Introduction to 2D Game Development Workshop

Participant E-book

Get started developing
in Unity for 2D games

Table of Contents

Introduction:

Learning Action Plan and
workshop overview

Action Plan

The Learning Action Plan was developed by Unity to help you identify your personal goals in learning how to use the engine, and determine the best path to take to achieve those goals.

The Plan should be treated as a living document: Every time you take a new course, build a new project, or research a new subject, you should record it. This will give you a record of your progress and aid you in keeping to your personal roadmap.

Project Introduction

The goal of this workshop is to get you up and running with Unity, focusing specifically on working with its suite of 2D features. As you work your way through the activities, you will build a 2D game from the ground up, covering major subject areas including sprites, animation, game mechanics, and publishing.

The project you will build is a side-scrolling, endless-runner game that will run on mouse input. After the initial mouse click, the player will begin to run, and all subsequent mouse clicks will result in the player jumping. The goal of the game is to run to the end of the level while collecting as many objects as possible without falling off the platforms.

Unity's official documentation is comprehensive and serves as an excellent reference point if you wish to explore additional aspects of a subject. It can be found at the following link: Unity User Manual.

Requirements

This workshop was developed for **Unity 2017.2** and utilizes components of the Engine that were not available before this build. While this workshop should work without issue in future builds, it cannot be completed with previous versions of the Engine

Activity 1:

Project setup and
scene creation

Set up and configure the 2D project

1. Launch the Unity Editor. In the Launcher, click the New icon in the upper-right corner of the Projects window.

2. Name the new project 2D Workshop and set the project's location to somewhere easy to remember.

3. Set the project to 2D and leave Enable Unity Analytics set to Off.

4. Click the Create project button on the lower right of the window.

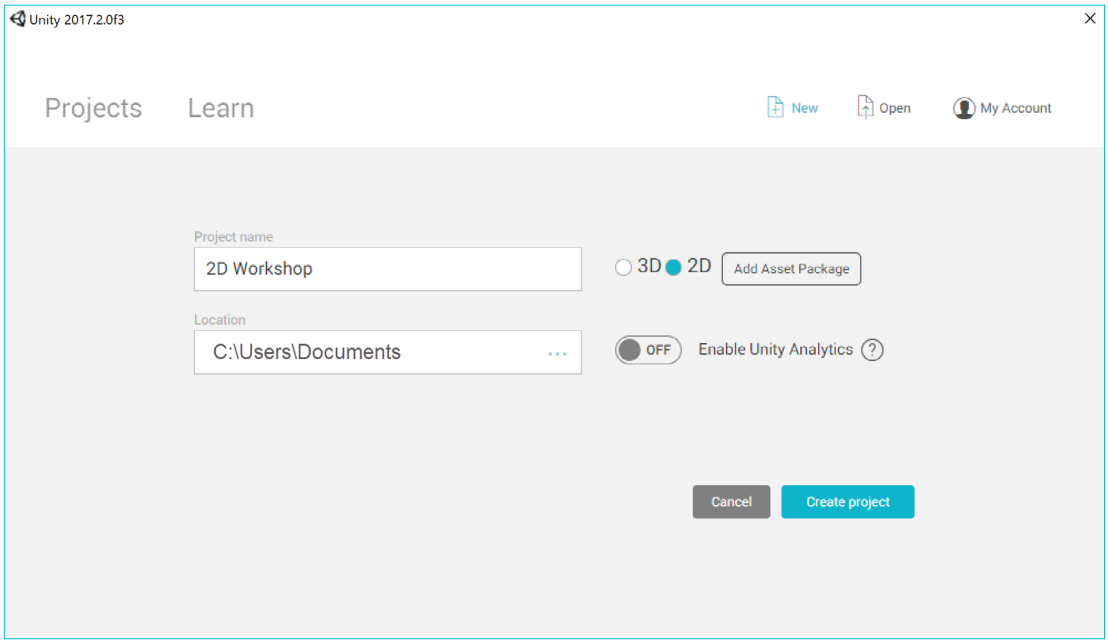


Figure 01: Properly configured Project setup

The window will close as the project is configured. After a moment the Unity Editor will launch. You will see the name of your project at the top of your screen.

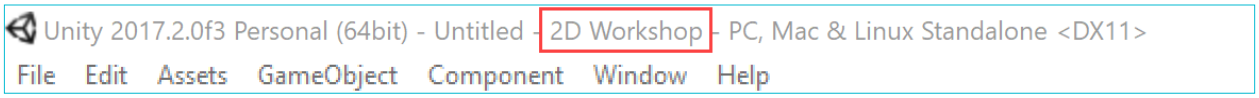


Figure 02: The Editor window lists the name of the active Project

Identify main Editor features

Unity's default layout features all of the major windows that are used during regular development. You can add additional windows from the Window dropdown menu.

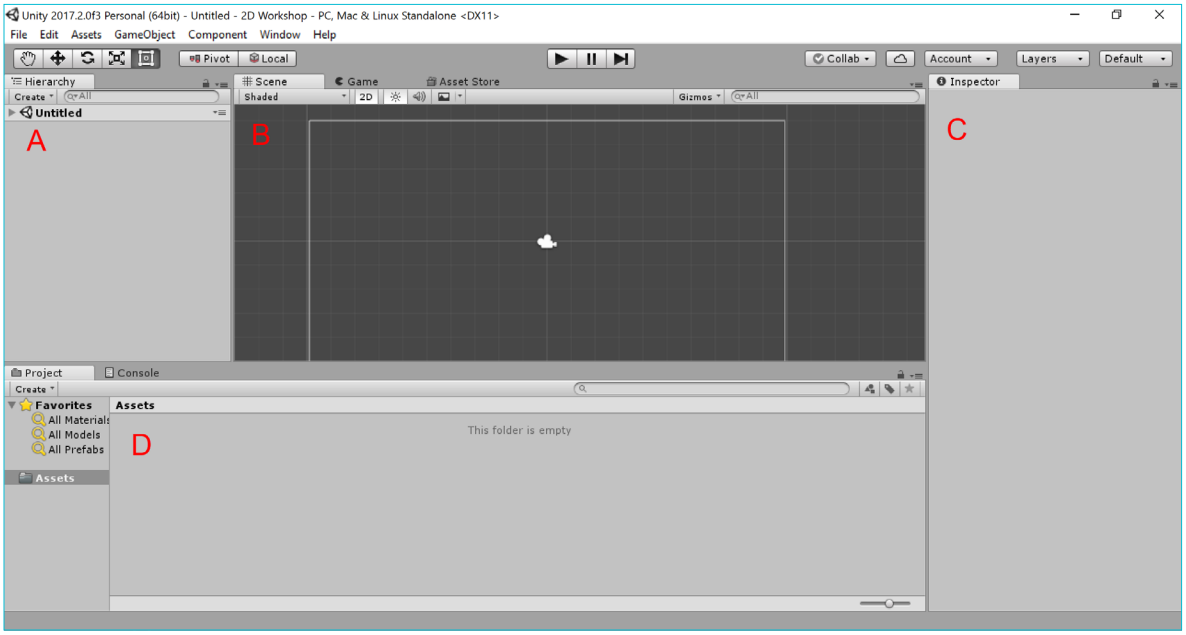


Figure 03: The default configuration of the Unity Editor

A. **Hierarchy Window**: Lists all GameObjects that exist in the active scene.
B. **Scene View**: Main work area.
C. **Inspector Window**: Lists the properties of a selected GameObject or asset.
D. **Project Window**: Lists all of the assets, scenes, and scripts contained within the created project.

All Unity windows can be “torn off” by left-clicking and dragging on the window’s tab, and can be docked elsewhere in the Editor, or left free-floating on the main or alternate monitors.

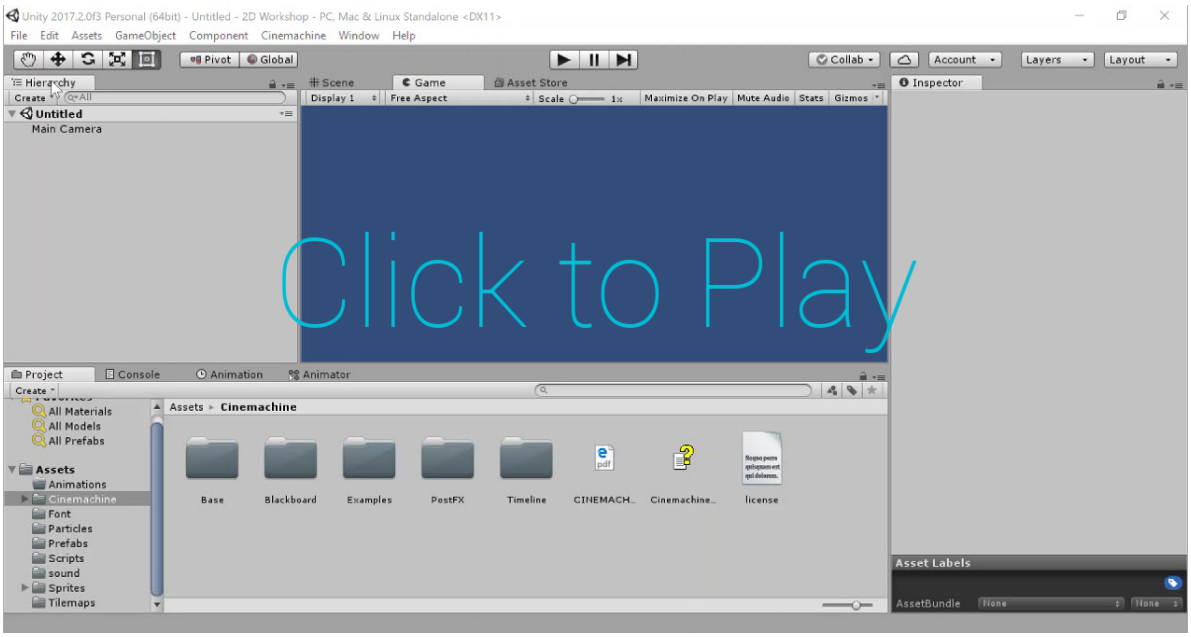


Figure 04: Tearing off the Hierarchy window.
Note the automatic snapping.

To restore the default layout, click the Layout dropdown in the upper-right corner of the Editor, and select Default.

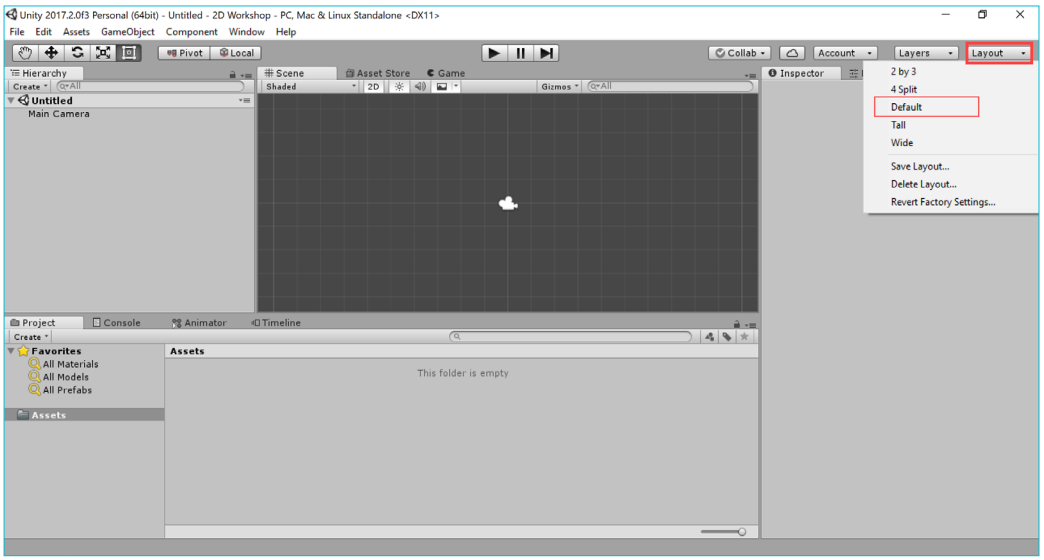


Figure 05: Restore the layout by selecting Default from the Layout dropdown.

Navigate in the Editor

Pan: To pan around in the Scene view, hold the ALT key, and Left Mouse Button click and drag.

Zoom: To zoom in and out of the Scene view, hold the ALT key and Right Mouse Button click and drag.

