slicing changes.
* Click on any image in the sheet to see the name and property of any sub-image in the sheet.
  + You will see info about the sprite image and can change the *Pivot* and *Pivot Unit Mode* for that image.
  + The Sprite Sheet will display in the Assets folder with an arrow that can be clicked to show all the sprites in it. Clicking on an individual sprint image brings up the properties in the Inspector.

Table

Description automatically generated with medium confidence

## Adding Animated Sprite to Game

To add an animated sprite into the game, do the following:

* Right click in the *Hierarchy* and select *Create Empty*, changing the name to what you want to call the sprite.
  + Don’t forget to do a *Reset* for the *Transform* section in the Inspector.
* Click on *Add Component* button and add a *Sprite Renderer*.
  + For *Sprite* choose the frame you wish the sprite to start animating with (*Lizard\_Walk\_7* in the lesson example).
* Be sure the Animator and Animation windows are somewhere in your Unity UI
  + You can enable/disable these using the *Window* 🡪 *Animation* menu selection submenu.
  + In the Animation tab you can open the menu using the three dots on the far right and enable feature to be displayed, like *Frame Rate*.

Terminology for animations:

* **Animator Component** – Assigns animations to *GameObjects* through an *Animator Controller*.
* **Animator Controller** – Arrangement of animations and transitions (state machine).
* **Animation** – Specific pieces of motion images.
* **Sprite Renderer** – Displays the 2D sprite on the screen

Here are the steps to creating the animation sequence:

* In your *Assets* folder create a new folder called *Animations*. You may want to have subfolders for multiple animations.
* In your folder right-click and create an *Animation Controller* (name it for what makes sense to you).
  + Selecting the Animation Controller and clicking on the tab for the *Animator* window will show the states.
  + Name this to a name that makes sense, like *Lizard Controller*.
* Create an *Animation* by:
  + Go to your sprite sheet and clicking on the arrow to show all the frames.
  + Select all the frames you want in the animation.
  + Right-click on one of the selected frames and select Create 🡪 Animation from the pop-up menu.
  + Name it appropriately, like *Lizard Walking Anim*.
  + In the Inspector click on the *Loop Time* checkbox to make sure the animation loops when played.
  + Move this *Animation* object into your Animation folder with the Animation Controller created earlier.
* On the Game Object you added the Sprite Renderer to click on *Add Component* and add an *Animator*.
  + For *Controller* select the name of the controller you created or drag it onto the object in the Hierarchy.
* Click on the Animator tab and drag the Animation you created earlier from the Animations folder into the Animator Window, which should have *Entry*, *Any State*, and *Exit* states in it after it was created, and it gets linked with a line to *Entry* automatically.
  + The line is called a *Transition*.
  + When you click on the Play button you should see your animation working now.
    - You may need to adjust the sample rate to make it look good. Do this by selecting the Animation object in your Animations folder and adjusting it in the Animation tab window.

## Playing with Animation Playback

You can adjust the playback of an animation using the Animation tab:

* Select an Animation in the Animations folder and click on the arrow for it in the Animation tab to show each frame in sequence (You may need to zoom out to see all and have room to the right).
* If you use Ctrl-A to select all the frames, position the mouse over the last one and you will see the pointer change into a double arrow (↔).
  + Press and hold the left button and drag to the left or right and it will try to space out the frames using the frame rate that is set, so they will be kept in groups with gaps between them.
* Selecting one image allows you to drag it to a new position in the frame order.
* Select a range of frames and you can create a gap between them and the previous frames for a pause.

## Basic Animation Transition

This is used to have a sequence of animations that play in a specific order.

* Drag the first animation into the Animator board space first, so it auto-generates the transition from the *Start* state to that animation for you.
* Now drag the other animations onto the Animator board (order doesn’t matter).
  + You can move things around on the board by dragging. Position doesn’t matter. Ctrl-A resizes to fit.
* Right-click on the first animation and select *Make transition* from the pop-up menu.
  + A transition arrow appears, and you can move the mouse to the next animation and click on it to anchor.
  + Repeat for all animations in order of use.
* Clicking on a Transition arrow line brings up information about in the Inspector:
  + Open the *Settings* by clicking on the arrow next to it:
    - The *Exit Time* and *Transition Duration* settings are not needed for 2D.
      * These are used for blending 3D animations, but for 2D you want a sharp transition.
    - Set Exit Time to 1 and Transition Duration to 0 and you will see it become a sharp transition.

## Moving a Sprite Animation using Transforms

This is how you can move an animated Sprite around on the game screen.

Here is some simple code to move a sprite to the left and gives you a slider in Unity to adjust the speed for testing:

[Range(0f, 5f)]

[SerializeField]

float walkSpeed = 1f;

// Update is called once per frame

void Update()

{

transform.Translate(Vector2.left \* walkSpeed \* Time.deltaTime);

} // Updater()

Vector2 also has *right*, *up*, and *down* for moving in those directions.

