Computer Game Programming (SE3173)

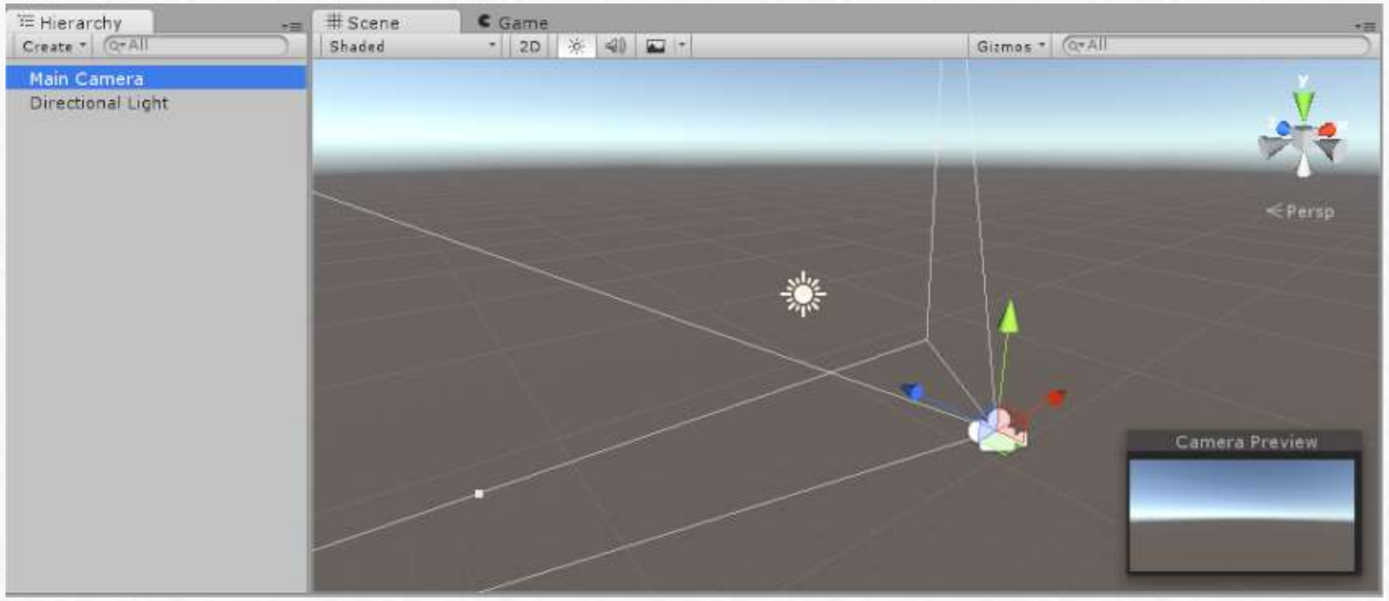
Ibrar Arshad
Ibrar.arshad@cust.edu.pk

Creating Gameplay

- Unity empowers game designers to make games
- What’s really special about Unity is that you don’t need years of experience with code or a degree in art to make fun games
- There are a handful of basic workflow concepts needed to learn Unity
  - Once understood, you will find yourself making games in no time
- With the time you will save getting your games up and running, you will have that much more time to refine, balance, and tweak your game to perfection

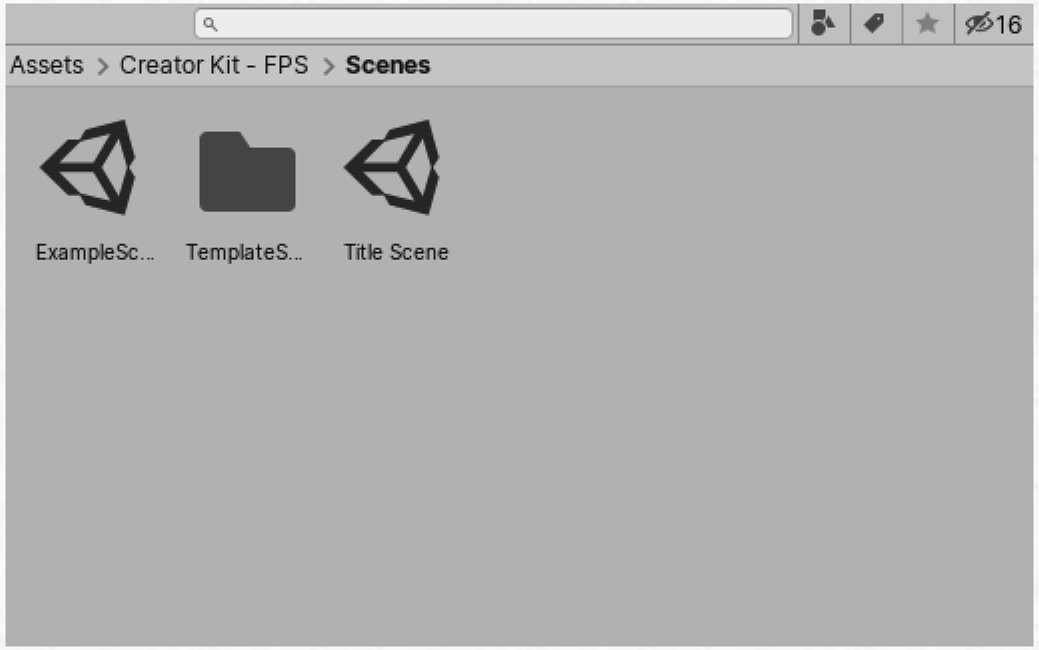
Unity – Scenes

- Scenes are where you work with content in Unity
- They are the assets that contain all or part of a game or application
  - For example, you might build a simple game in a single scene, while for a more complex game, you might use one scene per level, each with its own environments, characters, obstacles, decorations, and **UI**
- You can create any number of scenes in a project.



Creating a New Scene

- Use the New Scene Dialog to create a new scene from a specific scene template
- By default, the New Scene dialog opens when you create a new scene from the menu (File > New Scene)
- To save the scene you're currently working on, choose File > Save Scene from the menu



GameObjects

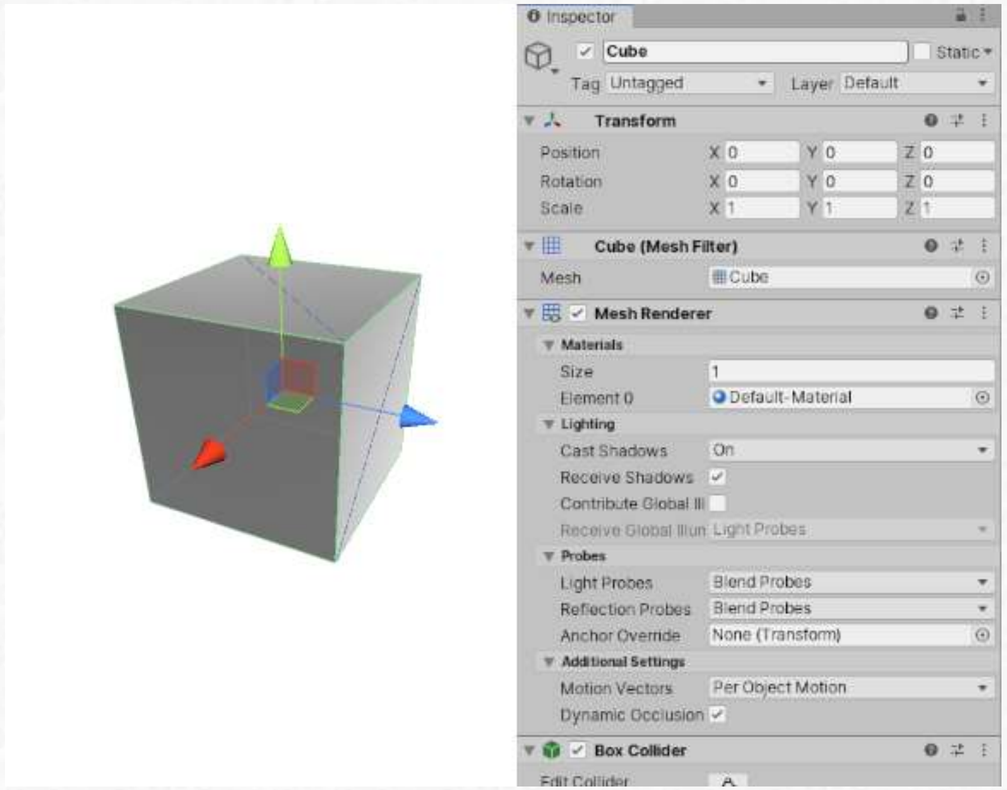
- The GameObject is the most important concept in the Unity Editor.
- Every object in your game is a GameObject, from characters and collectible items to lights, Cameras and special effects
- However, a GameObject can’t do anything on its own; you need to give it properties before it can become a character, an environment, or a special effect.