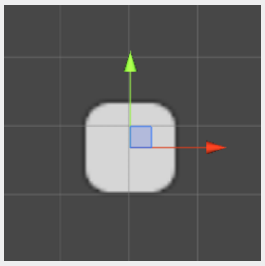
Note that you cannot rotate in the Scene view when working in 2D.

Above the Hierarchy tab is a row of tool buttons that can be used for navigating in the scene.

Figure 06: Toolbar

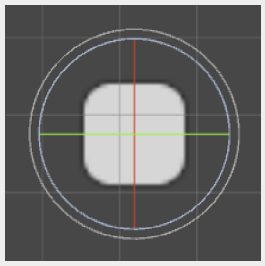
A. **Hand tool** (Hotkey Q): Allows you to left-click and drag around in the scene view. Objects cannot be selected when this tool is active.

B. **Move tool** (Hotkey W): Shows the move widget on a selected object, allowing you to move the object interactively in the Scene view. Click and drag on the arrow pointing along the desired axis. To move along both axes at the same time, click and drag on the blue square at the center of the tool.



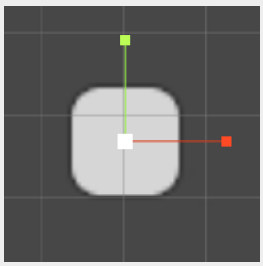
B: Move Tool

C. **Rotate tool** (Hotkey E): Shows the rotate widget on a selected object, allowing you to rotate the object interactively in the Scene view. Click and drag on the axis you wish to rotate along. The white outer circle allows you to rotate in multiple axes at once.



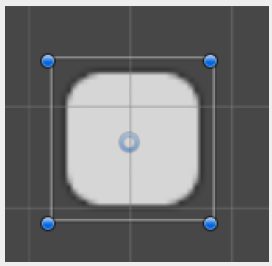
C: Rotate tool

D. **Scale tool** (Hotkey R): Shows the scale widget on a selected object, allowing you to scale the object interactively in the Scene view. Click and drag on the cube at the end of the axis you wish to scale. The white center cube allows you to scale in all axes at once.



D: Scale tool

E. **RectTransform tool** (Hotkey T): Designed to work with 2D objects, and is a combination of the Move, Rotate, and Scale tool. To move an object, click and drag on it; to rotate it, move the mouse pointer to one of the tool corners and it changes from an arrow to a rotate icon; and to scale it, grab one of the tool corners and drag.



E: RectTransform tool

F. **Pivot handle position**: Changes where the tool gizmo appears on the object. This only changes if the GameObject has a custom pivot point.

G. **Handle rotation**: Changes which axis the tool is constrained to. Click to switch between global/local. When set to local, the tool will be rotated based on the selected GameObject's relative rotation. When set to global, the tool will always be aligned to global rotations.

2D elements and their components

A sprite is a 2D graphical object. Unlike other image types such as textures, they are meant to be used as their own components rather than be applied to another object, such as a 3D mesh. In 2D projects, all images are automatically imported as sprites.

Sprites can be brought into Unity on their own as single sprites, or as a group combined in a single image, referred to as a sheet.

Single Sprite

Sprite Sheet

Sprite sheets are commonly utilized to group common sprite types together, such as parts of an animation. They are also useful for keeping the project more efficient, as objects sharing the same sheet are only processed once at run time--however, Unity features its own specialized sprite packer that offsets this issue for single sprites as well.

Before a Sprite sheet can be used in a game, it must be first sliced so each individual sprite can be accessed.

Preparing a Sprite Sheet

1. Select the Steam Sprite sheet and in the Inspector, set the Sprite Mode to Multiple, and then click the Sprite Editor button.

2. When asked if you want to apply Unapplied Import Settings, click Apply.

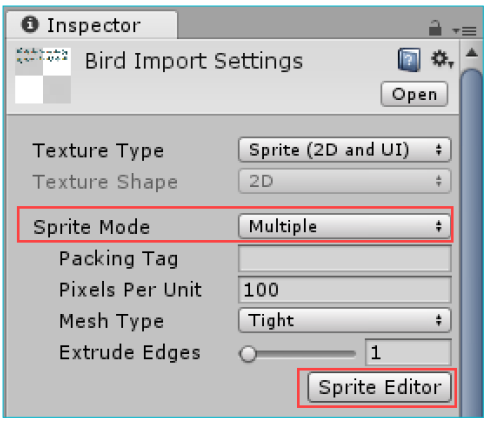


Figure 06: Sprite Mode must be set to Multiple before entering the Sprite Editor

3. The Sprite Editor window will open.

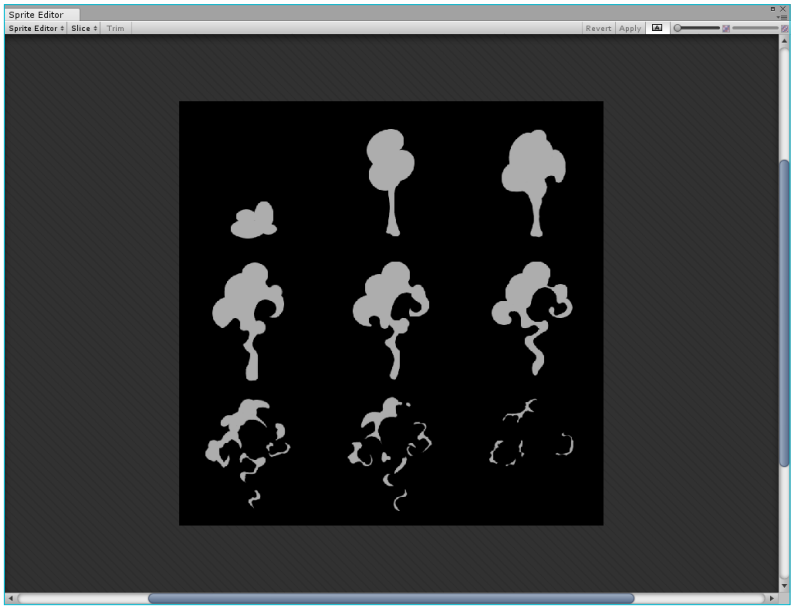


Figure 07: Steam Sprite sheet in the Sprite Editor.
Note: The sheet alpha is shown here for visibility

4. To generate individual sprites that are all occupying the same amount of space, click the Slice dropdown and set the Type to Grid By Cell Count.

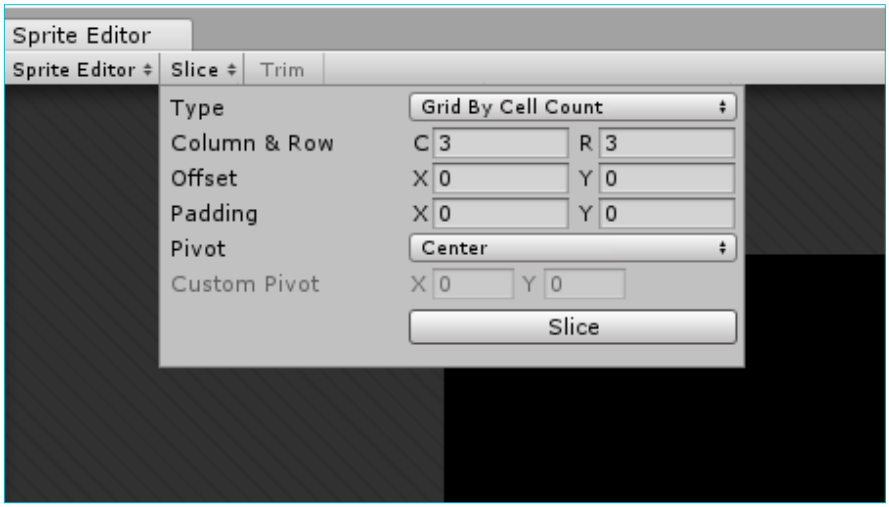


Figure 08: Slice settings for the Steam Sprite sheet

5. Set both the column and row option to 3 and click Slice. You will see each sprite evenly outlined.

6. Close the Sprite Editor, and when prompted, apply the changes.

7. In the Project window, click the small arrow on the right side of the Sprite sheet. You should now see each sprite represented individually.

Figure 09: When the arrow rollout on the right side of the sheet is clicked, all individual sprites will be visible.

8. Now individual sprites from the sheet can be used independently by dragging them into the scene.

Tile assets (Tilemaps)

Tile assets (commonly referred to as Tilemaps) are Sprite sheets that are specifically designed to group sprites to be used with Unity's Tilemap system, which allows you to quickly generate 2D levels by painting sprites onto a grid. This system is discussed in detail in a later chapter.

Tilemaps are no different from other Sprite sheets in how they are prepared, but often appear more densely packed with sprites of the same or similar size.

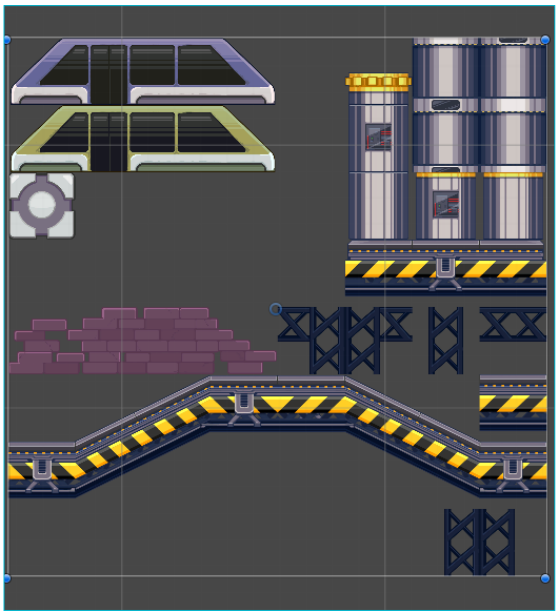


Figure 10: An example of an environmental Tilemap

Pixel density

When generating 2D art for Unity, it is important to keep all assets at the same relative scale. This ensures that the resolution of the assets remains the same, even if their size is adjusted in the Editor.

For sprites, this can be done by adjusting the Pixels Per Unit setting, which appears in the Sprite Mode area of the Inspector.

Pixels Per Unit refers to how many pixels appear in one Unity unit, which equates to a meter. By default, this is set to 100.

Generally speaking, Sprite transforms should not be scaled, but rather their size adjusted via Pixels Per Unit.

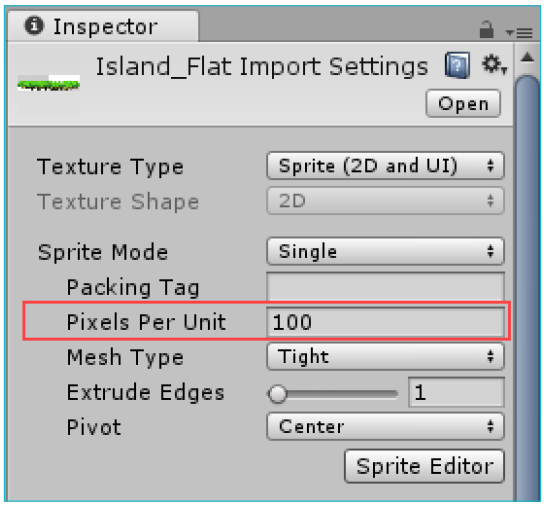


Figure 11: Pixels Per Unit setting

Creating and editing Colliders for sprites

Colliders allow GameObjects such as sprites to physically interact with other objects in a scene. Notable examples of elements that require a Collider include the Player, the ground, and elements that block the Player's path, such as walls.

Colliders essentially define the shape of the object, allowing Unity's Physics system to accurately calculate how other objects should react to it.

There are two major Collider types in Unity: 2D and 3D. Despite this specific delineation, 2D objects can have 3D Colliders, and vice versa. This allows designers to include both 2D and 3D objects in the same game without difficulty.

The main issue to keep in mind when working with Colliders is that the two types do not interact with one another. This means that if a GameObject with a 2D Collider comes into contact with a GameObject that has a 3D Collider, they will not detect each other, and will not interact in any way.

It is important, therefore, to be consistent with the Collider type used throughout a game. In this workshop, 2D Colliders will be used exclusively.

Configuring a Collider for a single sprite

The five most common 2D Colliders for sprites are Box, Circle, Edge, Polygon, and Capsule. Colliders do not need to perfectly match the shape of the sprite. Rather, they should approximate the shape closely enough that its interactions with other objects are convincing.

Highly detailed Colliders are expensive to process, and generally do not return a better interactive experience than a more primitive shape.