## Calling Code Functions using Animation Events

This is where you can add event markers in an animation that will call a method (one per event) in your code:

A screenshot of a computer

Description automatically generated with medium confidence

This solves the problem with the Lizard where when it jumps you don’t want it moving, but when the animation for walking starts you do want it to walk. Setting an event at the start of both animations, with the lizard jump speed set to 0 and the lizard walk set to whatever you tuned it to fixes the issues in the jump animation where the legs are not moving, but it looks like it is sliding along. You can also set other events where you change the speed, like making it run, play a sound, etc. Be sure the C# script with the functions you need are added to the root Game Object.

The code looks like this:

public void SetMovementSpeed(float speed)

{

currentSpeed = speed;

} // SetMovementSpeed()

Note: Be careful with the method name. This is kept as a string in the event properties, so spelling must be correct.

## Keyframe Animations

In Glitch Garden this is done using the *Trophy Cut* image, which has the trophy cup and base separated by a small blank area. When Sliced on Automatic you get two frames. By clicking on the cup image on top you can move the border on the bottom up if you want to, as shown in the video lesson.

Here is how you add this into the scene:

* Create a new Game Object in the hierarchy by right-clicking and selecting Create Empty
  + Reset the Transform location and name it appropriately (Trophy in this example).
* Open the Sprite Sheet by clicking on the arrow next to the icon in the folder for it.
  + Drag the top image sprite onto the Game Object in the Hierarchy to make it a child
  + Drag the bottom image onto the Game Object the same way.
    - You can rename them as Top and Bottom in the hierarchy, so it is clearer what they are.
  + Be sure to set *Order in Layer* so the images are visible and position them, so the top is above the base.
* Note: To change the relative size of all the sprites in the sheet, click on the sheet icon and change *Pixels per Unit*.

Here is how you animate the Trophy using *Key Frames*:

* Select the Object with the two sprite children in the Hierarchy and in the *Animation* tab click the *Create* button.
  + Place it in the Animation folder and call it something meaningful (Trophy Bounce in this example).
* Set the number of *Samples* to be the number you want per second, usually the same as your other animations.
* In the Animation tab, click on the round red button to start recording.
  + Now every time you move any of the sprite images, it will create a new keyframe.
    - For the one at the start move something and then move it back to the starting position.
  + To set where the next keyframe will be, drag the white line to the next position you want the next one.
    - You drag it by clicking on the line where it is on the scale bar and dragging it to a new position.
  + Move the sprites for the next keyframe and repeat at the next position.
  + Once all keyframes have been recorded, click the red button again to turn off recording.
    - Note: Be sure to have a keyframe for the first and last positions in the sequence.
  + By dragging the white line across each position on the scale bar you will see what each frame will be.
* Each keyframe will have a couple of diamonds, one above the other, showing that some change was recorded.
  + Click and drag to select a pair of diamonds to highlight them.
  + You can now drag them to a new position to change the position of the keyframe in the sequence, speeding up or slowing down the time it takes to reach that position when playing.
  + Click on the arrow next to *Position* for one of the sprites and you will see the X, Y, & Z numbers.
  + If you want to change any of them, change them in the Inspector fields (useful to get exact position values to match other keyframes).
  + You can also position the white line on a particular keyframe, click the record button, and change the sprite positions by dragging them, then click record off.
  + Another option is to select the *Curve* mode of display instead of *Dopesheet* (you may need to zoom out).
    - Select any X/Y/Z value that is changing & you will see the keyframes as dots along a curving line.
    - Dragging the dots higher or lower, left or right, will change the axis values and where the keyframe happens in the sequence.
  + The other way you can move the sprite is to rotate it, which you do by using the rotation mode selection.
  + You can also change colors at key frames in addition to positions and rotation.

## Performing an Animation using a Single Image

In Glitch Garden lesson #136, he first shows how you can animate the Zucchini image using an animation controller. Later in the lesson it shows how to do this with code, but it is simple to do it with just a couple of keyframes and even more complex animations can be done this way.

Here are the steps:

* Create an *Animation Controller* and assign the sprite image to it.
* Click the button that appears in the Animation timeline window to create an animation.
  + You now have a default animation length of 1 second, but you can change that by dragging the line at the end of the time scale right or left (to increase you will need to zoom out to see more time).
* Click on the Record button and drag the image to the endpoint to get it to travel from the starting point to that endpoint.
  + You can add more key frames to have it go to more than one position over time.
* You also add rotation and color changes at your keyframe spots as well.
  + You can select the rotation mode button (next to the move mode button), left-click on the white circle, and drag the mouse to the right to increase the amount of rotation, releasing the left button to finish.
  + You can enter it as a Z value in the Inspector for Rotation. No matter if it was a positive or negative number it always rotated the same direction, but when dragging to the right it became negative and that rotated clockwise. So not 100% sure how to control the direction of rotation.