GameObjects

- GameObjects are the fundamental objects in Unity that represent characters, props and scenery
  - They do not accomplish much in themselves, but they act as containers for **Components**, which implement the functionality
- To give a GameObject the properties it needs to become a light, or a tree, or a camera, you need to add components to it
  - Depending on what kind of object you want to create, you add different combinations of components to a GameObject.
  - Unity has lots of different built-in component types

GameObjects (Solid Cube)

- A solid cube object has a Mesh Filter and Mesh Renderer component, to draw the surface of the cube, and a Box Collider component to represent the object's solid volume in terms of physics.



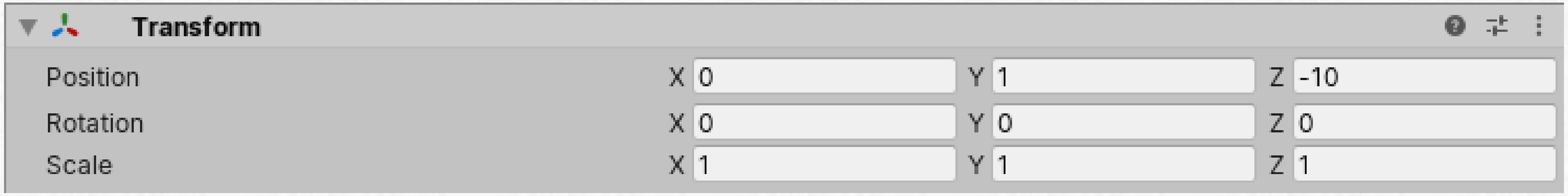
```
Inspector
✓ Cube                                  Static ▾
  Tag Untagged ▾          Layer Default ▾
▼ Transform
  Position        X 0     Y 0     Z 0
  Rotation        X 0     Y 0     Z 0
  Scale           X 1     Y 1     Z 1
▼ Cube (Mesh Filter)
  Mesh            Cube
▼ ✓ Mesh Renderer
  ▼ Materials
    Size                      1
    Element 0                 Default-Material
  ▼ Lighting
    Cast Shadows              On ▾
    Receive Shadows           ✓
    Contribute Global Ill
    Receive Global Illun      Light Probes ▾
  ▼ Probes
    Light Probes              Blend Probes ▾
    Reflection Probes         Blend Probes ▾
    Anchor Override           None (Transform)
  ▼ Additional Settings
    Motion Vectors            Per Object Motion ▾
    Dynamic Occlusion         ✓
▼ ✓ Box Collider
  Edit Collider
```

GameObjects Components

- A GameObject always has a Transform component attached (to represent position and orientation) and it is not possible to remove this.
- The other components that give the object its functionality can be added from the editor’s Component menu

Transforms

- The Transform is used to store a GameObject's position, rotation, scale and parenting state and is thus very important
- A GameObject will always have a Transform component attached - it is not possible to remove a Transform or to create a GameObject without one.
- The Transform component determines the Position, Rotation, and Scale of each object in the scene
- Every GameObject has a Transform.



Transform

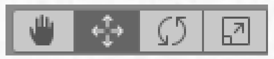
Position	X	0	Y	1	Z	-10
Rotation	X	0	Y	0	Z	0
Scale	X	1	Y	1	Z	1

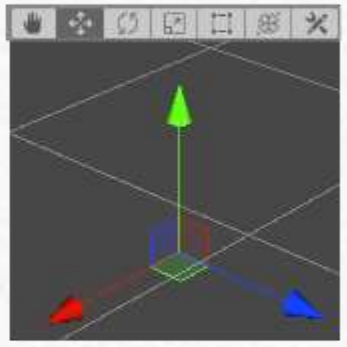
Transforms Properties

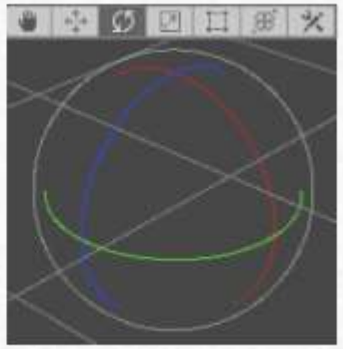
Property:	Function:
Position	Position of the Transform in X, Y, and Z coordinates.
Rotation	Rotation of the Transform around the X, Y, and Z axes, measured in degrees.
Scale	Scale of the Transform along X, Y, and Z axes. Value “1” is the original size (size at which the object was imported).

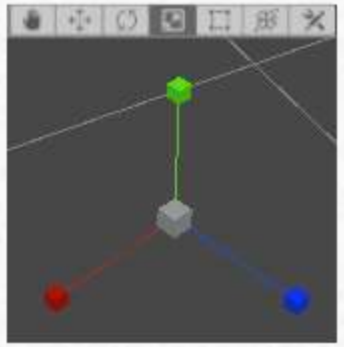
Editing Transform

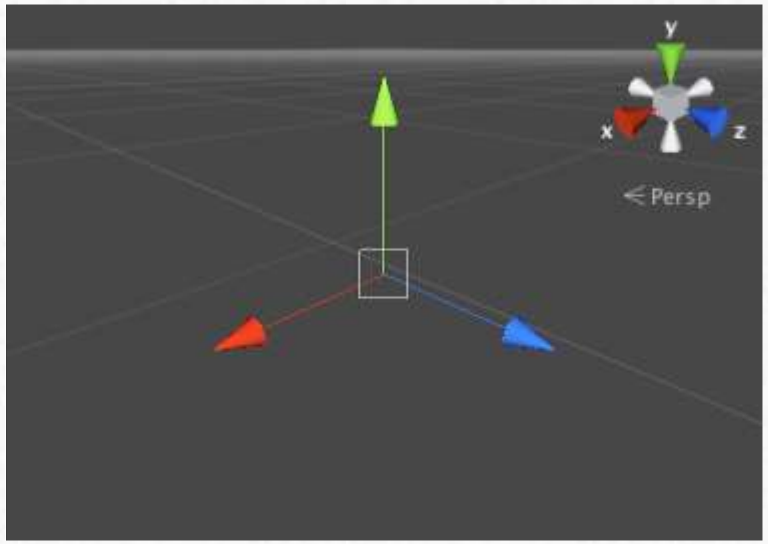
- Transforms are manipulated in 3D space in the X, Y, and Z axes or in 2D space in just X and Y. In Unity, these axes are represented by the colors red, green, and blue respectively.
- A Transform can be edited in the Scene View or by changing its properties in the Inspector
- In the scene, you can modify Transforms using the Move, Rotate and Scale tools. These tools are located in the upper left-hand corner of the Unity Editor.




Translate (W)

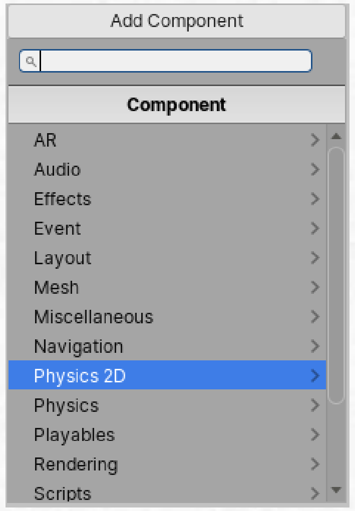

Rotate (E)

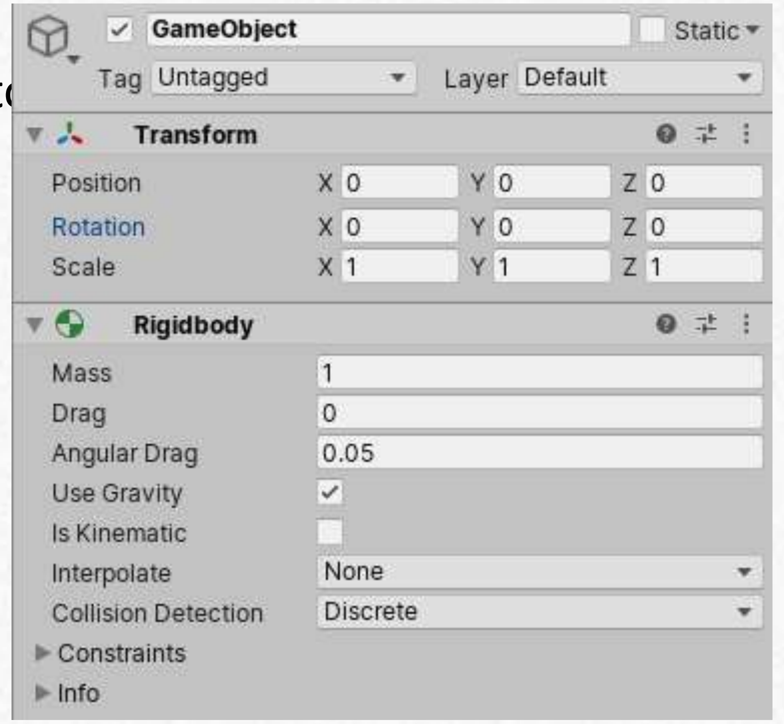

Scale (R)