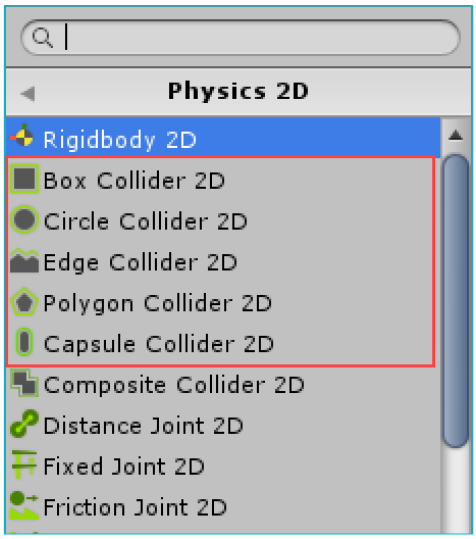


Figure 12: 2D Colliders

Configuring the Flat platform

1. Locate the TileMap_01 Sprite sheet from the Project window and slice it using Automatic Slicing.

2. Drag the flat platform sprite from the TileMap_01 sheet into the Scene view.

3. With the platform sprite selected, in the Inspector, click the Add Component button, and navigate to Physics 2D. Select Box Collider 2D.

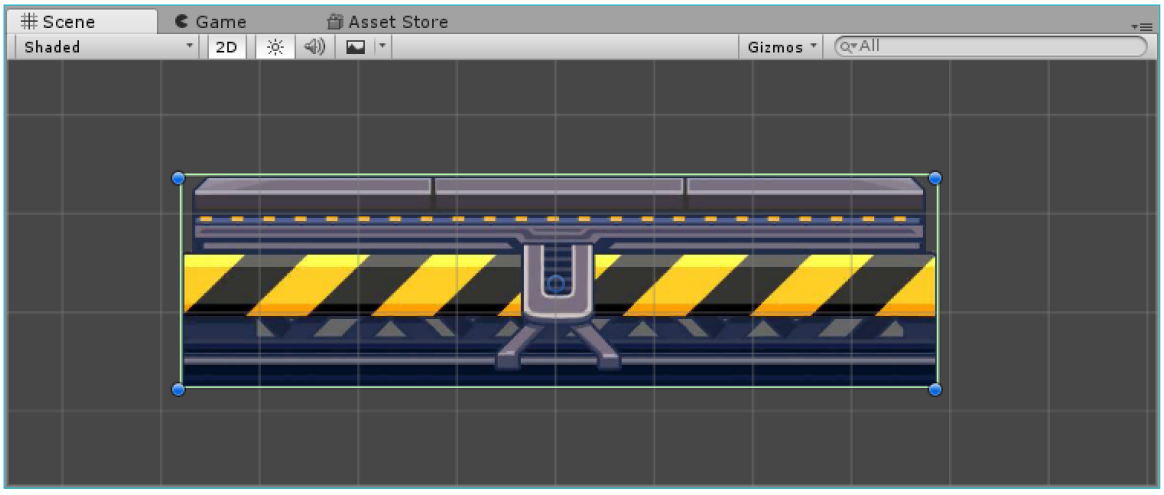


Figure 13: A Box Collider 2D applied to the platform Sprite.

The Collider appears as a green outline surrounding the platform.

4. In the Inspector, click the Edit Collider button.

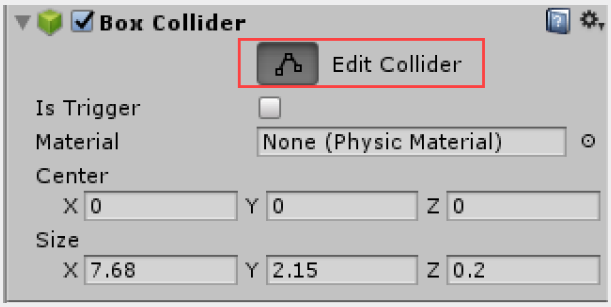


Figure 14: The Edit Collider button

This activates the Editor handles on the Collider itself, which are represented as green squares in the center of each edge.

5. Align the left and right edges of the Collider to align with the top, narrower level of the platform. Click the Edit Collider button to exit Edit mode.

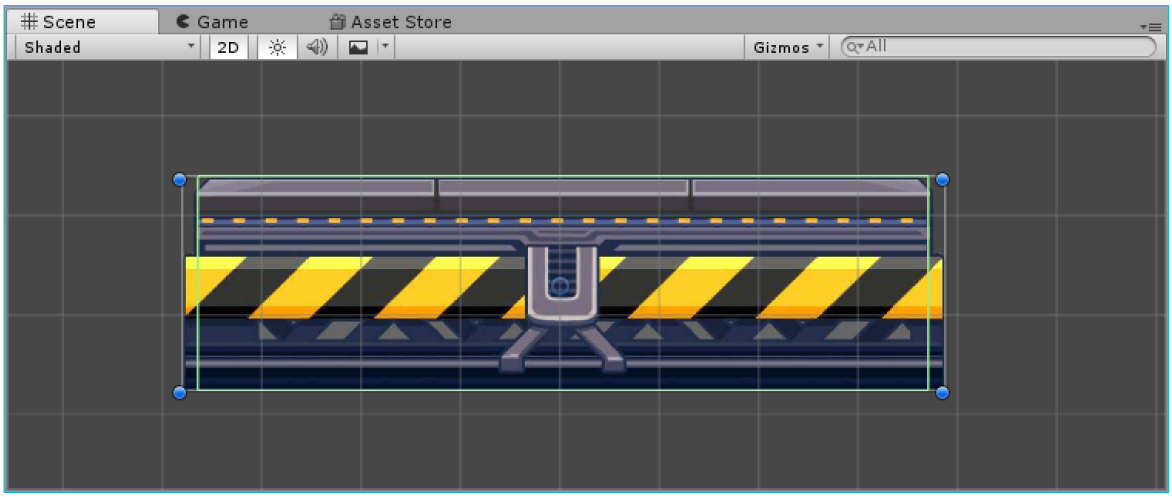


Figure 15: The Collider slightly adjusted to more accurately fit the edges of the platform.

Configuring the angled platform

1. Drag the angled platform sprite into the Scene view.

2. Apply an Edge Collider to the sprite. A single line appears in the center of the sprite. Enter Edit Collider mode.

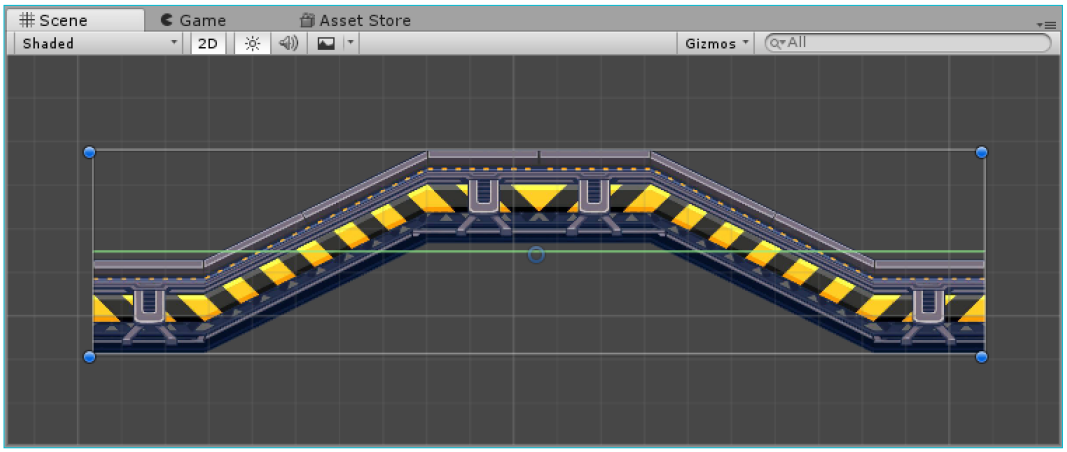


Figure 16: An unedited Edge Collider.

3. As the mouse is moved across the Collider, a green box appears. Clicking and dragging creates a new point on the Collider, which can be used to edit the shape. Holding CTRL and clicking a point will delete it. Edit the Collider so it follows the outline of the top of the platform.

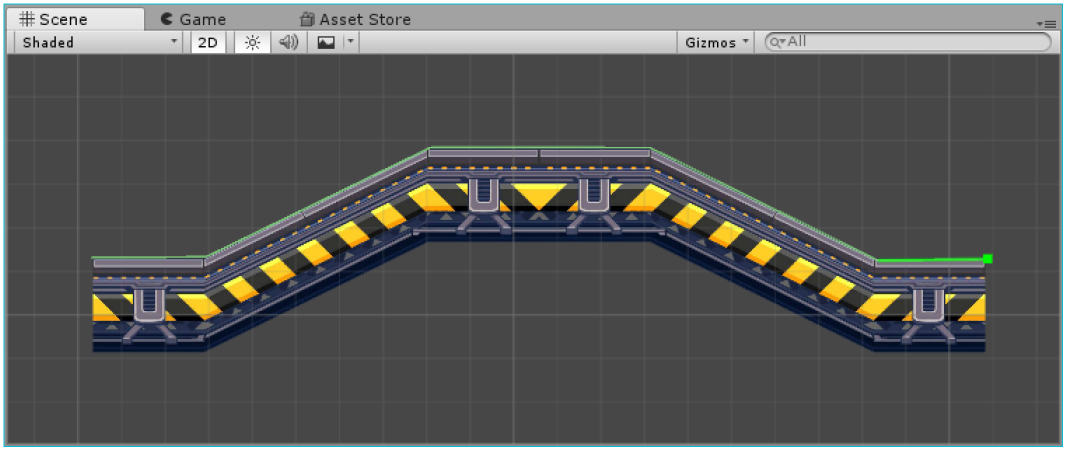


Figure 17: The Edge Collider configured to match the shape of the Sprite.

Edge Colliders are ideal for objects that are too complex for the other primitive Colliders, but because the Collider is a single edge, it's possible for Players to fall through the sides of the object.

Due to this issue, this Collider type should only be used when the Player won't have access to the sides of the sprite, such as when they are blocked by other Colliders.

Creating Prefabs

Prefabs are a special type of component that allows fully configured game objects to be saved in the Project for reuse. These assets can then be shared between scenes, or even other projects without having to be configured again.

This is extremely useful for objects that will be used many times, such as the platforms that were configured above. A major benefit to Prefabs is that they are essentially linked copies of the asset that exists in the Project window.

This means that changes made or applied to the original Prefab will be propagated to all other instances. This makes fixing object errors, swapping out art or making other stylistic changes very efficient.

Creating the platform Prefabs

1. In the Project window, create a new folder named “Prefabs”.

2. Select the configured flat platform sprite in the Hierarchy and drag it into the Prefabs folder. A Prefab is automatically generated.

3. Repeat this process for the angled platform.

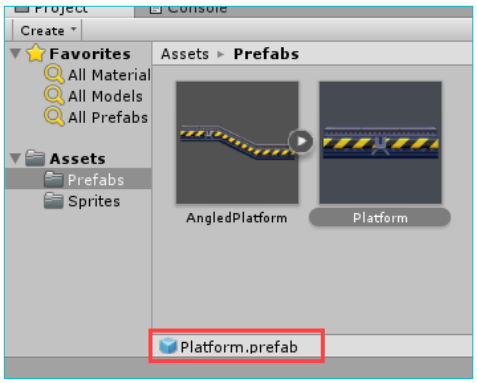


Figure 18: Platform Prefabs. Note that the asset type is a .prefab

Prefabs look quite similar to other objects that appear in the Project window. However, when selected, their file type will be a *.prefab. When the Prefab is selected, the Inspector displays all of the components that were configured on the original object.

Notice that the flat platform sprite in the hierarchy has now turned blue, which signifies that it is a Prefab object.

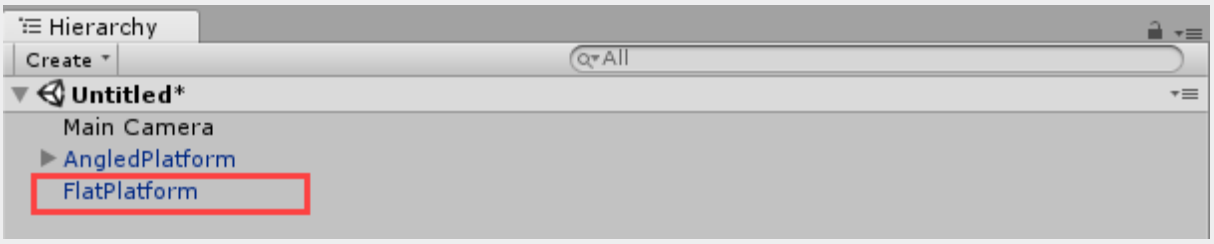


Figure 19: Prefabs appear blue in the Hierarchy

Editing a Prefab

When you convert an object to a Prefab, a new set of buttons is added above the Transforms values.