## Spawning Animations at Random Intervals

In *Delay for Loading a Scene* it shows how to create a coroutine that will do a delay before doing something, but you can also make Start() be a coroutine that does a delay while repeating some code. In this case the settable values you might pass as parameters are values you can set in the SerializeField values, while the position and rotation values are derived from where you have placed the object in the scene that the script is attached to:

[SerializeField]

float minSpawnDelay = 1f; // Minimum second before spawning an attacker

[SerializeField]

float maxSpawnDelay = 5f; // Maximum second before spawning an attacker

[SerializeField]

Attacker attackerPrefab; // Animation of attacker

bool spawn = true; // Keep spawning while true

// Start is called before the first frame update

IEnumerator Start()

{

while(spawn)

{

yield return new WaitForSeconds(Random.Range(minSpawnDelay, maxSpawnDelay));

SpawnAttacker();

} // while

} // Start()

/\*\*\*

\* Spawn an attacker.

\*\*\*/

private void SpawnAttacker()

{

Instantiate(attackerPrefab, transform.position, transform.rotation);

}

## Parent & Child Animation

Starting with lesson #146 & #147 of Complete C# Unity Game Developer 2D (archived version for Glitch Garden) it shows how to use a Parent Game Object and a Child Game Object with the animation in the Child object. The reason this can work out better is that children positions are always relative to the parent object, so just moving the parent moves any children at the same time.

In this case we use the following steps to create the Lizard animation:

* Place a Lizard prefab on the Scene hierarchy
  + Remove the Sprite Render component, leaving the Animator, Attacker script, Box Collider 2D, Rigid Body 2D, and Health script components
  + Delete the old Lizard Animation Controller from earlier and create a new one, setting it in the Animator component as the controller
* Create a child Game Object for Lizard and name it Body with the Transform position @ 0,0,0 so it is the same position as the parent object (should be the default)
  + Add a Sprite Renderer component with a sprite image (Lizard\_Walk\_4 is used here)

### Parent & Child Sprites with Collider

In the Snow Boarder lesson there are two images of the snow boarder, a top and a bottom. Once these are joined together in the Scene Editor (be sure to zero them relative to the parent object) you can set the center of the parent to match the center of the parent. Do this by following these steps:

* First click on the parent to see where the center is
  + Place the center square at the intersection of some grid lines to make it easy to know the position
* Next click one of the children images and then shift-click on the other image(s) so all are selected
* You will now see a center for the two images
* Move it to be as close as possible to the center of the parent object
  + You may need to repeat the last three steps to get it as close as possible

For the Snow Boarder image, it isn’t quite horizontal, so before adding the collider level it out using similar steps:

* Select all the child images by clicking and shift-clicking
* Enter ‘E’ to go into rotate mode and rotate the image to be as level as possible

To add the collider, in the case of Snow Boarder, you select the parent object and add the collider to it, not to one of the child images. For Snow Boarder you add a Capsule Collider and change it from Vertical to Horizontal. Different images might require a different type of collider or different settings. In this case, adjust the shape and size of the collider to match the size of the snow board in the sprite image.

## Collision Detection in 2D

To do this, you need to have some components added to your GameObject:

* <Type> Collider 2D, where <Type> can be several names:
  + *Box* is used for Glitch Garden because an exact fit is not required.
  + *Circle* where you can adjust the diameter
  + *Capsule* starts as a circle, but you can adjust 4 points on both sides and top/bottom to make it oblong.
  + *Composite* is one I haven’t tried yet
  + *Edge* allows you to start with a line with two endpoints, which you drag around the outline of your sprite image, and you can add as many points as you want any place between two points, eventually joining the two endpoints.
  + *Polygon* is a type I haven’t used yet, but I imagine lets you choose for different polygonal shapes.
  + *Tilemap* is a type I haven’t used yet.

Many of these colliders also have 3D versions without the 2D at the end of the name.

For a collider that will trigger something happening, like a projectile hitting something, be sure to check the *Is Trigger* checkbox. When the checkbox is set another sprite will pass through instead of stopping when it makes contact. You need to add a function to the script for the object to handle the trigger event when it happens. That function looks like the one below. An equivalent you can use without *Is Trigger* being set is OnCollisionEnter2D():

private void OnTriggerEnter2D(Collider2D collision)

{

var health = otherCollider.GetComponent<Health>(); // Find the Health class for it

Debug.Log("I hit: " + otherCollider.name); // Report what was hit for debugging

health.DealDamage(damage); // Cause the damage, which will destroy it if dead

}

There is an equivalent function called OnTriggerExit2D() that can be used to say when an object is leaving a collider.

For either method to be called, the object that you want to have the trigger reported must have a Rigid Body component. While it takes more computing, setting the *Collision Detection* to *Continuous* instead of the default of *Discrete* gets better detection if one or more of the objects is moving fast. Best to test to see what works best for your case.

### Combining a Rigid Body Component with a Collider

* Rigid Body 2D
  + Be sure to set the type to *Dynamic*, *Kinematic*, or *Static*. Kinematic is used for Glitch Garden.