Using Components

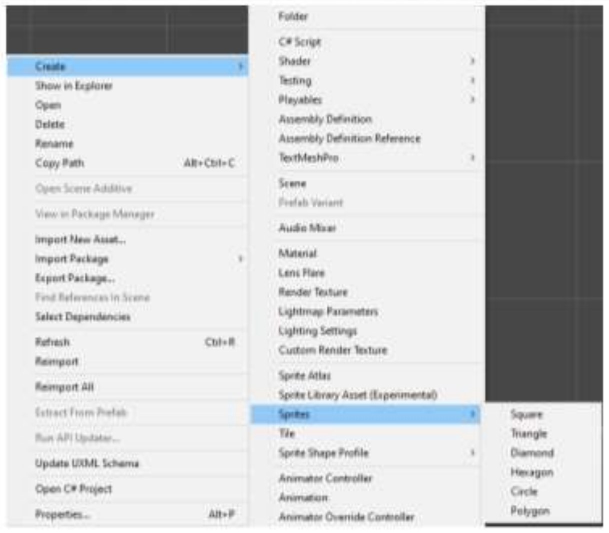
- Components are the nuts & bolts of objects and behaviors in a game
- They are the functional pieces of every **GameObject**
- You can add Components to the selected GameObject through the Components menu
  - Try this now by adding a **Rigidbody** to the empty GameObject we just created
  - Select it and choose **Component->Physics->Rigidbody** from the menu.
  - When you do, you will see the Rigidbody's properties appear in the Inspect

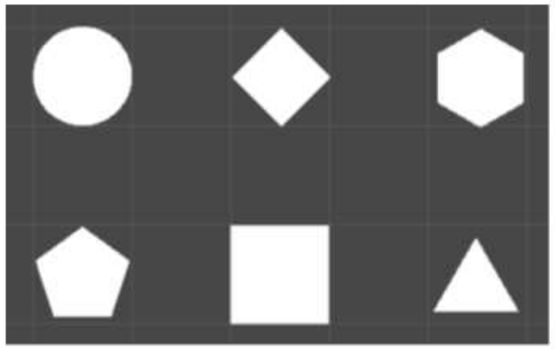



GameObject Static
Tag Untagged Layer Default
Transform
Position X 0 Y 0 Z 0
Rotation X 0 Y 0 Z 0
Scale X 1 Y 1 Z 1
Rigidbody
Mass 1
Drag 0
Angular Drag 0.05
Use Gravity ✓
Is Kinematic
Interpolate None
Collision Detection Discrete
Constraints
Info

2D Primitive GameObjects

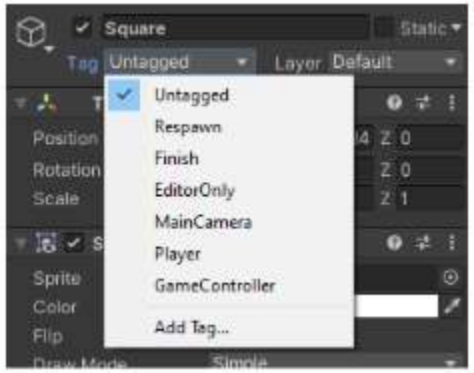
- Unity provides 2D Primitive **GameObjects**
 to help you prototype your Project quickly, without needing to import your own Assets
- To create a 2D primitive, go to **GameObject > 2D Object > Sprites** and select one of the following option
    - Square
    - Circle
    - Capsule
    - Isometric Diamond
    - Hexagon Flat-Top
    - Hexagon Point-Top
    - 9-Sliced

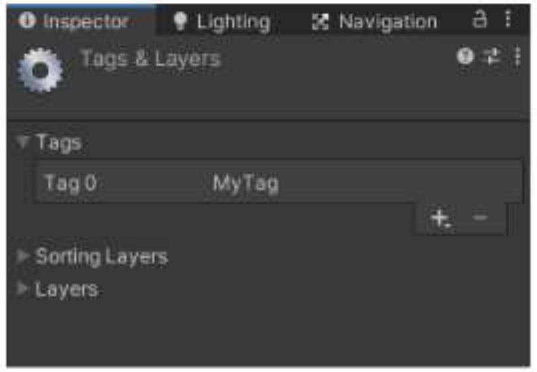




Tags

- A Tag is a reference word which you can assign to one or more GameObjects.
  - Tags are useful to identify GameObjects for scripting purposes: finding an object by its tag, identify collision objects, etc.
  - Once you name a Tag, it cannot be renamed later.
  - A GameObject can only have one Tag assigned to it





Prefabs

- Unity’s **Prefab** system allows you to create, configure, and store a **GameObject** complete with all its components, property values, and child GameObjects as a reusable Asset
- The Prefab Asset acts as a template from which you can create new Prefab instances in the **Scene**
- When you want to reuse a GameObject configured in a particular way – like a non-player character (NPC), prop or piece of scenery – in multiple places in your Scene, or across multiple Scenes in your Project, you should convert it to a Prefab
- This is better than simply copying and pasting the GameObject, because the Prefab system allows you to automatically keep all the copies in sync.

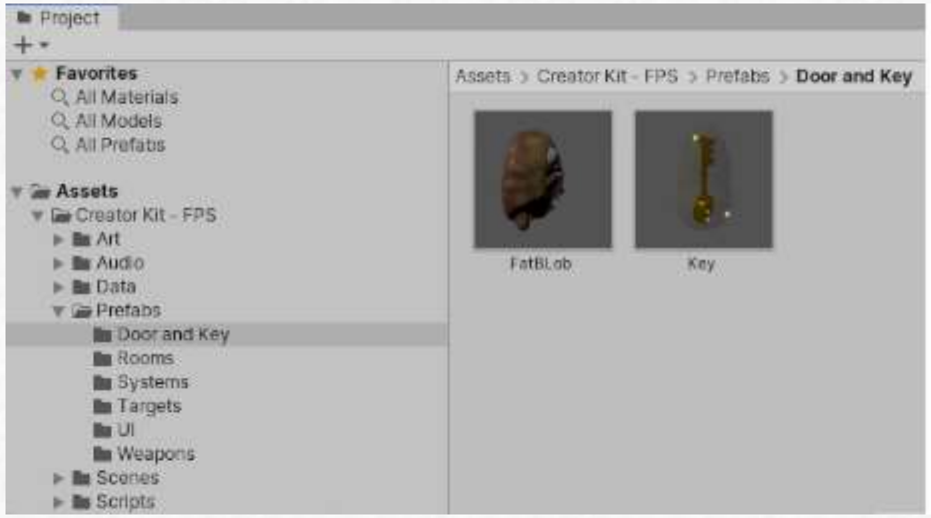
(https://learn.unity.com/tutorial/prefabs-e#5f49e5bcedbc2a280d7cf66f)

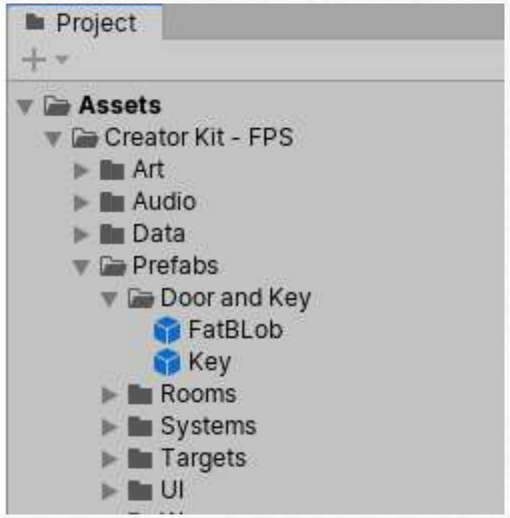
Prefabs Examples

- Some common examples of Prefab use include:
  - Environmental Assets - for example a certain type of tree used multiple times around a level (as seen in the screenshot above).
  - Non-player characters (NPCs) - for example a certain type of robot may appear in your game multiple times, across multiple levels. They may differ (using *overrides*) in the speed they move, or the sound they make.
  - Projectiles - for example a pirate’s cannon might instantiate a cannonball Prefab each time it is fired.
  - The player’s main character - the player prefab might be placed at the starting point on each level (separate Scenes) of your game.