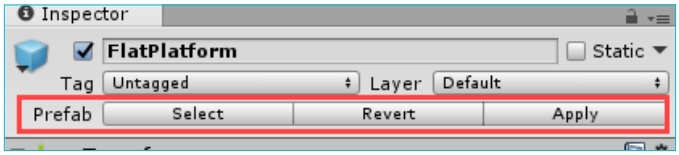


Figure 20: Prefab buttons appear above the Transform values in the Inspector

Like any other asset in Unity, Prefabs can be edited after they are created. A Prefab can be edited on a per-object basis, where a single instance of a Prefab is changed for an individual purpose, or these changes can be applied to all instances of the Prefab.

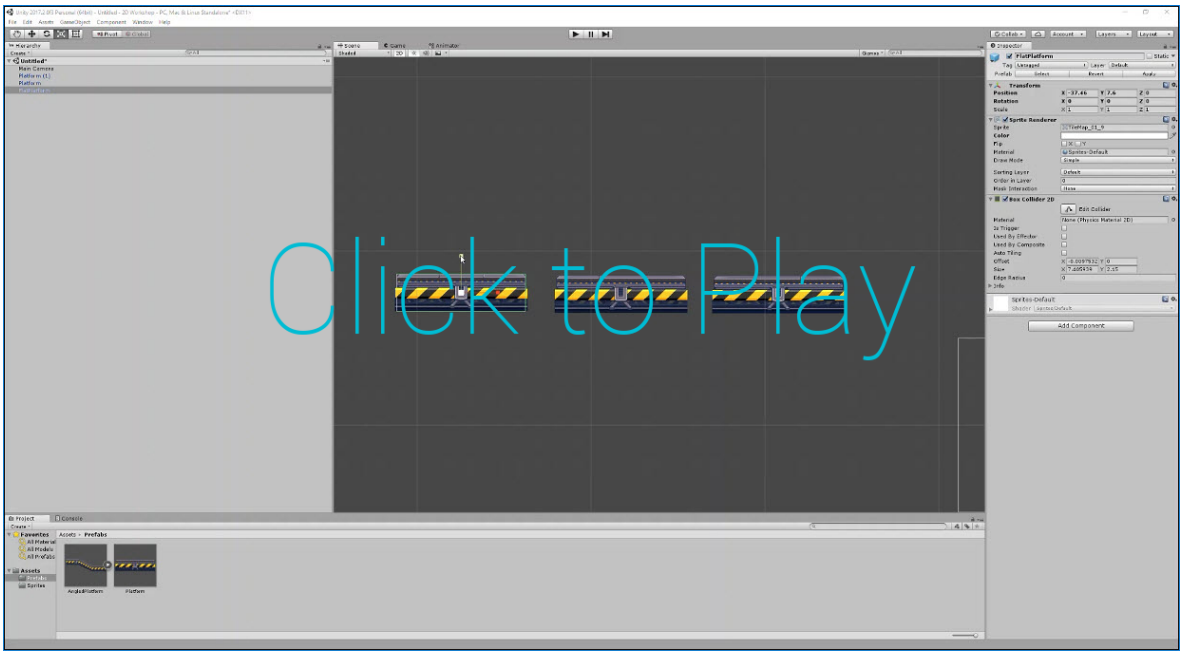


Figure 21: The video first illustrates reverting an edited Prefab back to its original form, then changes to one Prefab being applied to all others.

The Select button is used to highlight the original Prefab in the Projects folder. This is helpful for projects that have become large with many different Prefabs spread throughout many folders.

The Revert button removes all changes applied to a Prefab that do not match with the Prefab in the Project folder.

The Apply button takes all the changes made to the selected Prefab and applies them to all other existing instances, as well as to the original in the Project window.

Introducing the Tilemap

The Tilemap system is a set of tools for rapidly generating 2D sprite-based levels. It allows the Level Designer or Artist to focus entirely on creating the level rather than aligning and sorting sprites manually. The interface is similar to a paint program.

Configuring the Tile Palette

1. In the Project window, create a new folder named “Tilemaps”.

2. Right-click in the Hierarchy window and navigate to the 2D Object flyout. Select Tilemap

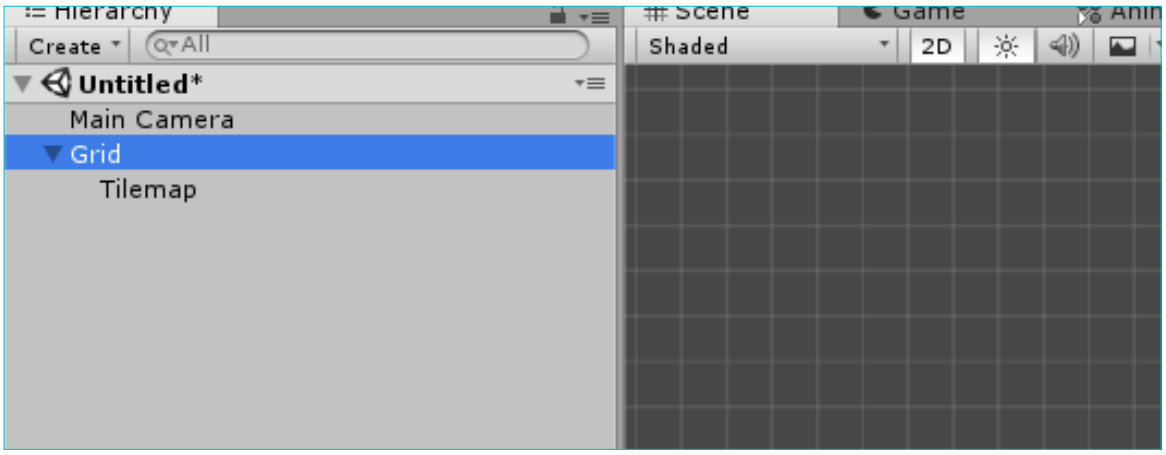


Figure 22: The Tilemap object is a child of the Grid, which is visible in the Scene view.

A Grid GameObject is added to the Hierarchy, with a child object named Tilemap. When the Grid object is selected, the Scene view changes to show a defined grid pattern in the background.

You can alter the size of the grid by changing the Cell Size in the inspector. Unless the sprites being used were specifically designed to be non-square, the X and Y Cell Size values should always remain the same.

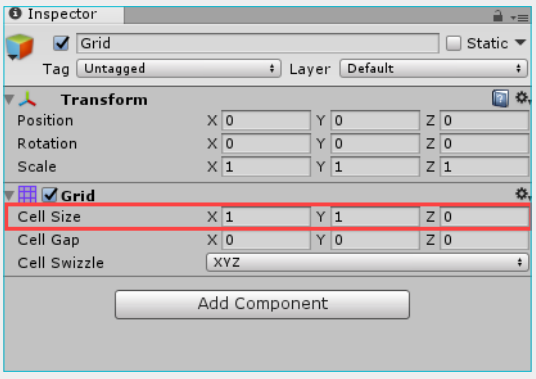


Figure 23: The Cell Size values change the scale of the grid

3. Rename the Tilemap child object to “BKG_Tilemap”.

4. From the Window dropdown, select Tile Palette. Dock the Tile Palette window below the Inspector to maintain visibility in the Scene view.

5. Click the Create New Palette button. In the Name field, type “Background”.

6. Select the Tilemap folder in the Project window to save the Palette.

7. Locate the TileMap_01 and TileMap_02 Sprite sheets and select several Sprites. Drag them into the Tile Palette gridded work area.

8. When prompted, save each Tile asset in the Tilemaps folder.

Working with the Tile Palette

The Tile Palette features several tools to build 2D scenes.

Figure 24: Tile Palette tools.

A. **Selection tool**: Click and drag to select an area of sprites on the grid
B. **Move tool**: Moves sprites selected with the Selection tool
C. **Brush tool**: Paints the selected sprite in the Scene
D. **Rectangle tool**: Allows a selection on the grid to be drawn, and then fills it with the active sprite
E. **Picker tool**: Sets the selected sprite as the active sprite in the Tile Palette. If the Paint tool is active, holding CTRL will activate the Picker.
F. **Eraser tool**: Erases sprites clicked on in the Scene view
G. **Fill tool**: Fills the grid area with the active sprite

To begin painting sprites into the Scene view, select the Brush tool, and then click the desired sprite. Left-click and drag to lay down sprites on the grid.

Working with layers

It's often helpful to create a 2D level with multiple Sprite layers, both for organizational and design purposes. Multiple layers allow the designer to sort sprites into different "depths," allowing them to establish objects that appear in the foreground, middleground, and background.

This allows sprites to sit on top of one another, as well as allow some sprites to pass in front of the Player, while others stay behind at all times. This adds a level of visual interest and believability to the level design.

1. Right-click the Grid object in the Hierarchy and, from the 2D Objects Flyout, select Tilemap.

   Notice that rather than creating a secondary Grid, a new Tilemap was added as a child.

2. Rename the new Tilemap "FG_Tilemap".

3. Select the active Tilemap dropdown in the Tile Palette and set it to FG_Tilemap.

4. Select a sprite and paint in the Scene view. Notice that this sprite appears above the background sprites.

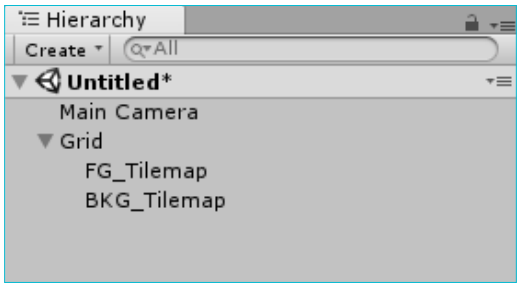


Figure 25: Multiple Tilemaps parented under a single Grid

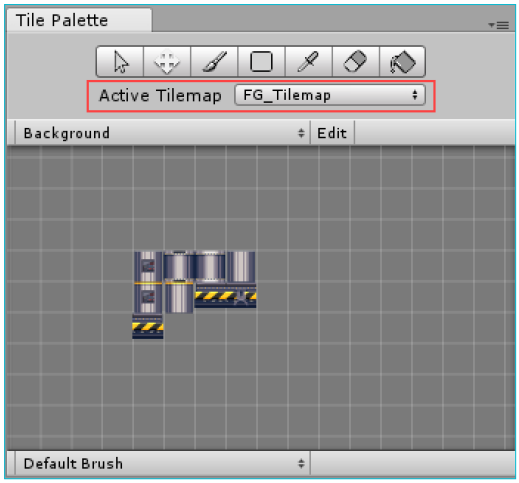


Figure 26: Switching the Active Tilemap allows you to control what layer is painted on

Activity 2:

Animation and
Player creation

Creating Sprite sheet animations

Sprite-based animations are automatically generated when more than one sprite is selected and dragged into the Scene or Hierarchy at the same time.

1. Use Automatic Slicing to configure the Collectibles Sprite sheet.

2. Select the coin sprites and drag them from the Project view into the scene.

3. Unity will prompt you to save the name of the animation. Note that the location will default to whatever folder is currently active in the Project view, so it will likely be sprites. In the root Assets folder, create a new folder and call it Animations.

4. Name this animation 'Coin_Spin' and save it in the Animations folder.

5. Press Play, and you should see your animation play.

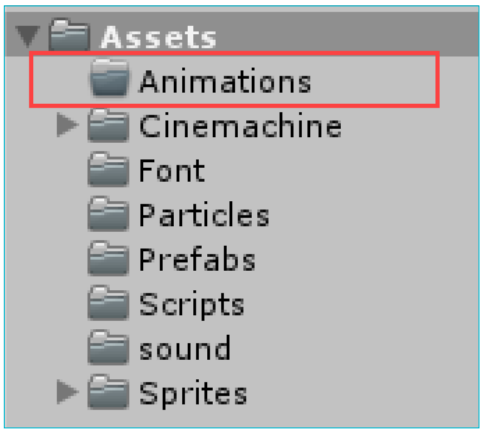


Figure 27: Create a new folder for Animations

Building on Sprite sheet animation with keyframe animation

You can apply multiple animation clips to a single object and then layer them together to achieve a more complex effect.

1. From the Window dropdown, select Animation. Dock the Animation window behind the Project window for ease of navigation.