Adding a RigidBody2D will allow for physical effects when a collision happens, like pushing with another sprite that has a RigidBody2D on it.

## Triggered Particle Video Effects (VFX) for Explosions

This is like Trigger Sound Effects (Explosions, dropping a Bomb, Firing a Laser, etc.) and you may want to do both together. You call the method to create and play the VFX from a trigger method that is typically triggered by a collision or because enough damage has been taken by an object.

private void TriggerDeathVFX()

{

if (!deathVFX)

return; // Only happens if deathVFX is not set

GameObject explosion = Instantiate(

deathVFX,

transform.position,

transform.rotation);

Destroy(explosion, durationOfVFX);

AudioSource.PlayClipAtPoint(deathSFX, Camera.main.transform.position, deathSoundVolume);

Destroy(gameObject);

} // TriggerDeathVFX()

Be sure that the VFX object is set for the correct layer so it can be seen. You probably want it to be 1 higher than the object the VFX is being played for, so it appear on top of it. To set this Open the Particle System for the VFX object in the Inspector and click on the line for Renderer (which should already be checked) to open the list of render settings. Set *Order in Layer* to whatever value is needed to make the effect visible.

Graphical user interface, text, application

Description automatically generated

## Mouse Click Detection Using Collider and OnMouseDown()

For this game we want to detect when a mouse click happens within the area of the game board where you can place defenders. To do this add a new Game Object (Create Empty and name it appropriately), reset the position, and add a Box Collider 2D component to it. To size and place the collider properly within the game, set it to a size of X=7 x Y=5 and position it at X=4 & Y=4, or however big you need for your board. Since the board is in World Units, this makes it easy to position the collider as well. The position you are setting is for the middle of the collider, so if you need to calculate it keep that in mind.

Create a script and attach it to the Game Object to handle the left mouse button clicks. You don’t need the standard methods, but need to create an OeMouseDown() method. Visual Studio will fill it in as a template method by simply typing the name. By adding a Debug.Log() call you can verify it is working easily and the code will look like this:

private void OnMouseDown()

{

Debug.Log("Mouse was clicked.");

SpawnDefender(GetSquareClicked());

} // OnMouseDown()

## Determining Where a Mouse Click Happened using World Coordinates

This is a way you can turn Game coordinates into World coordinates. I have added a way to clamp the values, so they stay on the play area of the board, and they are in the middle of a game square.

private Vector2 GetSquareClicked()

{

Vector2 clickPos = new Vector2(Input.mousePosition.x, Input.mousePosition.y);

Vector2 worldPos = Camera.main.ScreenToWorldPoint(clickPos);

worldPos.x = Mathf.Clamp(Mathf.RoundToInt(worldPos.x), 1.0f, 7.0f); // Always ensure it is in the middle of the square left to right

worldPos.y = Mathf.Clamp(Mathf.RoundToInt(worldPos.y), 1.0f, 5.0f); // Always ensure it is in the middle of the square up to down

return worldPos;

} // GetSquareClicked()

## Spawning a Defender via Instantiate and Storing the new Object

This is used to spawn a defender, but notice how it is different from how attackers were spawned. Because this is being assigned to a variable, you should add the ‘as’ keyword to give it a specific type:

[SerializeField]

GameObject defender;

private void SpawnDefender(Vector2 worldPos)

{

GameObject newDefender = Instantiate(defender, worldPos, Quaternion.identity) as GameObject;

} // SpawnDefender()

Another way to instantiate with the correct type is this:

GameObject newDefender = GameObject.Instantiate(defender, worldPos, Quaternion.identity);

Both produce an object of type GameObject at a specific location with a known rotation.

## Creating a Row of Buttons to Select Defenders to Spawn

This is an easy way to create a row of buttons that you use to select the type of defender you wish to place on the game board. Here are the basic steps:

1. Create a GameObject using *Create Empty* and name it Buttons.
2. Add a 3D Object to Buttons called a *Quad* (named Background Quad).
   1. Change the size of the Quad and its position so that it stretches across the bottom of the Game area, but not where the game will be played.
      1. I did this by setting *Scale X* = 0 and *Scale Y* = 0.9, with *Position X* = 0 and *Position Y* = -2.9.
      2. Set *Position Z* = -0.01 to make sure it is above the game board relative to the camera.
   2. Create a Materials folder in Assets in in there right-click and create a *Material* object (called Button Material).
      1. Set to a color you like (chose 0x35007E and set Alpha to 75).
      2. Change *Rendering Mode* from *Opaque* to *Transparent*.
3. Add one of the images from the Cactus Idle animation sequence in the Sprites folder as a child of Background Quad by dragging it onto it in the hierarchy (used Cactus\_Idle\_29) and rename it to Cactus.
   1. Resize as needed to fit within the square for the button (I set *Scale X* = 0.1 and *Scale Y* = 1.1).
   2. Click Add Component and Add a Box Collider 2D.
      1. Click on Edit Collider and drag the edges to fit the width and height for that button square.
4. Add the Trophy Whole image from the Sprites folder as a child of Background Quad and rename it to Trophy.
   1. Resize as needed to fit within the square for the button (I set *Scale X* = 0.125 and *Scale Y* = 1.45).
5. Add a Script called DefenderButton to both buttons at once by highlighting both Cactus and Trophy in the Hierarchy and clicking the *Add Component* button in the Inspector to create the script. This assigns it to both objects at the same time.
6. Click Add Component and Add a Box Collider 2D for both sprite images to make the clickable button area.
   1. Click on Edit Collider and drag the edges to fit the width and height for that button square.
7. For both sprite images click on the color selector and lower the R G and B values to something that lets you see what the image is, but lowers the intensity to “grey” it out (I used 100, but Rick use 41, so test it on your own).