Creating Prefab Assets

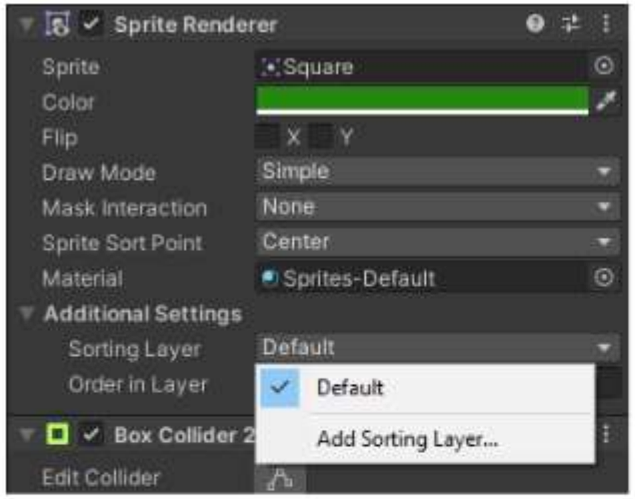
- To create a Prefab Asset, drag a **GameObject** from the Hierarchy window into the Project window
- The GameObject, and all its components and child GameObjects, becomes a new Asset in your Project window
- Prefabs Assets in the Project window are shown with a thumbnail view of the GameObject





Unity – Sorting Layers

- Sorting Layers are used in conjunction with Sprites to define the overlay order of different Sprites.
- To change their order, drag the handle at the left-hand side of each Layer item.
- Change the value of the Order in Layer to set the Renderer's priority among other Renderers within the same Sorting Layer.

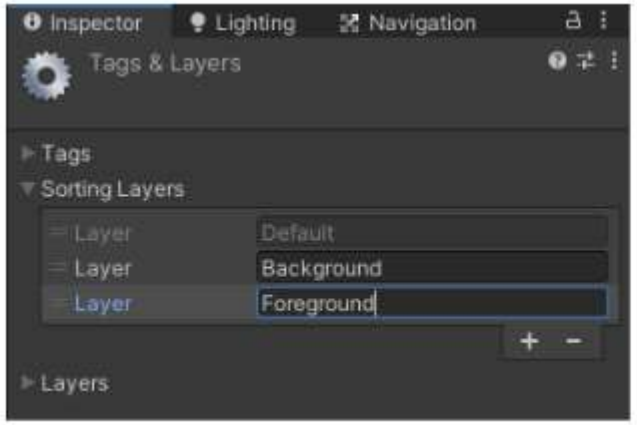


Sprite Renderer

Sprite	Square
Color	🟩
Flip	X Y
Draw Mode	Simple
Mask Interaction	None
Sprite Sort Point	Center
Material	Sprites-Default

Additional Settings

Sorting Layer	Default
Order in Layer	



Sorting Layers

Layer	Default
Layer	Background
Layer	Foreground

Inputs

- Input allows the user to control your application using a device, touch, or gestures. You can program in-app elements, such as the graphic user interface (GUI) or a user **avatar**, to respond to user input in different ways.
- Unity supports input from many types of input devices, including:
  - Keyboards and mice
  - Joysticks
  - Controllers
  - Touch screens
  - Movement-sensing capabilities of mobile devices, such as accelerometers or gyroscopes
  - VR and **AR** controllers

Input Manager

- The **Input Manager** window allows you to define input axes and their associated actions for your Project.
- To access it, from Unity's main menu, go to **Edit > Project Settings**, then select **Input Manager** from the navigation on the right.
- The **Input Manager** uses the following types of controls:
  - **Key** refers to any key on a physical keyboard, such as W, Shift, or the space bar.
  - **Button** refers to any button on a physical controller (for example, gamepads), such as the X button on an Xbox One controller.
  - A **virtual axis** (plural: **axes**) is mapped to a control, such as a button or a key

Mapping virtual axes to controls

Key family	Naming convention
Letter keys	a, b, c…
Number keys	1, 2, 3…
Arrow keys	up, down, left, right
Numpad keys	[1], [2], [3], [+], [equals]…
Modifier keys	right shift, left shift, right ctrl, left ctrl, right alt, left alt, right cmd, left cmd
Special keys	backspace, tab, return, escape, space, delete, enter, insert, home, end, page up, page down
Function keys	f1, f2, f3…
Mouse buttons	mouse 0, mouse 1, mouse 2, and so on.

Accessing the Controls

- You can also query input for a specific key or button with Input.
- GetKey and the naming conventions specified above. For example:

```
Input.GetKey("a");
```

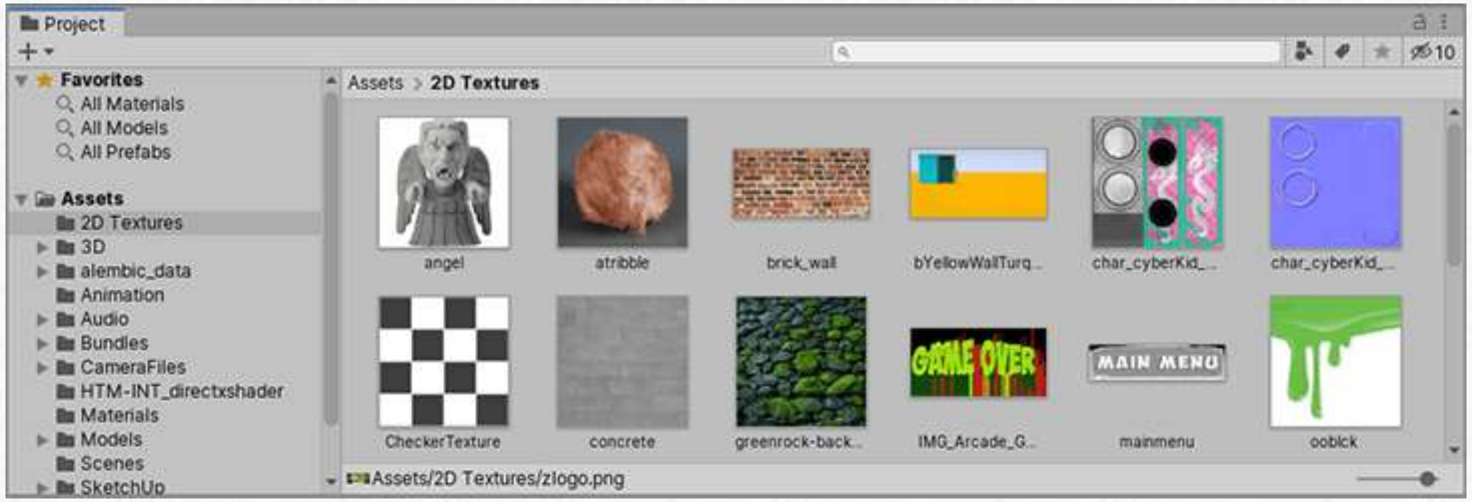
```
float horizontalInput = Input.GetAxis ("Horizontal");
```

Asset Workflow

- An asset is any item that you use in your Unity project to create your game or app. Assets can represent visual or audio elements in your project, such as 3D models, textures, **sprites**, sound effects, or music.
- Assets can also represent more abstract items such as color gradients, animation masks, or arbitrary text or numeric data for any use.
- An asset might come from a file created outside of Unity, such as a 3D Model, an audio file, or an image

Importing Assets

- You can bring assets created outside of Unity into your Unity project. To do this, you can either export the file directly into the Assets folder for your project, or copy it into that folder
- For many common formats, you can save your source file directly into your project's Assets folder and Unity can read it
- Unity also detects when you save new changes to the file and re-imports files as necessary.