2. Select the coin Sprite from the Hierarchy. The automatically generated keyframes become visible in the Timeline.

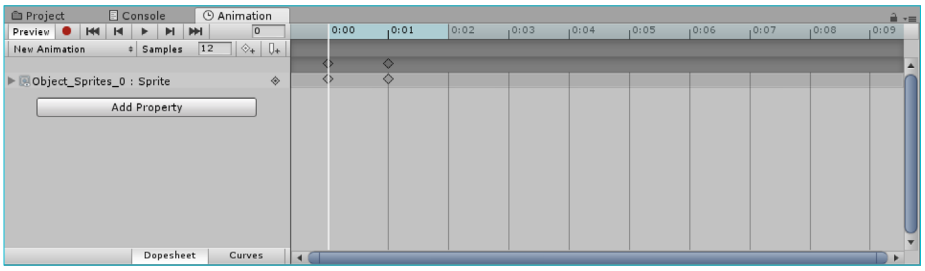


Figure 28: The Animation window

To see a preview of which sprite exists at each keyframe, click the roll-down arrow to the left of the property.

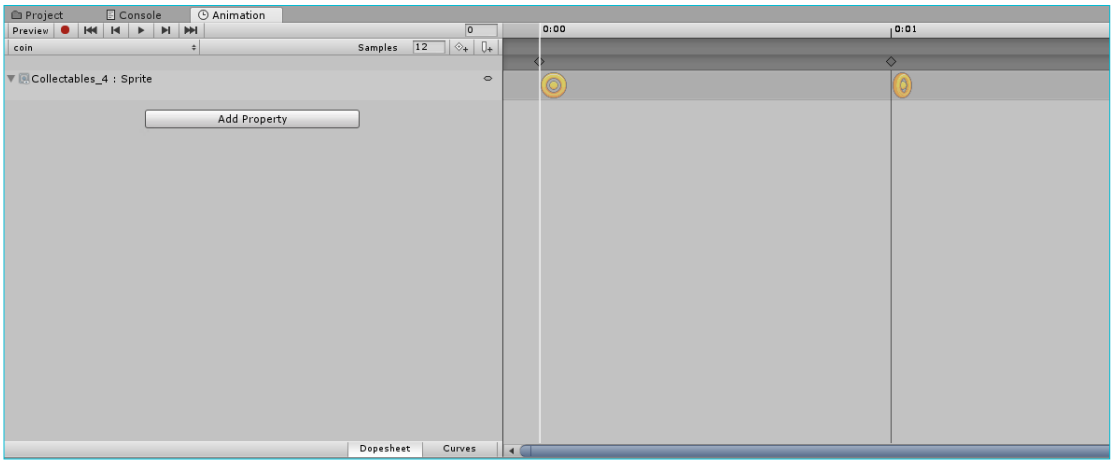


Figure 29: Sprite previews visible

To create a new animation clip, click the dropdown to the left of the Samples section, and select Create New Clip.

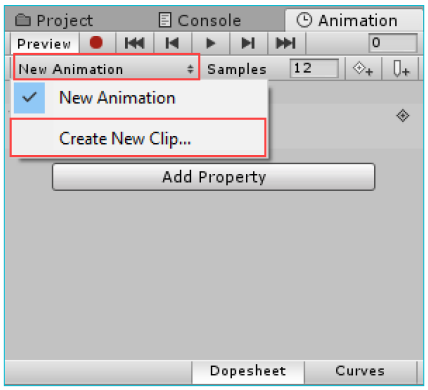


Figure 30: Create new clip dropdown

3. Name this new animation ‘Coin_Bounce’ and save it in the Animations folder. The Animation editor will switch to the new clip, ready for animation.

4. Unity utilizes automatic keyframing, meaning that once in Animation Mode, any movement applied to the selected object is recorded into the animation. Click the red Record button to enter Animation Mode.

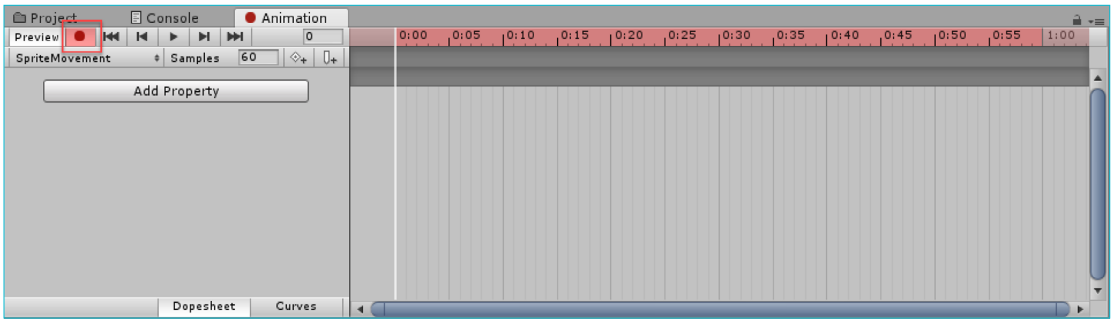


Figure 31: Note how both the Record button and the Timeline turn red to signify Animation Mode is active.

5. The white line in the Timeline is the Playhead, and indicates the point at which a keyframe will be added. To move it, left-click and drag. Advance the Playback head to 0:20.

6. In the Scene view, drag the Coin up a small distance. Note in the Animation editor that the Sprite properties appear on the left, and in the Timeline, keyframes appear both at 0:00 and 0:20.

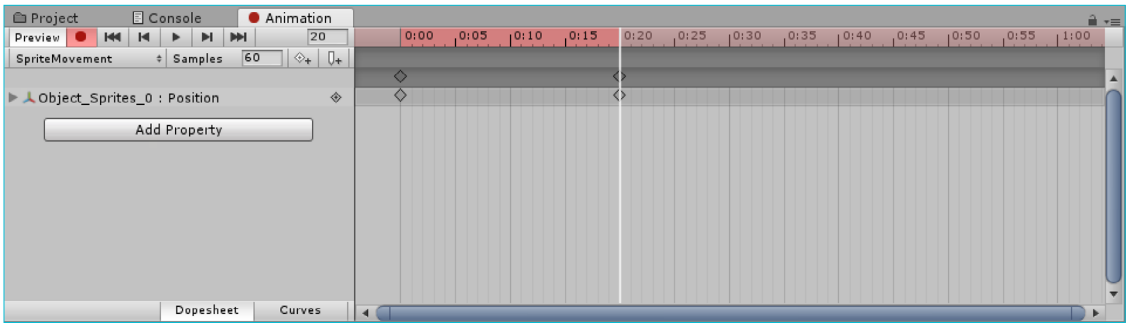


Figure 32: Keyframes are automatically added whenever the selected object is moved while in Animation mode.

7. Click the Record button to exit Animation Mode. Play-test the game and you will see that only the original animation continues to play.

Working with the Animator

To combine multiple animations together, they must be configured in the Animator.

1. Select the Animator window from the Window dropdown. Dock it behind the Animation window for ease of use. The Animator workspace can be moved by holding ALT + left-clicking and dragging.

2. Notice that when the animated sprite is selected, two nodes are visible in the workspace.

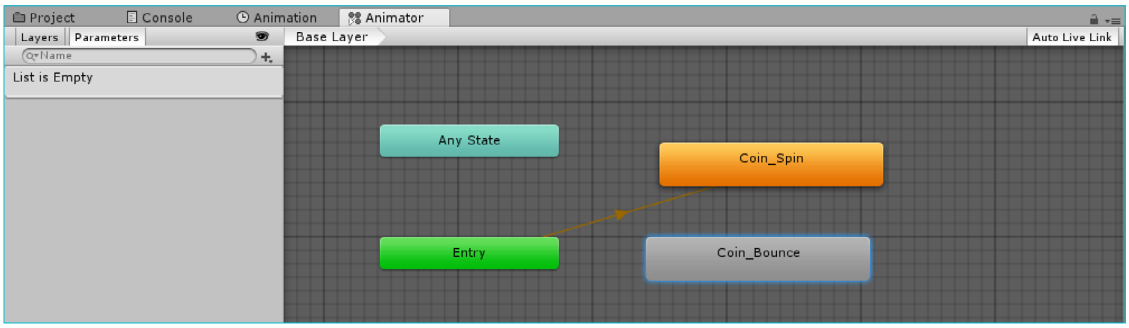


Figure 33: Both animations that have been created for the sprite appear in the Animator

Each node represents an animation associated with the selected sprite. The orange node is the default animation, and will play as soon as the game is active. The default animation can be changed by right-clicking the desired node and choosing “Set as Layer default state”.

3. Select the non-default node and press Delete to remove it from the workspace. Because both animations should run at the same time, it is necessary to create a second layer in the Animator.

4. Click the Layers tab on the upper-left corner of the Animator window. Above the Base Layer, click the + button to add a new layer.
   Name this new layer “Movement”.

5. Click the Movement layer to make it active, then locate the Coin_Bounce animation from the Project window. Drag it into the Movement layer.

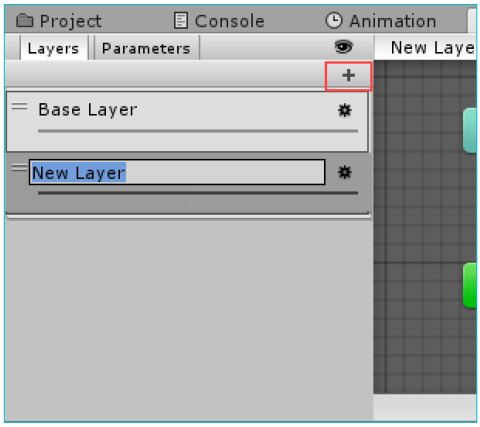


Figure 33: The Layers tab, ‘add new layer’ button highlighted.

6. Click the Gear icon on the right side of the Movement layer see its options. Set the Blending mode to Additive, and the Weight to 1. This will allow the Movement Layer to combine its animation with the Base Layer.

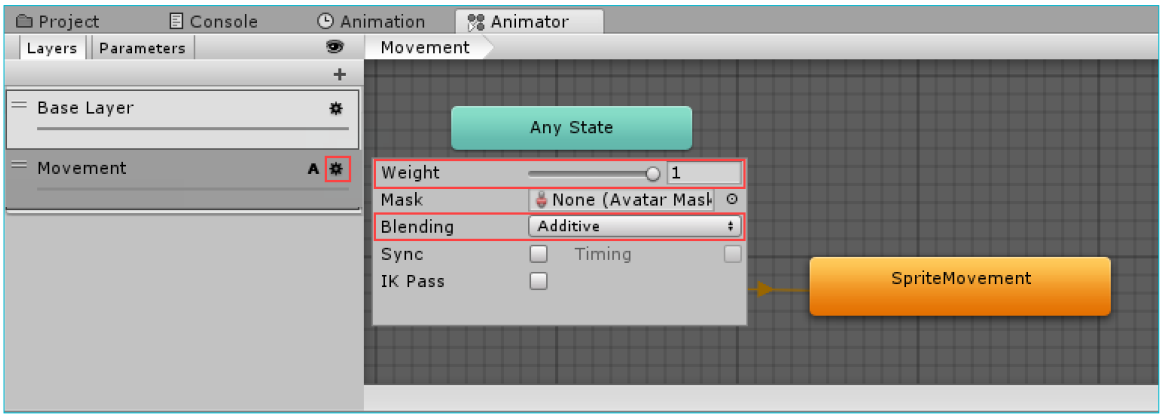


Figure 34: Movement layer configuration. Note that Weight is set to 1 and Blending is set to Additive.

7. Play-test the game to see both animations playing together.

Generating Player character animations

1. Select the Character_Moves sheet and, in the Inspector, set the Sprite Mode to Multiple.

2. Open the Sprite Editor and set the Slice mode Type to Grid by Cell Count, and set Columns (C) to 4, and Rows (R) to 2. Close the Sprite Editor and apply the changes when prompted.

3. Expand the Character_Moves sheet. Select sprites 0-2 and drag them into the Hierarchy. When prompted, name the animation Player_Idle and save it in the Animations folder.

4. In the Hierarchy, rename Character_Moves_0 to “Player”. Select it and, in the Inspector, set its Tag to Player.

5. Return to the Character_Moves and select 4-6. Drag them on top of the Player sprite in the Hierarchy. This prevents an additional Animator Controller from being generated.

6. Save the animation as Player_Run.

7. The final animation only contains a single keyframe, so it isn’t possible to simply drag and drop the sprite to create an animation. In the Animation window, click on the dropdown and select ‘Create New Clip’.