This DefenderButton script will be handling turning a button on and off. They act like mutually exclusive buttons, so when you click on one that is greyed out it becomes fully lit and any other lit one is greyed out. I added a settable value for greying them out, so it can be controlled in the Unity Editor, since some may look better with different values. Default it to the value you used for most of your sprites in the editor and override it for any that need a different one. You can also add a Start() method to automatically grey all the buttons out when the game starts.

public class DefenderButton : MonoBehaviour

{

[SerializeField]

byte greyColorValue = 100; // Settable in the Unity Editor in case it is needed to be different for various images

private void OnMouseDown()

{

var buttons = FindObjectsOfType<DefenderButton>();

foreach (DefenderButton button in buttons)

{

button.GetComponent<SpriteRenderer>().color =

new Color32(button.greyColorValue, button.greyColorValue,

button.greyColorValue, 255); // Make button image greyed out

} // foreach()

GetComponent<SpriteRenderer>().color = Color.white; // Make button image fully visible

} // OnMouseDown()

} // class DefenderButton

## Set and Place Selected Defender

This covers how we can use the buttons to place the selected defender type, each of which has a cost in “stars”. The basic steps are as follows:

1. Create two new C# scripts in your Scripts folder: Defender and StarDisplay.
   1. Add the Defender script to the prefabs for both Cactus and Trophy.
   2. For this simply add an integer value called starCost that can be set in the Unity Editor.
   3. Add the Defender script to both the Cactus and Trophy prefabs.
2. In the DefenderButton class (see Spawning a Defender via Instantiate and Storing the new Object) add a method called SetSelectedDefender() with a parameter of type Defender.
3. In the OnMouseDown() method in the DefenderButton class (see Creating a Row of Buttons to Select Defenders to Spawn) add a line to call SetSelectedDefender() with the defenderPrefab value that was set for the button in the Unity Editor.

Here is the code for SetSelectedDefender():

Defender defender;

/\*\*\*

\* This sets the type of Defender to instantiate, based one the currently selected button.

\*\*\*/

public void SetSelectedDefender(Defender defenderToSpawn)

{

defender = defenderToSpawn;

} // SetSelectedDefender()

Here is the code for OnMouseDown() to call SetSelectedDefender():

FindObjectOfType<DefenderSpawner>().SetSelectedDefender(defenderPrefab);

**Note:** For some reason the Trophy defender is not being displayed, but it is being placed.

# Complete C# Unity Game Developer 2D (2nd Version) Notes

In 2021 they revamped the original class and switched it up with only the last part of the class being the same. It uses Unity 2021.1.4f version for the course, so what is shown in here may change by the time the LTS version shows up. When I started it in December 2021, they had 2021.2.7f1 available (2021.3 will be LTS).

## New (and some Old) User Interface Controls in Unity 2021

### The Mouse & QWERTY Control Keys for Objects

* You can use the mouse wheel to zoom in and out of the Scene window, but you can also hold the Alt key and click the right mouse button – dragging left to zoom out and right to zoom in.
* Clicking and holding your middle mouse button (the wheel in most modern mice you can drag the scene around inside the Scene window. If you click the hand in the toolbar area), or click the *Q*-key, you can use the left mouse button for this, as before, but this lets you do it without changing the mouse mode using the toolbar button.
* Note that for the following you can also hover the mouse pointer over the X, Y, or Z values for Position, Rotation, or Scale. The mouse pointer will turn into left and right arrows, and you can then click the left mouse button and drag it left or right to change that one value for the object:
  + Selecting an object in the hierarchy and hitting the *W*-key will act like clicking the Move tool button () and bring up the green and red arrows that you can drag with the mouse to move it up/down or left/right. Clicking on the lightly filled blue square at the intersection of the two lines allows for dragging the object in any direction.
  + Clicking on the Rotation tool button () or hitting the *E*-key will bring up the rotation handles around the selected object, which appear as a blue circle inside a white, plus a green and red line.
    - Clicking on the white or blue circle and dragging it lets you rotate the object in Z (not sure why two circles that do the same thing).
    - Clicking on the green line and dragging it lets you rotate the object in Y.
    - Clicking on the red line and dragging it lets you rotate the object in X.
  + Clicking on the Scale tool button () or hitting the *R*-key will bring up two lines ending in small squares that you can drag to scale the object in either X or Y dimensions. There is also a dot in the middle that if you click on it the object will scale equally in all dimensions by simply dragging the mouse around.
* Clicking on the Rect tool button () or hitting the *T*-key will select the transform mode. This is shown by putting blue dots around the selected object that can be clicked on and dragged to transform the object. The corner dots will change the object in two dimensions and only for the sides where the dot is a corner, but clicking on the line between two dots will change it in only one dimension on the side between the two dots.
* Clicking on the Transform tool button () or clicking the *Y*-key will bring up all the controls listed above for the other keys. This allows you to change anything you want without having to switch between modes, but I find it too confusing.
* Clicking on something in a folder or the Hierarchy and hitting <F2> will put it in renaming mode. You can then enter a new name for the object.