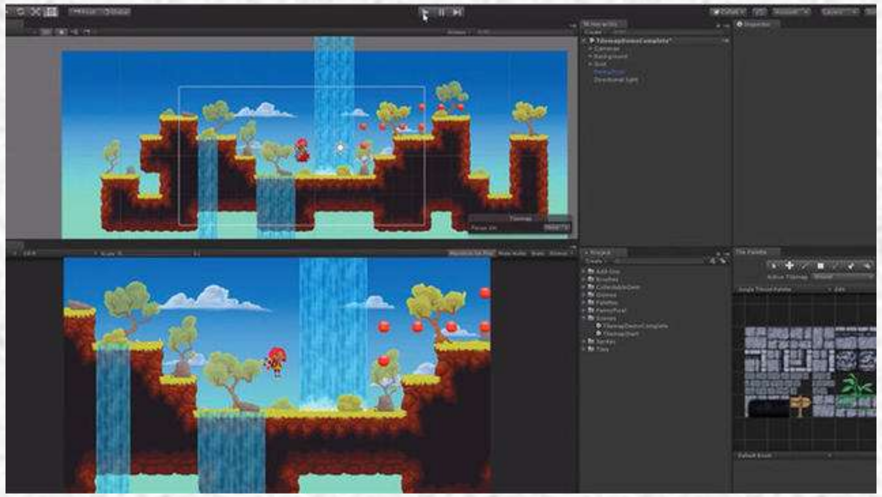


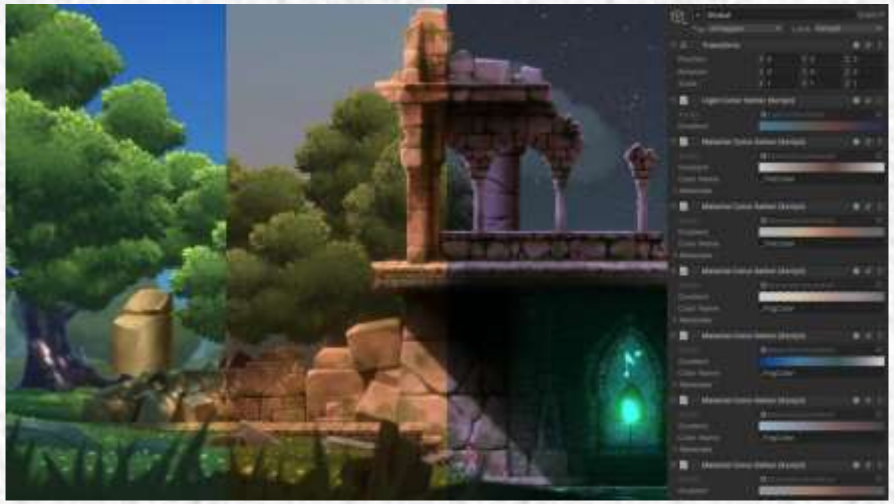
Computer Game Programming (SE3173)

Ibrar Arshad
Ibrar.arshad@cust.edu.pk

Gameplay in 2D

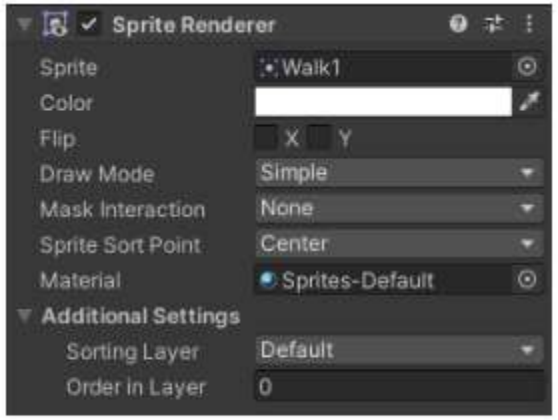
- While famous for its 3D capabilities, Unity can also be used to create 2D games
- The familiar functions of the editor are still available but with helpful additions to simplify 2D development



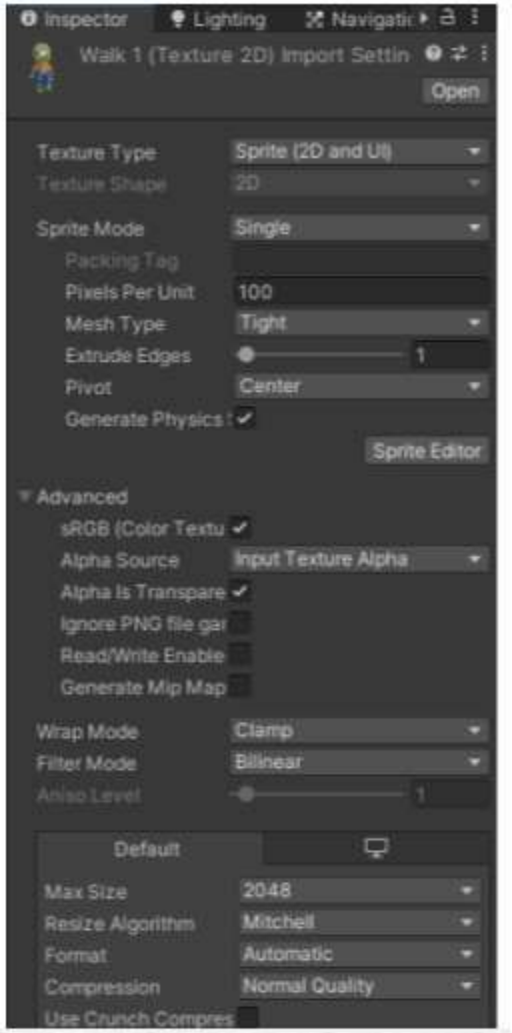


2D Sprites

- Sprites are 2D Graphic objects (images)
  - If the project mode is set to 2D, all imported images are automatically set as a Sprite
- The Sprite Renderer component renders the Sprite and controls how visually appears in a Scene



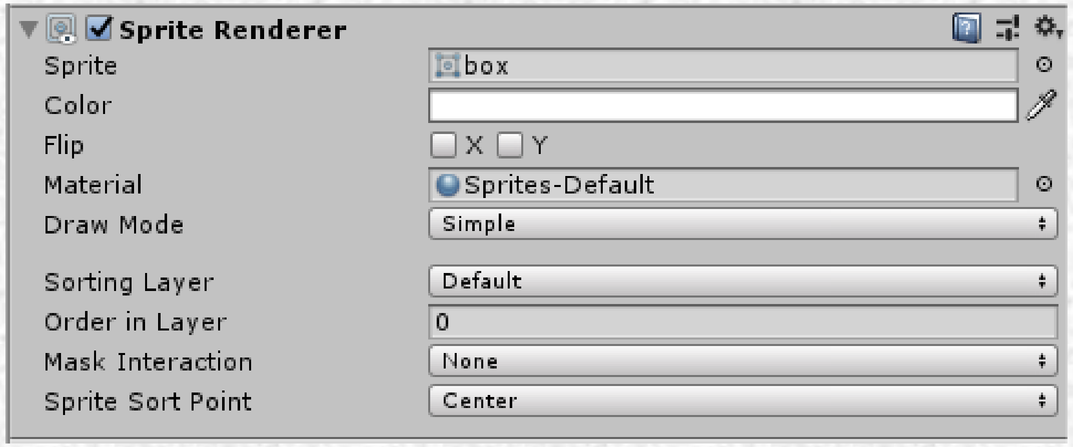
Sprite	Walk1
Color	
Flip	X Y
Draw Mode	Simple
Mask Interaction	None
Sprite Sort Point	Center
Material	Sprites-Default
Additional Settings	
Sorting Layer	Default
Order in Layer	0



Texture Type	Sprite (2D and UI)
Texture Shape	2D
Sprite Mode	Single
Packing Tag	
Pixels Per Unit	100
Mesh Type	Tight
Extrude Edges	1
Pivot	Center
Generate Physics	✓
Advanced	
sRGB (Color Textu	✓
Alpha Source	Input Texture Alpha
Alpha Is Transpare	✓
Ignore PNG file gar	
Read/Write Enable	
Generate Mip Map	
Wrap Mode	Clamp
Filter Mode	Bilinear
Aniso Level	1
Max Size	2048
Resize Algorithm	Mitchell
Format	Automatic
Compression	Normal Quality
Use Crunch Compres	

2D Sprites

- The **Sprite Renderer** component renders the **Sprite** and controls how it visually appears in a **Scene** for both 2D and 3D projects
- When you create a Sprite (**GameObject > 2D Object > Sprite**), Unity automatically creates a GameObject with the **Sprite Renderer** component attached
- You can also add the component to an existing GameObject via the **Components** menu (**Component > Rendering > Sprite Renderer**).



Sprite Renderer

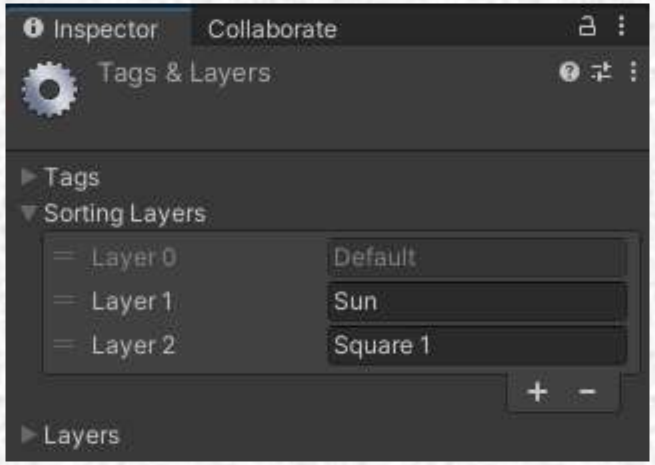
Sprite	box
Color	
Flip	☐ X ☐ Y
Material	Sprites-Default
Draw Mode	Simple
Sorting Layer	Default
Order in Layer	0
Mask Interaction	None
Sprite Sort Point	Center

Sprite Renderer Properties

Property	Function
Sprite	Define which Sprite texture the component should render. Click the small dot to the right to open the object picker window, and select from the list of available Sprite Assets.
Color	Define the vertex color of the Sprite, which tints or recolors the Sprite's image. Use the color picker to set the vertex color of the rendered Sprite texture. See the Color section below this table for examples.
Flip	Flips the Sprite texture along the checked axis. This does not flip the Transform position of the **GameObject**
Material	Define the Material used to render the Sprite texture.
Draw Mode	Define how the Sprite scales when its dimensions change. Select one of the following options from the drop-down box.
Sorting Layer	Set the Sorting Layer of the Sprite, which controls its priority during rendering. Select an existing Sorting Layer from the drop-down box, or create a new Sorting Layer.
Order In Layer	Set the render priority of the Sprite within its Sorting Layer. Lower numbered Sprites are rendered first, with higher numbered Sprites overlapping those below.