8, Save the animation as Player_Jump.

9. Click on the record button to enter Animation mode. While Animation mode is active, drag the Character_Moves sprite 3 into the Sprite slot in the Inspector.

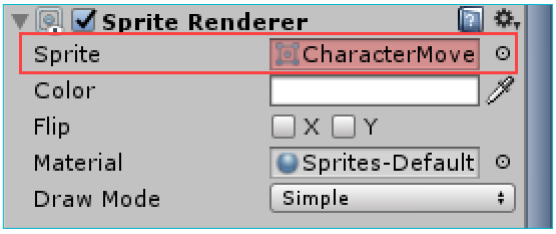


Figure 35: Drag sprite 3 into the Sprite slot of the Sprite Renderer. Note that the Sprite slot turns red to indicate it is keyframed.

10. Exit animation mode. Note that the Player appears to remain in the jump position. Switch the currently viewed animation to Player_Idle and the player will return to standing.

Creating a Player character controller

1. From the Hierarchy, select the Player and locate the Animator section in the Inspector. Double-click the Controller to open the Animator window.

2. All three animations should be visible in the Animator window. If any are missing, locate them in the Project window and drag them into the Animator.

3. Click the Parameters tab in the upper left of the Animator window. Click the + button to reveal the Parameter creation options.

4. Create a Trigger and name it Start, spelled exactly as written.

5. Create a Trigger and name it Jump, spelled exactly as written.

6. Create a Bool and name it Grounded, spelled exactly as written.

7. Evenly space the animation nodes in the Animator workspace.

8. Right-click the Player_Idle node and select Make Transition. A white line is attached to the mouse pointer.

9. Left-click the Player_Run node to complete the Transition.

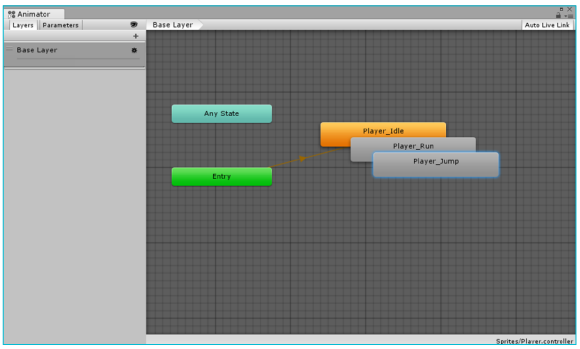


Figure 36: All three animations present in the Animator. Note that Player_Idle is set to default.

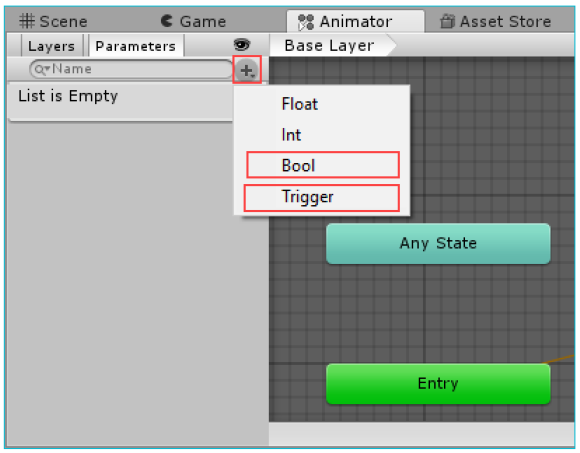
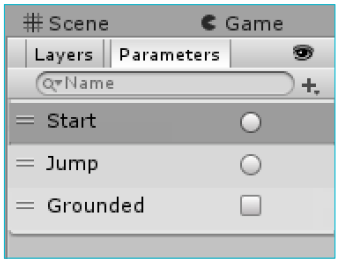


Figure 37(above): Animator Parameters to be used

Figure 38(side): Properly configured Animator Parameters

Note that Bools are displayed as square checkboxes while Triggers are round radio buttons.



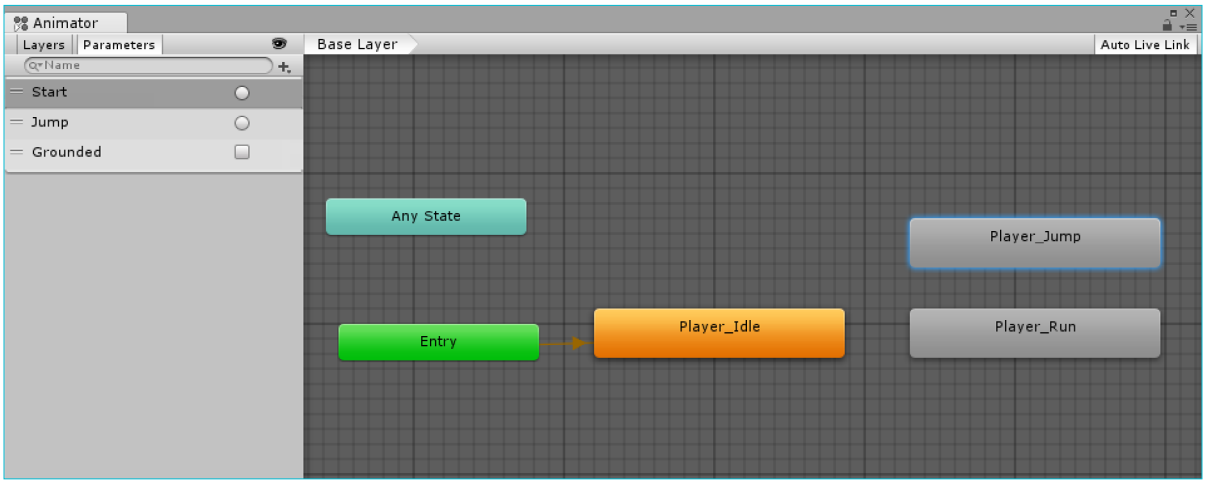


Figure 39: Space animation nodes so they're all easily visible and there's room to create transitions in the next step.

10. Select the Transition and, in the Inspector, uncheck Has Exit Time. In the Conditions section, click the + button to create a new condition, and select Start from the dropdown.

11. Select the Transition and, in the Inspector, uncheck Has Exit Time. In the Conditions section, click the + button to create a new condition, and select Start from the dropdown.

12. Select the Transition and, in the Inspector, uncheck Has Exit Time. In the Conditions section, click the + button to create a new condition, and select Jump.

13. Create a Transition from Player_Jump to Player_Run.

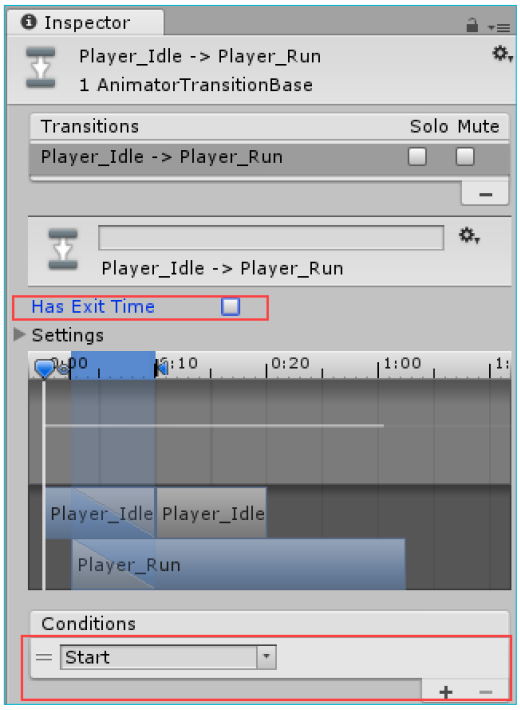


Figure 40: Has Exit Time Unchecked and Conditions set to Start.

14. Select the Transition and, in the Inspector, leave the Has Exit Time option **checked**. Under Conditions, click the + button to create a new condition. Select Grounded and set it to True.

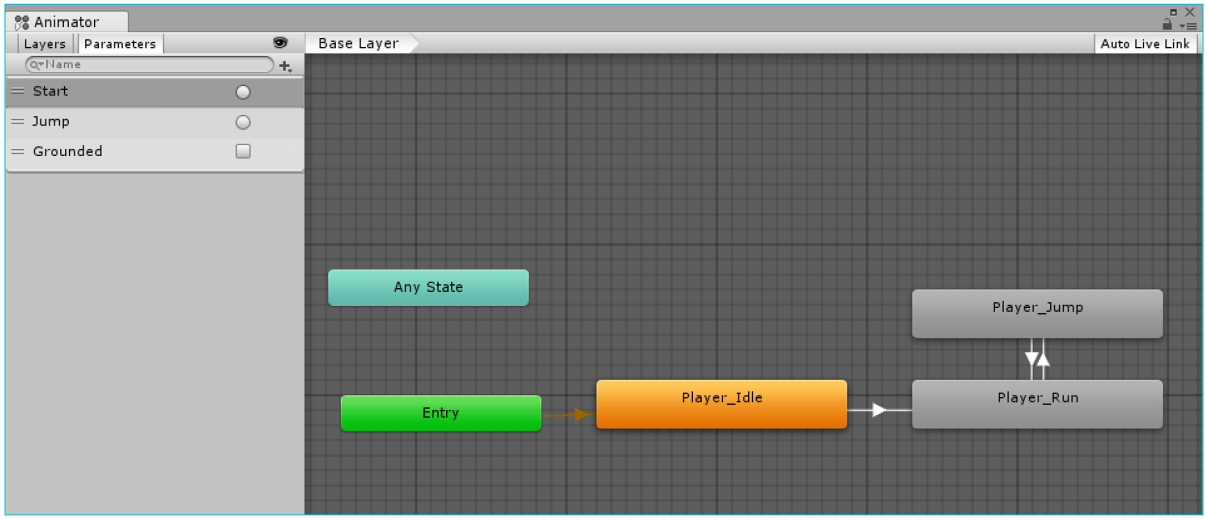


Figure 41: Completed Transition layout

Creating the remaining requirements of the controller

1. Select the Player in the Hierarchy. In the Inspector, click the Add Component button, then go to Physics 2D, and choose Capsule Collider 2D.

2. Click the Edit Collider button and use the handles that appear on the Collider to adjust it to more closely fit the Player sprite, paying close attention to align it with the feet

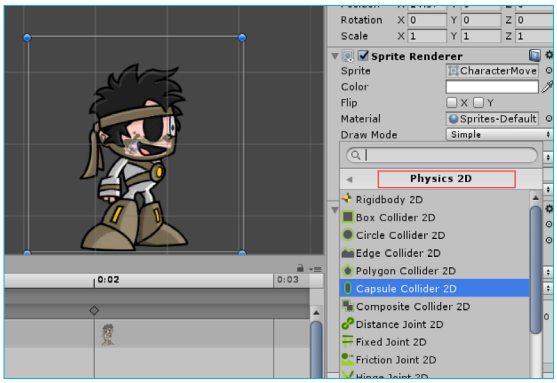


Figure 42: Apply a Circle Capsule2D to the Player.

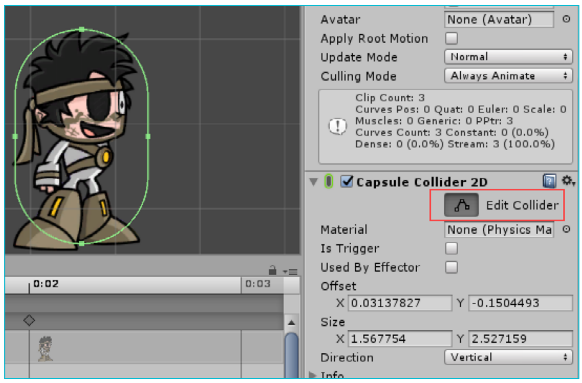


Figure 43: Use the Edit Collider button to align the Capsule to better fit the Player, leaving some space to account for the Player's run sprite.