## Additional Shortcut Keys for use on Objects

## Duplicating an Object

With an object selected, hitting the *Ctrl-D* key combination will *Duplicate* the object in the exact same spot. It is best to have the Move button active (or press the *M*-key after you press *Ctrl-D*) so you can move the new copy to a different location in the scene, or else you might forget that you have two identical objects in the same spot.

## Adding, Disabling, or Removing a Component from Multiple Objects at Once

In the hierarchy you can select multiple objects either by holding the Ctrl key down and clicking on each one, or if in order click on the first one, hold the Shift key down, and click on the last one. In the Inspector you can click the Add Component button (or drag a script over) to add to all the objects. Similarly, if they have a component in common you can delete or disable the component in all of them.

## Rotating an Object using Code in a Script

### Simple Rotation

This is easily done with the following line of code:

objectName.transform.Rotate(0, 0, steerSpeed);

If the GameObject is in a variable, simply use the name of the variable. The example line will rotate around the Z axis from the current position by steerSpeed in degrees, rotating counter-clockwise, leaving X and Y rotation unchanged.

### Moving Rotation

When you are going to be moving an object you are rotating you will be doing objectName.transform.Translate(0, moveSpeed, 0) to move it along the Y axis (or you can change it along the X axis) just before you do the Rotate() as shown in Simple Rotation. Now the trick here is that you need to keep in mind: After you rotate the object using a Z rotation it will also be rotating the X & Y axis. If you want to move in a circle, you can just keep incrementing the position along the X or Y axis you were moving along while continuing to rotate the object by the same amount each time.

The same applies for any axis that you are rotating around. The other axis will be adjusted accordingly, so any movement after the rotation will be in a slightly altered direction.

## Camera Tricks

### Making a Camera Follow an Object using a Script

This is simply done by creating a script with a GameObject that is assigned the object you want to follow. You can either do this by assigning it to an object you find in the Start() method or make it a SerializeField object that you set in the Editor.

You can then follow the object using code like this:

[SerializeField]

GameObject obj; // Set in the editor to object camera is to follow.

[SerializeField]

float cameraOffset = -10f; // Camera needs to be offset from the object.

void LateUpdate()

{

transform.position = obj.transform.position + new Vector3(0f, 0f, cameraOffset);

} // Update()

You will note that LateUpdate() is being used instead of Update() here. This is because you want the camera to move after the object has finished moving and if you used Update() it might appear jerky, because the camera might try to move before the object does sometimes. This assures that the object has finished moving before you move the camera.

## Using Tags to Identify Objects

Every GameObject can have a Tag assigned to it to help you identify the type of object it represents in your game. Unity comes with a set of predefined tags that are available, which may vary based on the version of Unity you are using. For Unity 2021.2.7f1 it starts with this list, which you can choose from by clicking on the dropdown list for Tag in the Inspector:

* Untagged
* Respawn
* Finish
* EditorOnly
* MainCamera
* Player
* GameController

Everything defaults to Untagged. At the end of the list you will find “Add Tag…” that allows you to define you own tags. These tags will be available for any object within your projects once they are defined.

Tags are added by clicking on the plus (+) sign in the Tags & Layers dialog. Any tags assign to the object will be listed above the plus and minus signs, and numbered starting with Tag 0.

Tags can be removed from an object by clicking on the name of the tag to select it in the list and then clicking the minus (-) sign. The tag is not removed until the next time the project is loaded, so be aware that it will still be present until that happens. You can restore the default list, removing all defined tags, by doing a Reset in the Tags & Layers mode. If the defined tag is in use by an object, you can’t remove it, so you must change any object with that tag to be Untagged or have a different tag name assigned to it and then you can remove the tag name from the list.

There are two ways to check to see if an object has a tag property that matches what you are looking for. Since tags are simply a string you can check for them like this:

if (objectName.tag == "Package") { /\* Do something \*/ }

However, it is faster to use a method common to all GameObjects:

if (objectName.CompareTag("Package")) { /\* Do something \*/ }

## Using GetComponent() to Change Values on Components of a GameObject

You typically add different Components to a GameObject, like a SpriteRenderer to display an image or a C# script to execute code operations for that object. There are times in a script that you need to change properties of a component or, in the case of a C# script attached to the object, call a method that performs some action from another script.