2D Sorting Layers

- The Sorting Layer and Order in Layer are available to all 2D Renderers through the Inspector window
- Set the Renderer to an existing Sorting Layer or create a new one to determine its priority in the rendering queue
- Change the value of the Order in Layer to set the Renderer's priority among other Renderers within the same Sorting Layer



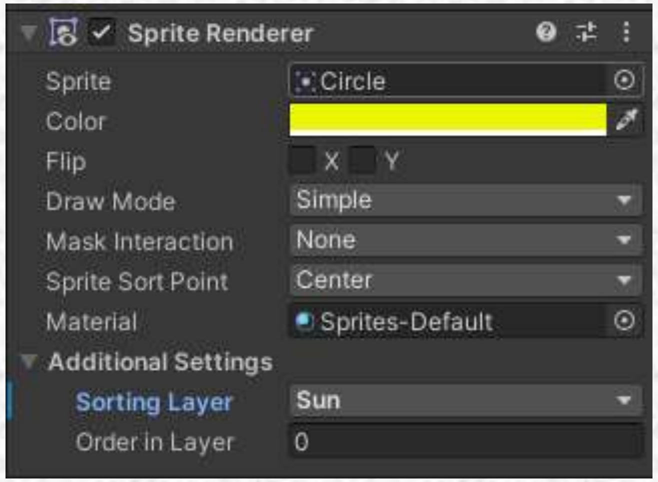
Inspector | Collaborate

Tags & Layers

▸ Tags
▾ Sorting Layers

Layer 0	Default
Layer 1	Sun
Layer 2	Square 1

▸ Layers



Sprite Renderer

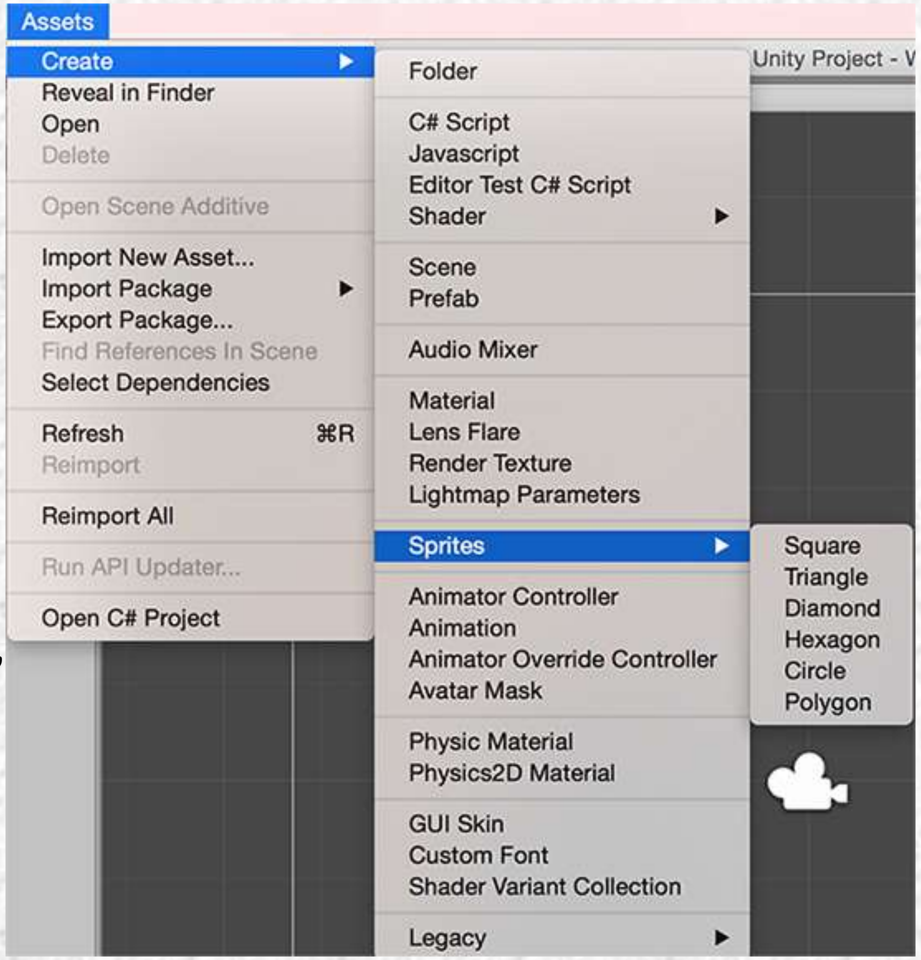
Sprite	Circle
Color	
Flip	X Y
Draw Mode	Simple
Mask Interaction	None
Sprite Sort Point	Center
Material	Sprites-Default

▾ Additional Settings

Sorting Layer	Sun
Order in Layer	0

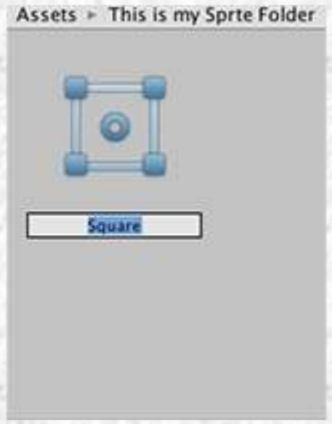
Sprite Creator

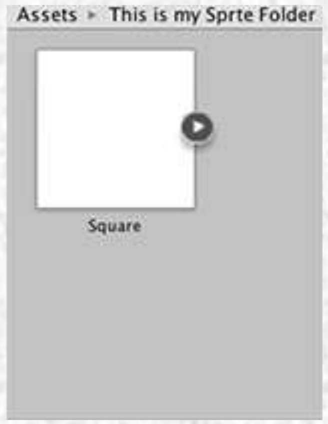
- With this tool you can create temporary placeholder **sprite** (2D) graphics. You can use these in your project during development and then replace them with the graphics you want to use.
- Select **Assets>Create>Sprites** and then select the placeholder sprite you want to make (square, triangle, diamond, hexagon, or polygon).



Using the Sprite

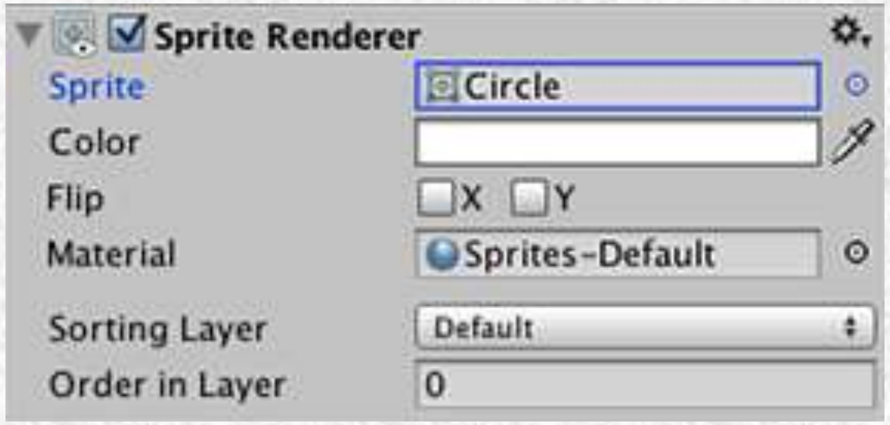
- Your new placeholder sprite appears as a white shape in the asset folder you currently have open.
- The new sprite's name defaults to its shape name but you have the option to rename your sprite when it is first created
- If you are not sure what you want to call it, leave it as the default; you can change it later by clicking on it





Replacing your Placeholder Sprite

- To change your placeholder sprite, click on it in the **Scene View** and then edit via the **Sprite Renderer Component** in the **Inspector**
- A Unity window that displays information about the currently selected GameObject, asset or project settings, allowing you to inspect and edit the values.
- Edit the **Sprite** field: You can click on the small circle to the right of the input field to bring up the **Sprite Selector** where you can browse and select from a menu of available 2D graphic assets.