3. Click the Add Component button in the Inspector then go to Physics 2D and select Rigidbody 2D.

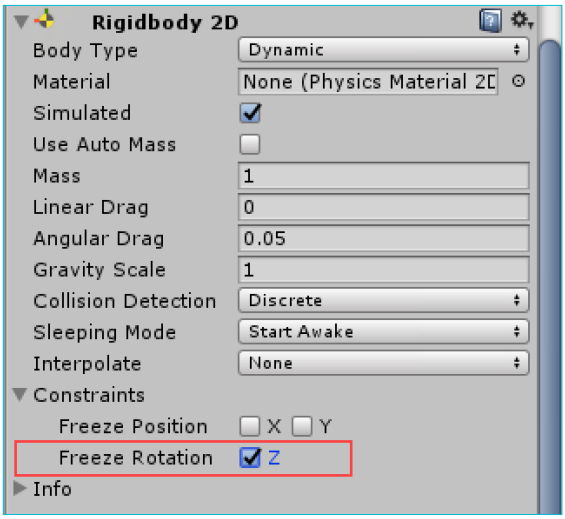


Figure 44: Freezing Rotation Z. This prevents the Player from toppling over when

4. In the Rigidbody 2D component, click the Constraints roll-down, and check Freeze Rotation Z.

5. Navigate to the Scripts folder in the Project window. Drag the Player script onto the Player in the Hierarchy window.

Creating the Player camera

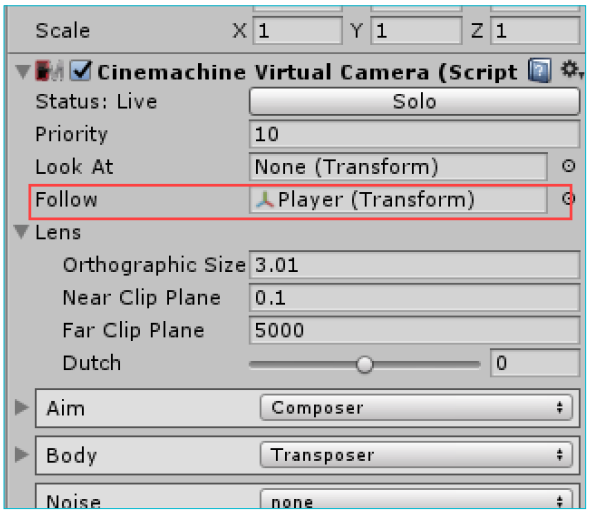


Figure 45: Adding the Player to the Follow slot in the Virtual Camera

1. Click the Cinemachine dropdown at the top of the Editor and choose Create Virtual Camera

2. Select CM vcam1 from the Hierarchy and, in the Inspector, drag the Player into the Follow slot.

3. Adjust the Orthographic Size until the Player takes up the desired amount of space in the Game window.

Activity 3:

Creating game mechanics

Creating a reusable Prefab for winning and losing

Regardless of whether the player wins by reaching the end of the level or loses by falling off the platform, the same core mechanic remains the same: the game will restart, placing the Player back at the beginning of the level.

1. Right-click in the Hierarchy and select Create Empty to add an empty GameObject into the scene. Rename it “Restart_Zone”.

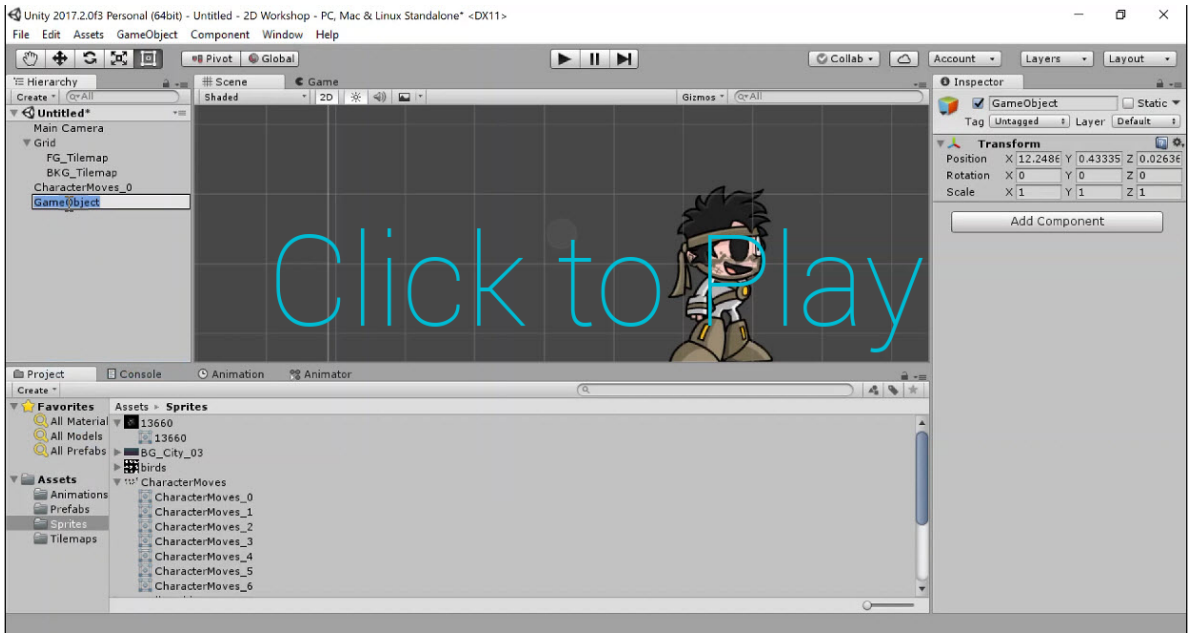


Figure 46: Creating an empty GameObject in the Hierarchy

2. Add a 2D Box Collider to Restart_Zone from the Add Component button in the Inspector.

3. Check the Is Trigger box on the Box Collider.

4. Navigate to the Scripts folder in the Project window and create a new C# Script by right-clicking and selecting C# Script from the Create flyout.

5. Name the script Restart. Double-click the script to open it in Visual Studio.

6. Delete lines 7-15, so the Script appears as below:

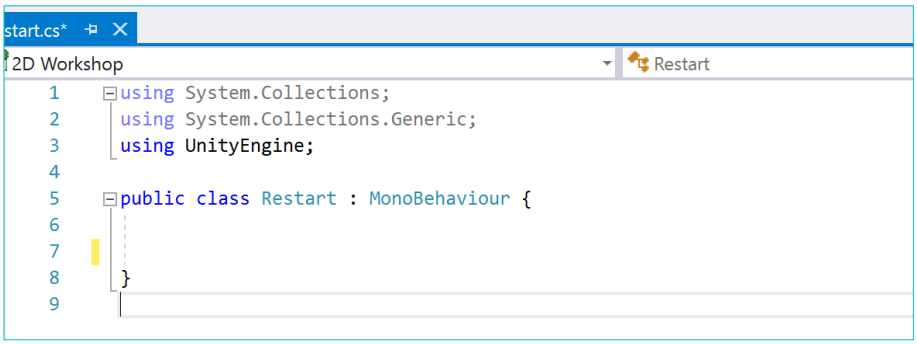


Figure 47: Start and Update functions removed from the Restart script.

7. Type the following code, taking special care with spelling and spacing. If errors occur, closely compare your code with what is seen below.

```
System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;

public class Restart : MonoBehaviour {

    void OnTriggerEnter2D (Collider2D other)
    {
        if (other.gameObject.CompareTag ("Player")) {

            SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex);

        }
    }

}
```

8. Save the file and return to Unity.

9. Select the Restart_Zone and drag the Restart script from the Project window into the Inspector.

10. Move the Restart_Zone to an area where the Player is able to run into it and play-test the level. Once the Player has hit the Restart_Zone, the game should reload.

11. Drag the Restart_Zone into the Prefabs folder in the Project window.

Creating the Lose Zone

1. Move the Restart_Zone already in the scene to below the platforms where the Player will fall.
2. Edit the Box Collider so that it reaches across the entire width of the Scene.

Creating the Win Zone

1. Drag a new Restart_Zone Prefab from the Project window and place it at the end of the Scene.
2. Edit the Box Collider so it is approximately 2 to 3 times taller than the Player.
3. From the Prefabs folder, drag out the Victory_Particles and drop them on top of the new Restart_Zone to make them a child of the Prefab.

Preparing the collectible Prefab

1. Select the collectible object of your choosing from the Sprites folder in the Project window and drag it into the Scene.

2. Rename the sprite to “[item name]_Collectible”.

3. Apply a 2D Box Collider to the sprite from the Add Component button in the Inspector.

4. Check the Is Trigger box on the Collider.

5. Apply the CollectibleObject script to the sprite. Note that this automatically adds an AudioSource to the collectible.

6. Uncheck the Play on Awake option on the AudioSource.

7. From the Sounds folder in the Project window, select the collectible sound effect of your choosing and drag it into the Sound slot on the CollectibleObject script.

8. Drag the sprite into the Prefabs folder in the Project window to make it a Prefab.

9. Add several Collectible Sprite Prefabs to your Scene as defined by the level design.

Activity 4:

Finalizing the game

Designing the UI

The primary purpose of the user interface is informing the player what their score is, based on how many items they've collected. The remaining elements of the UI are purely there for visual effect.

1. In the Hierarchy, right-click and, from the UI flyout, select Canvas

   All UI elements must be children of the Canvas object, otherwise they will not be visible in the game.

2. Right-click the Canvas object and create a UI Text object. Place it in the upper-right corner of the Canvas.

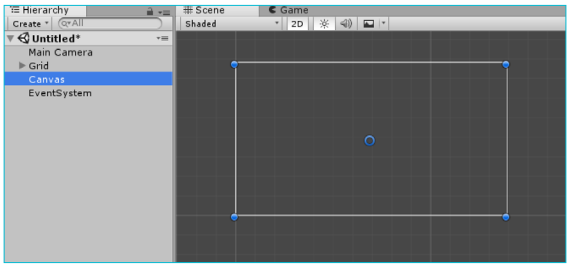


Figure 48: The canvas is significantly larger than the rest of the Scene.

3. Adjust the Anchor of the UI Text object by clicking the Anchor Presets in the Inspector and selecting Upper Right. This will ensure that the text remains in the proper position regardless of what screen the game is played on.

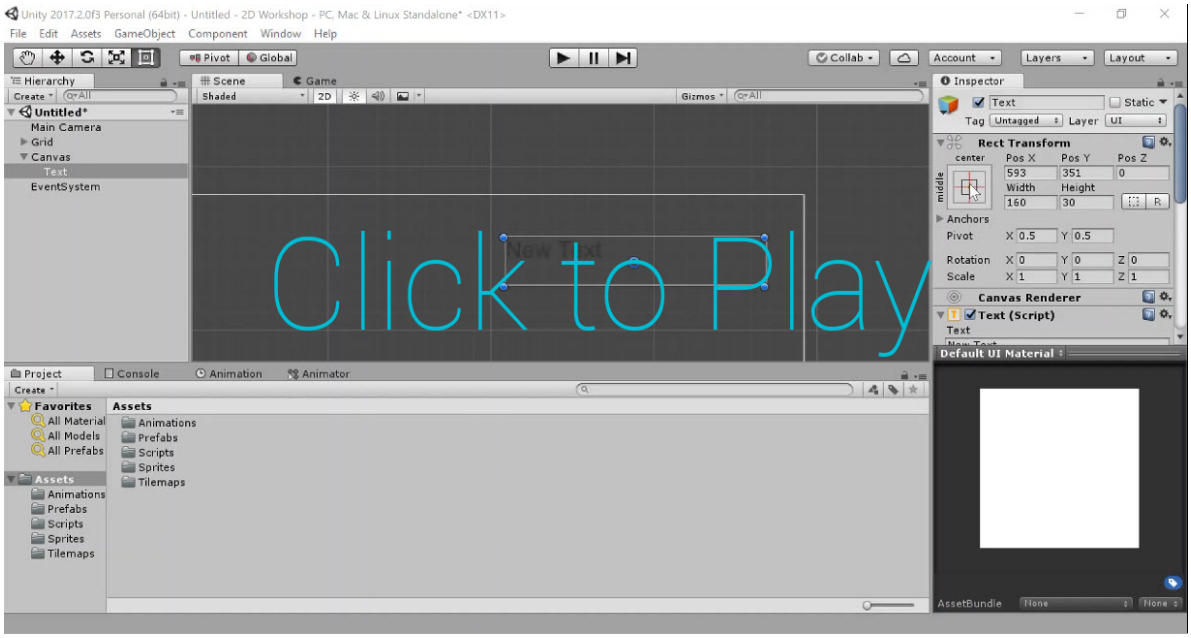


Figure 49: Adjusting the Text anchor using the Anchor Presets

4. Rename the UI Text to “Score_Counter”.

5. Right-click the Canvas object again, and add a UI image. Place the image in the lower-left side of the canvas and set its Anchor Preset to lower left.

6. Rename the UI image to “Player_Image”.

7. Locate the Player_Profile sprite in the Sprites folder. Reselect Player_Image and drag the Player_Profile sprite into the Source Image slot in the Inspector.