If you need to perform the same action more than once it is best to cache the reference to the component when the Start() method executes, so save having to find it multiple times. If the component is only going to be used one time, or there are multiple objects of the same type, it might not be possible or worth it to cache the reference, in which case you should look it up when you need to use it. The reference can be stored in a temporary local variable if needed for multiple times within the same method before the method finishes.

Here is an example of it being cached in the Start() method and later used in the OnTriggerEnter2D() method to change the color based on whether it has a package or not:

Color32 hasPackageColor = new Color32(255, 0, 255, 255);

Color32 noPackageColor = new Color32(255, 255, 255, 255);

SpriteRenderer spriteRenderer;

void Start()

{

spriteRenderer = GetComponent<SpriteRenderer>();

} // Start()

void OnTriggerEnter2D(Collider2D collision)

{

if (collision.CompareTag("Customer"))

spriteRenderer.color = noPackageColor;

else if (collision.CompareTag("Package"))

Destroy(collision.gameObject, delayDestroy);

} // OnTriggerEnter2D()

# Snow Boarder Game from Complete C# Unity Game Developer 2D

## Creating a Sprite Shape for the Snow Surface

You do this by choosing the *2D Object* 🡪 *Sprite Shape* 🡪 *Closed Shape* in the Hierarchy. This creates a closed loop made up of splines. The other option in the sub-menu is Open Shape, which is a line made up of splines.

### Creating a Custom Sprite Shape Profile for a Sprite Shape

The default shape looks sort of like the tread of a tank. This is the default Sprite Shape Profile. You can create your own Sprite Shape Profile by right-clicking in an Asset folder and selecting *Create* 🡪 *2D* 🡪 *Sprite Shape Profile*. Normally you would do this in your Sprites folder for the project. An image representing the Sprite Shape Profile appears inside the Closed Shape in the Scene Editor window after you drag your new Sprite Shape Profile over the box labeled Profile for the Sprite Shape Controller component. This now overrides the default Sprite Shape Profile.

To use your custom Sprite Shape Profile, select the Sprite Shape in the Hierarchy and drag the Sprite Shape Profile you have created to the *Profile* box in the Inspector. For the Snow Boarder game this was renamed to be *Snow Profile*.

#### Changing the Look of a Sprite Shape with the Sprite Shape Profile Settings

There can be a list of Sprites added to the Sprite Shape Profile. By default, it has one that is set to *Sprite Shape Edge*. You can change this by dragging a sprite from your assets in to replace the default one. For Snow Boarder we use *Snow-tile-low- res*. This becomes the edge of the shape, but leaves the middle still as white.

You can fill in the middle by dragging a sprite into the *Texture* box if the Fill section for the Sprite Shape Profile. The default is sprite-shape-fill when it is created, which is simply white. For Snow Boarder we change this into a blue color.

### Creating More Control Points for Modifying the Sprite Shape

In the Sprite Shape Controller section in the Inspector click on the Edit Spline button and the control points that are pre-defined for the shape will show up along the edge of the shape. You can add additional control points by clicking on any spot along the edge. You can drag these points all over the place and each point is the center of a spline shape that makes up the edge. The splines merge and meet up to make smooth curved lines.

### Deleting Control Points for a Sprite Shape

You can remove a Control Point by simply clicking on it and hitting the Delete key on your keyboard.

### Changing the Spline Shape Around a Control Point for a Sprite Shape

When you click on a control point for the Sprite Shape you will see two other dots, slightly smaller and lighter in color with a line between them going through the Control Point. Click on either of these end points and drag them around. Moving them to the side will cause the spline to bend. Pulling them closer or farther from the Control Point will change the length for that end of the spline. Not only does the Sprite Shape change with the spline, but any applied texture will also follow the change in shape.

### Adjusting the Collider Offset from the Sprite Shape

This is done in the Sprite Shape Controller component in the Inspector (for these shapes you should be using an *Edge Collider*). It has a section dedicated to Collider settings. In there you can adjust the Offset for the collider from the shape. By default, this is the max setting of 0.5, so adjusting it lower will bring the edge of the collider closer to the sprite image. The setting you need will likely vary depending on the texture you have applied to it.

### Adding a Sprite Image as a Texture to a Sprite Shape

You can set a sprite image as a “texture” for a Sprite Shape. In the Sprite Shape Controller component, you will see a box called Profile that can be set to the sprite image you wish to use as a texture. Simply drag the sprite you wish to use into the box from a folder and drop it in the box.

#### Setting Border Limits on the Sprite used as a Texture

Click on the Sprite in your Assets and click on the Sprite Editor button in the Inspector. You will see border settings in the dialog box that comes up. Normally these are all set to 0, but if you change them, you limit what part of the sprite is used. In the one used for Snow Boarder the Left and Right borders prevent the curve at the ends from being used.