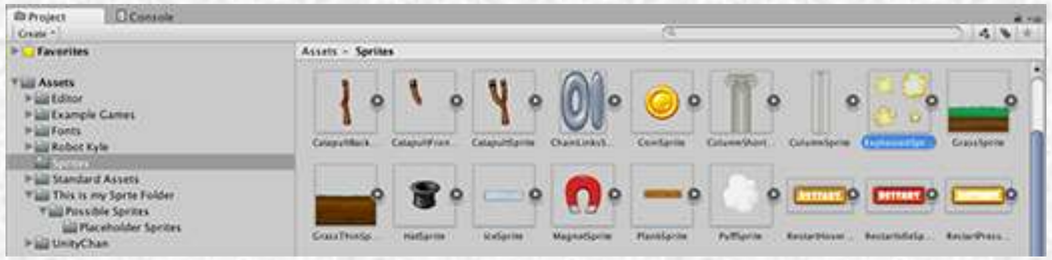
Sprite	Circle
Color	
Flip	☐ X ☐ Y
Material	Sprites-Default
Sorting Layer	Default
Order in Layer	0

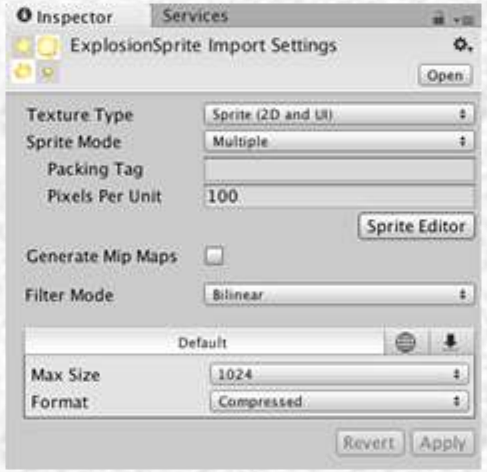
Sprite Editor

- Sometimes a Sprite Texture contains just a single graphic element but it is often more convenient to combine several related graphics together into a single image
  - For example, the image could contain component parts of a single character, as with a car whose wheels move independently of the body.
- Unity makes it easy to extract elements from a composite image by providing a Sprite Editor for the purpose

Opening the Sprite Editor

- To open the Sprite Editor:
    - Select the 2D image you want to edit from the Project View
    - Click on the **Sprite Editor** button in the Inspector
- You can only see the Sprite Editor button if the Texture Type on the image you have selected is set to Sprite (2D and UI).




Inspector | Services

ExplosionSprite Import Settings

Open

Texture Type	Sprite (2D and UI)
Sprite Mode	Multiple
Packing Tag	
Pixels Per Unit	100

Sprite Editor

Generate Mip Maps	☐
Filter Mode	Bilinear

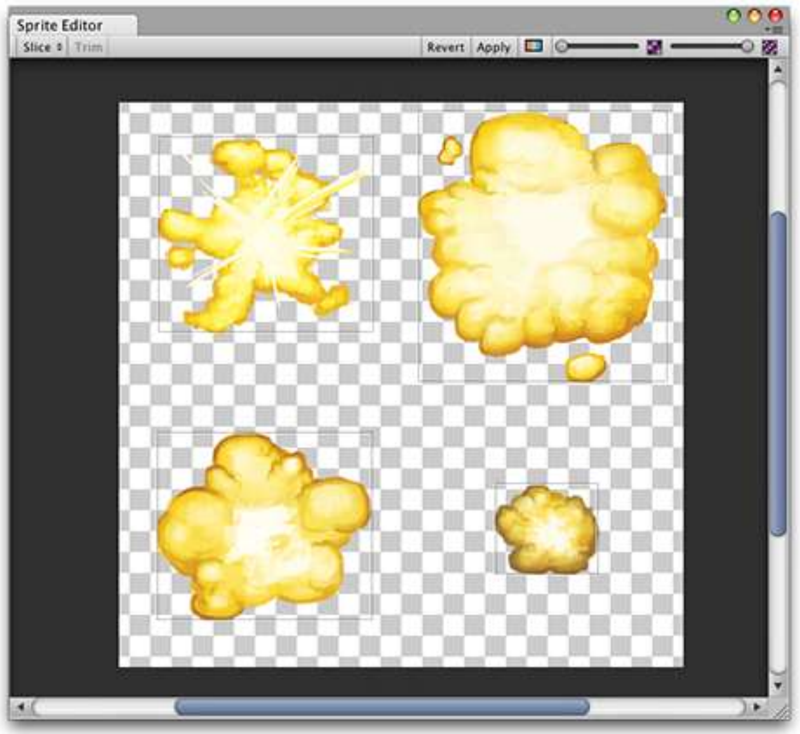
Default

Max Size	1024
Format	Compressed

Revert | Apply

Opening the Sprite Editor

- Set the Sprite Mode to Multiple in the Texture Import Inspector if your image has several elements.

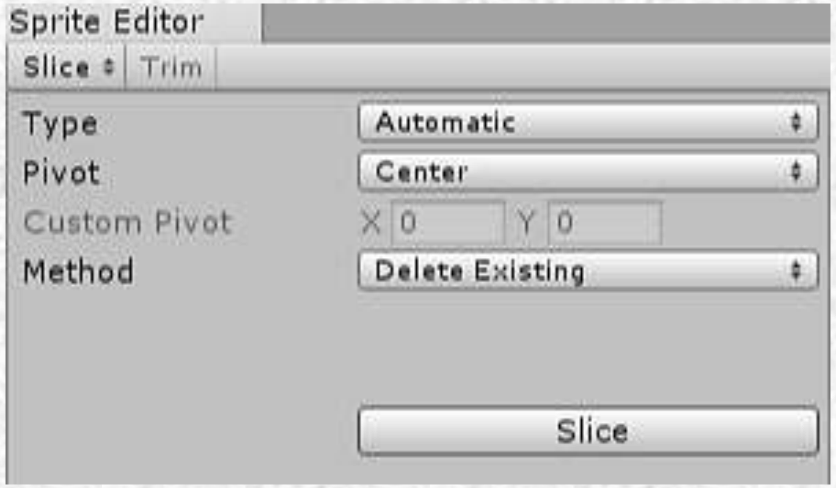


Opening the Sprite Editor

- The most important control in the sprite editor is the Slice menu at the top left, which gives you options for separating the elements of the image automatically.
- Apply and Revert buttons allow you to keep or discard any changes you have made.

Automatic Slicing

- Isolating the Sprite rectangles manually works well but in many cases, Unity can save you work by detecting the graphic elements and extracting them for you automatically
- If you click on the **Slice** menu in the control bar, you will see this panel



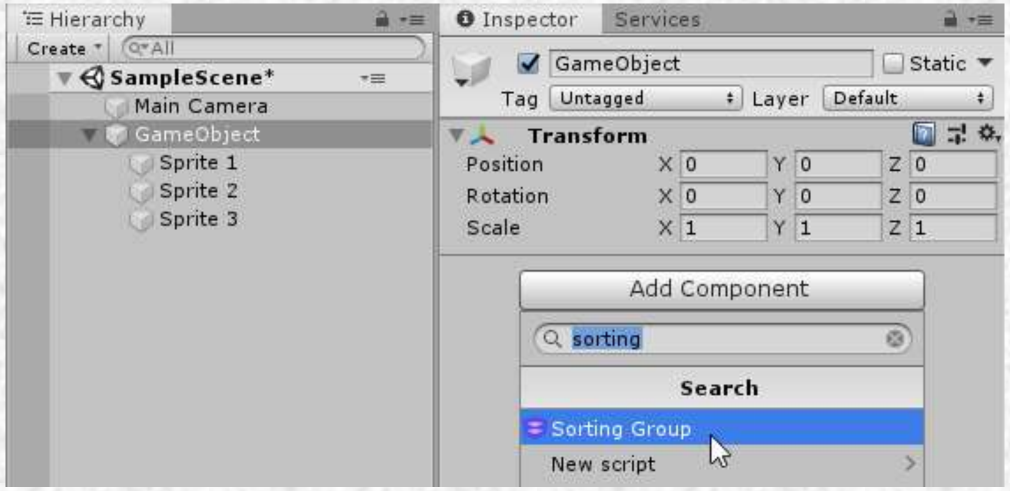
Type: Automatic
Pivot: Center
Custom Pivot: X 0 Y 0
Method: Delete Existing

Automatic Slicing

- With the slicing type set to **Automatic**, the editor will attempt to guess the boundaries of Sprite elements by transparency. You can set a default pivot for each identified Sprite
- The **Method** menu lets you choose how to deal with existing selections in the window.
- The **Delete existing** option will simply replace whatever is already selected, **Smart** will attempt to create new rectangles while retaining or adjusting existing ones, and **Safe** will add new rectangles without changing anything already in place.