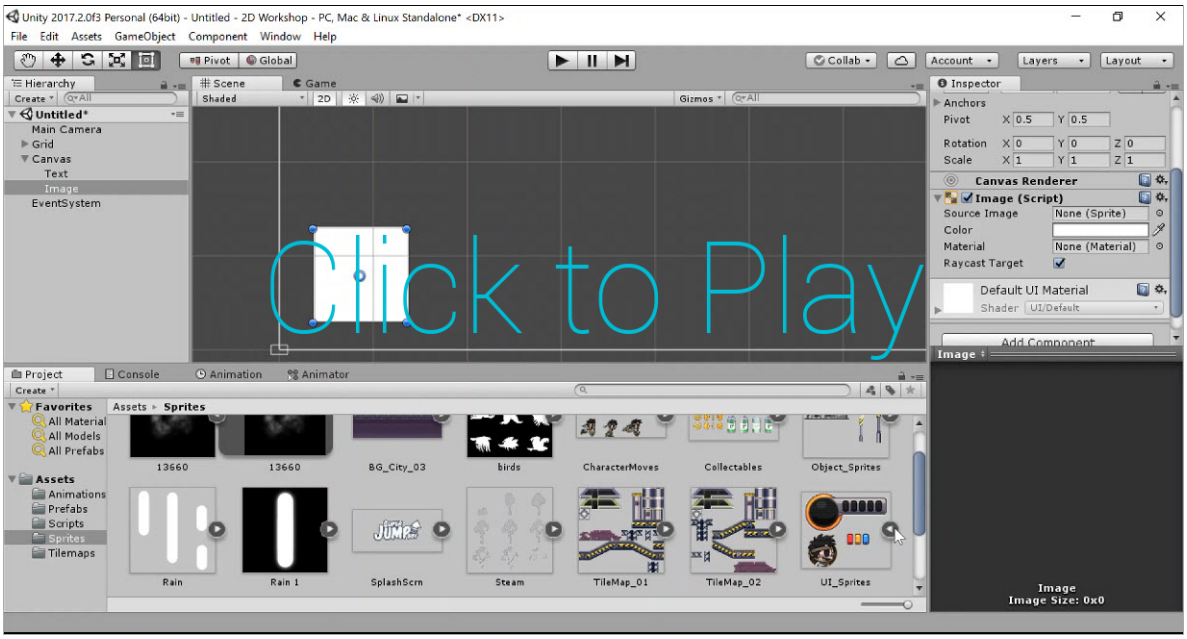


Figure 50: Applying the Source image to the UI Image and setting its native size`

8. Apply the UI script to the Canvas.

9. Drag the ScoreCounter Text object into the Score slot on the UI script.

Creating a global AudioSource

1. Create a game empty in the Hierarchy and name it “Audio_Manager”.

2. In the Inspector, add an icon to allow the Audio_Manager to be selected from the Scene view.

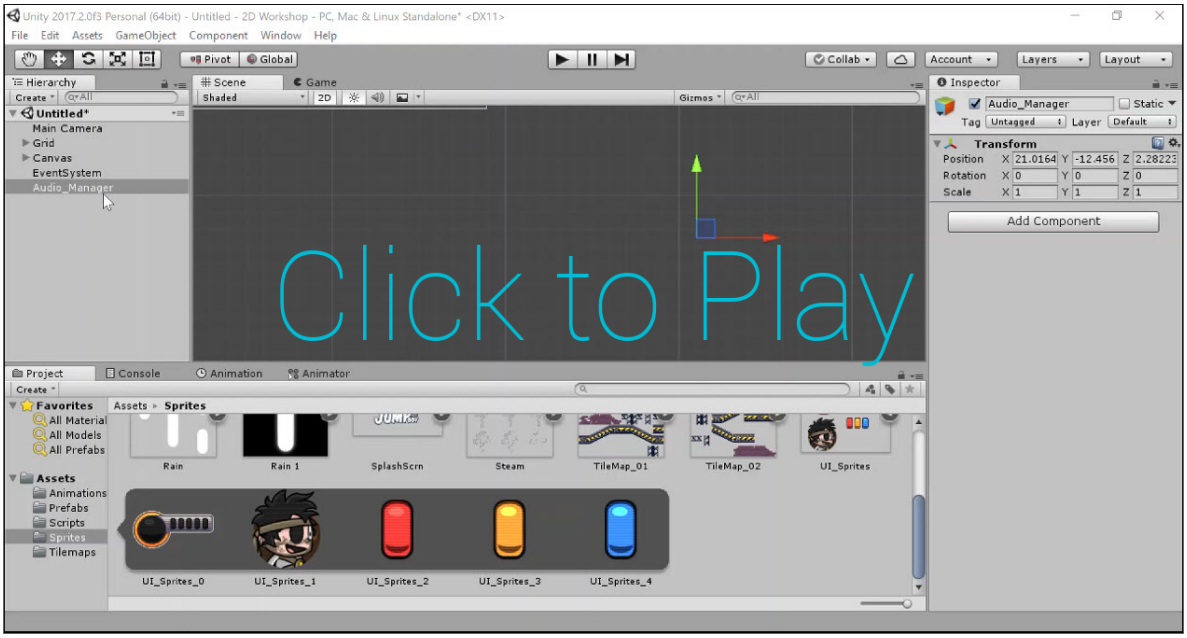


Figure 51: Adding an Icon to the Audio_Manager to give it visibility in the Scene view.

3. Add an AudioSource to the Audio_Manager by clicking the Add Component button in the Inspector, and selecting it from the Audio section.

4. Click the Loop box in the Audio Inspector.

5. Set the Volume to something low, such as 0.5.

6. Drag the background music of your choice from the Sounds folder into the AudioClip slot.

Publishing for PC/Mac

Make sure the game scene has been saved before building and publishing!

1. From the File menu, select Build Settings. At the top of the window, click Add Open Scene.
2. Select Windows or Mac as the Target Platform based on your preferences.
3. At the bottom of the window, select Player Settings.
4. Drag the Icon image from the Images folder and apply it to the Default Icon slot in Player Settings
5. Click the Splash Image dropdown to show the image slot for the Application Config Dialog.
6. Locate the Splash_Screen image in the Images folder and drag it into the Application Config Dialog slot.
7. Click Build and Run in the Build Settings window.
8. When prompted, save the file to the Desktop.

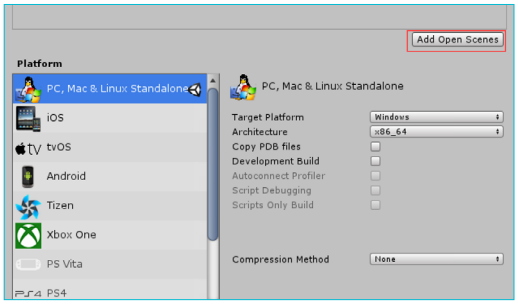


Figure 52: Clicking on the Add Open Scenes button will load the current scene into the build

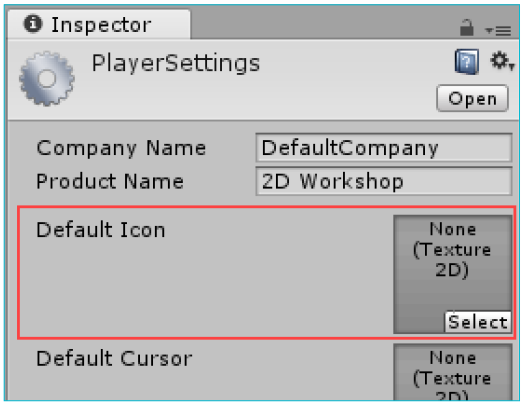


Figure 53: The default icon will be what is visible on the game .exe

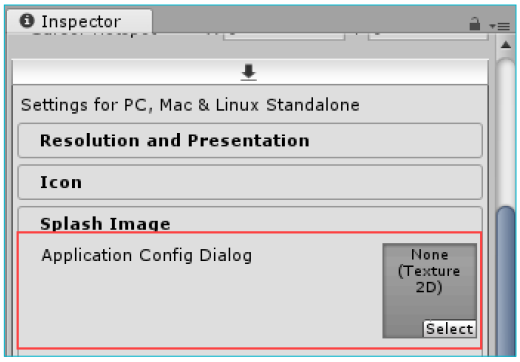


Figure 54: The Splash image appears as the game's banner image on the configuration screen that first appears when the application is launched.

For PC Games

In addition to an executable file, a data folder is generated. In order to run the game, these two items must be distributed together. The data folder should not be named, nor should the executable be placed inside of it – both of these actions will prevent the game from running.

Setting up Android devices for builds

In order to publish games to your Android device for testing, it must first be set to Developer mode. The process for getting to the Settings menu to enable this option will vary slightly from device to device, but the actual process of setting Developer mode remains the same. Please be aware that some apps (usually banking related) will detect Developer mode and warn you that it may impact how the software runs. If you are concerned, you can turn off Developer options at any time from your device's System menu.

1. Navigate to your device's Settings options.

2. Tap on About Phone and locate the Build Number.

3. Tap on the Build Number seven times. After a few taps you will see a pop-up that states you're halfway to becoming a developer. When completed, you will be notified that Developer mode is enabled.

Publishing for Android

Publishing for Android requires downloading the Android Software Development Kit (SDK), the Java Development Kit (JDK), and the appropriate USB driver for your device. The SDK is a large file and will take some time to fully download. You can find an in-depth tutorial for configuring your device and Unity for Android here: Building your Unity game to an Android device for testing.

1. Download and install the necessary USB drivers for your device. Refer to instructions from Android here: Install OEM USB Drivers.

2. From the Edit dropdown in Unity, select Preferences.

3. Scroll to the bottom of the window to the Android section.

4. Click the Download button for SDK. As described in the linked tutorial above, download the Command Line Tools and update the SDK packages appropriate for your device with the SDK Manager. **Take special note of where you saved these files on your computer.**

5. Click the Download button for JDK. On the Oracle website, click Download JDK and select the correct version for your computer. **Take special note of where you save these files on your computer.**

6. Back in Unity, click the Browse button next to SDK and navigate to where you saved the SDK.

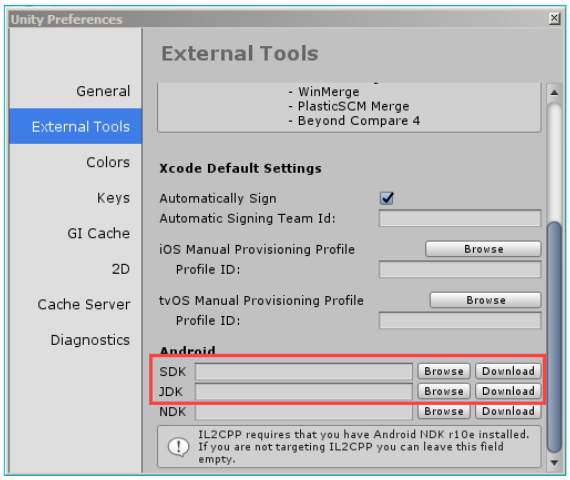


Figure 55: The SDK and JDK paths must be defined prior to publishing.

7. Browse for and set the location of the JDK.

8. From the File menu, select Build Settings.

9. Select Android from the list and, at the bottom of the window, select Switch Platform. The project will recompile for Android.

10. Click Player Settings. Scroll to the bottom and open Other Settings.

11. Configure your bundle identifier as com.unity.2dWorkshop.

12. Drag the Icon image from the Images folder and apply it to the Default Icon slot in Player Settings.

13. Connect your device to your computer via USB.

14. In the Build Settings window, click Build and Run.

15. Once the build is complete, the game will launch on your device.

Publishing for iOS

Publishing for iOS is outside of the scope of this workshop as it requires setting up a developer account with Apple before configuring devices for testing purposes. If you are interested in learning about the process of building this game out to iOS, please follow this Unity Tutorial: Building your Unity game to an iOS device for testing.

Further reading can be found here: Getting Started with iOS development.

Note: A Mac running OS X 10.11 or later is also required for iOS game output.

Closing

Updating the Learning Action Plan and next steps

Closing

We hope that you have learned a lot by participating in this workshop and that it will help you on your path as a Unity developer. Take some time to update your Learning Action Plan, and be sure to share your experience with your peers!

There are several workshops currently available in addition to the 2D workshop that you have just completed. Check with your local Authorized Training Partner to see what is available in your area.

There are also many official Unity online courses available as well, including:

Swords and Shovels from PluralSight
The Ultimate Guide to Game Development with Unity from Udemy
VR Nanodegree by Udacity