#### Wrap Mode for Sprites use for Texture or Fill

If you click on the Sprite in your Assets, you will see a setting called *Wrap Mode*. For both the sprites used in Snow Boarder this is set to *Repeat*. Changing this to one of the other settings will change how it looks as it fills the area it covers in your Sprite Shape.

### Adjusting the Height of a Sprite Shape Texture Image

This is a lot trickier than most thing and **very** non-intuitive. Here are the steps, which are done using the Sprite Shape Controller area of the Sprite Shape:

* Click on the Edit Spline button () and you will see the control points appear
* Drag a bounding box to cover all the spline points, which will now be highlighted as yellow points instead of white
  + After you highlight the spline points the buttons for Tangent Mode and the Position X & Y boxes, which were grey, will become active and available. In addition, a Height drag bar and value, plus controls called *Corner* and *Sprite Variant* will appear just under them
* Changing the height, which defaults to 1, will cause the texture to get larger or smaller

### Adding a *Surface Effector 2D* to Make Snow Boarder Move

Selecting the Sprite Shape, you can add a component called an Effector to it. In the case of Snow Boarder, since we are using a Closed Sprite Shape, we use one called a *Surface Effector 2D*. You will see a warning show up right away. You clear this warning by enabling the checkbox in the Collider for the Sprite Shape marked *Used by Effector*. Since you may have more than one collider for an object you select this for the one that will be used by the Effector to do it’s magic.

You can change the speed or force of the Effector by changing the *Speed* setting, which defaults to 1. Think of this effect as a conveyor belt that is always moving. A positive value moves to the right and a negative value moves to the left.

The *Force Scale* setting is how much effect Gravity will have. Moving it to 1 means it overcomes Gravity (set on the object being effected), while a lower fractional amount will allow Gravity to have more of an effect.

Note that changing these values in code can be used to speed up, slow down, or change direction under player control.

## Creating Sprites with Collider 2D and Rigid Body 2D Components Pre-Installed

When you right click in the Hierarchy, instead of choosing 2D Object 🡪 Sprite and one of the shapes from that submenu, you can instead choose 2D Object 🡪 Physics and choose either a Static Sprite or Dynamic Sprite from the submenu. A Static Sprite defaults to being a square while a Dynamic Sprite defaults to a circle. You can change the chape of the sprite by selecting a different one for the Sprite setting in the Sprite Renderer component.

The advantage of choosing these over the generic sprite shapes is that it automatically adds a Collider 2D and Rigid Body 2D to the sprite. The Collider will be one for the default shape of the sprite you choose, so choosing a different shape will probably require you change the Collider component to a matching shape. The Rigid Body 2D will be set to Dynamic or Static, based on your choice.

## Adding Cinemachine to Your Project for Creating Virtual Cameras

In your Project window you will see a Packages folder under the Assets folder in the list. Clicking on the arrow to see the contents as a list, or click on the name to see the folder contents, you will see the list of standard packages Unity adds for any 2D project.

To add the Cinemachine package go to *Windows* 🡪 *Package Manager* and change the *In Project* setting to be *Unity Registry* to see all the possible Unity packages. In the search box just type “cine” and you will see *Cinemachine* listed, which is part of the *Cinematic Studio* set of packages. Click on the name on the left and the click the Install button.

Now when you right click in the Hierarchy area you will see a new Cinemachine entry in the menu.

### Making a Virtual Camera

In the Cinemachine submenu for the Hierarchy you simply select Virtual Camera, and you will get something labeled *CM vcam* with a number. A symbol for the virtual camera will appear to the right of the Main Camera, showing there are one or more virtual cameras.

It will work best if you name the camera for what it will be used for. Using VC as the start of the name will help show which items are Virtual Cameras.

### Making a Virtual Camera Follow Something in the Scene

You can do this by clicking on the VC in the hierarchy and in the Inspector you will find a setting called *Follow*. Clicking on this you can select the item you want the camera to follow, or drop the object on it from the hierarchy. Using the newer version than what is shown in the class I get a warning saying it needs a target set as well. This is just below the Follow setting and setting it to the same object removed the warning.

#### Adjusting the Follow Offset of the Virtual Camera

Sometimes you want the camera to not be centered of the object it is following. In the example of Snow Boarder it is better to see more of what is coming up than what has already passed. This can be easily done by changing the *X* offset of the camera, so it is in front of the object being followed.

In the lesson this shows up under the *Body* settings, but in my newer version of Unity it shows up in the Aim settings. The settings for this are *Screen X* and *Screen Y*. By default, they start at 0.5 on the slider. Lowering the value will make it lead the object being followed on the positive side of that axis, so a lower Screen X will show more of what is ahead of the object being followed on the X axis. A setting of 0.25 would mean that ¾ of the screen will be ahead of what you are following on that axis.