Sorting Groups

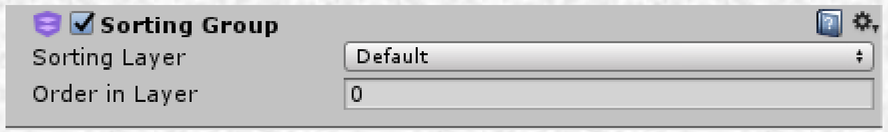
- Sorting Groups allow you to group GameObjects with Sprite Renderers together, and control the order in which they render their Sprites
- Unity renders Sprite Renderers in the same Sorting Group together, as if they are a single GameObject.
- To place a GameObject into a Sorting Group, add the Sorting Group component to it. To do this, select the GameObject and go to **Component > Rendering > Sorting Group**



GameObject
Tag: Untagged
Layer: Default
Position X 0 Y 0 Z 0
Rotation X 0 Y 0 Z 0
Scale X 1 Y 1 Z 1
sorting

Sorting Group properties

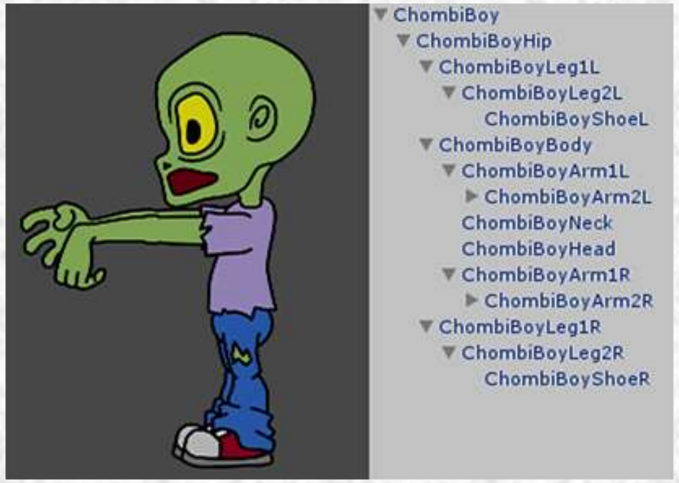
- Unity uses a **Sorting Group’s Sorting Layer and** Order in Layer values to determine its priority in the rendering queue among other Sorting Groups and GameObjects in the Scene


Sorting Group
Sorting Layer: Default
Order in Layer: 0

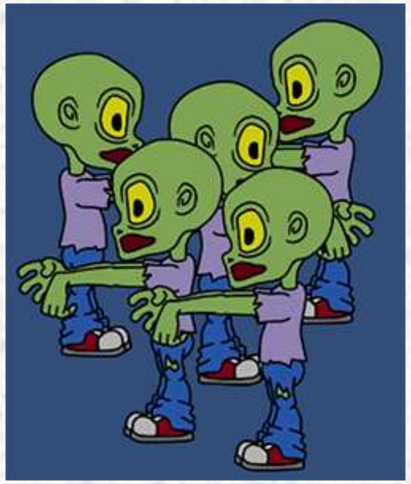
Property	Function
Sorting Layer	Select or add a Sorting Layer from this drop-down menu to determine the Sorting Group’s position in the render queue.
Order in Layer	Set the render order of this Sorting Group within its **Sorting Layer**. Unity queues Renderers with lower values first in the render queue, so they appear before Renderers with higher values.

Using Sorting Groups

- The most common way to create a 2D multi-Sprite character is to arrange and parent multiple Sprite Renderers together in the Hierarchy window to form a character
- You can use **Sorting Groups** to help manage this kind of complex multi-Sprite character.

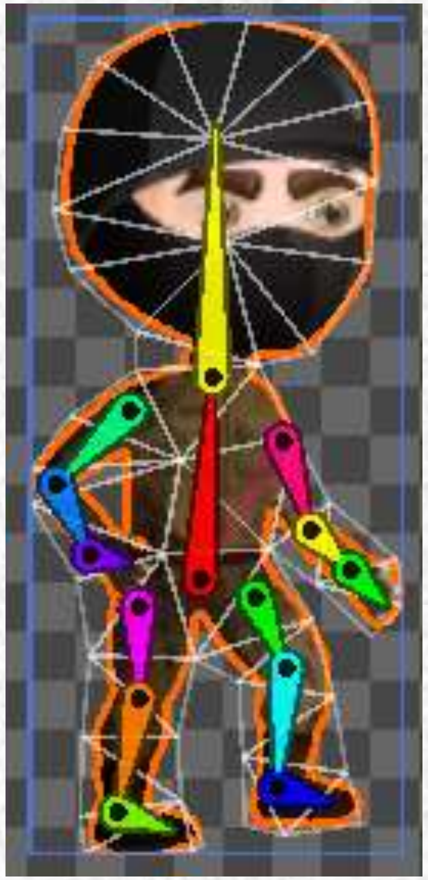


```
▼ ChombiBoy
  ▼ ChombiBoyHip
    ▼ ChombiBoyLeg1L
      ▼ ChombiBoyLeg2L
          ChombiBoyShoeL
    ▼ ChombiBoyBody
      ▼ ChombiBoyArm1L
        ► ChombiBoyArm2L
        ChombiBoyNeck
        ChombiBoyHead
      ▼ ChombiBoyArm1R
        ► ChombiBoyArm2R
    ▼ ChombiBoyLeg1R
      ▼ ChombiBoyLeg2R
          ChombiBoyShoeR
```



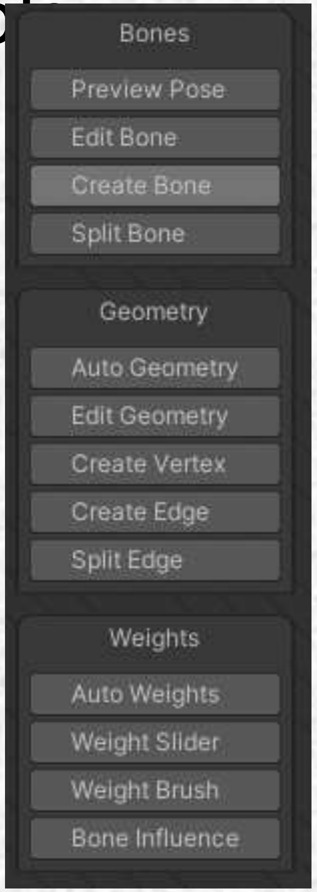
Rig a 2D Character using Skinning Editor

- In order to start rigging a 2D character Sprite Asset, your character will already be prepared and saved into the Assets folder of your project
- The Skinning Editor is used to create a skeleton in the character rigging process.
- The skeleton, or ‘bones’, (also known as a rig,) will be bound to the Sprite itself
- The Sprite’s graphics act as a sort of ‘skin,’ giving the Skinning Editor its name
- The Skinning Editor is a subset of the Sprite Editor

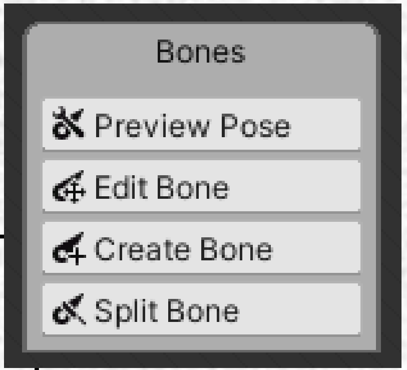


Understanding the Skinning Editor

- The Skinning Editor consists of three groups of contro
  - Bones
  - Geometry
  - Weights

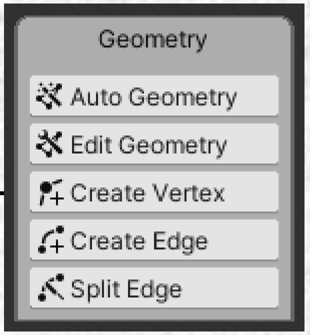


Bones



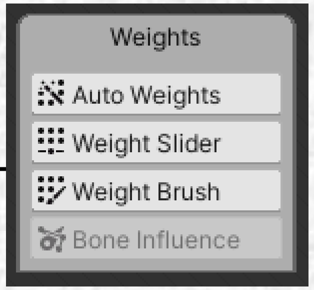
- Preview Pose
  - This allows you to move and rotate bones to see how the rigging affects posing and animation. You can connect, disconnect, and reposition bones, but these changes are not permanent and are only active during the preview.
- Edit Joints
  - Move, reposition, disconnect, and reconnect bones.
  - To move a bone without disconnecting its joints, click-and-drag either the root or the tip of the bone
  - To disconnect a joint, click-and-drag the bone until it separates
- Create Bone
  - This allows you to create a bone.
  - If you are switching to this tool from another, the bone will by default be a child of the root of the previously selected bone chain, starting a new branch in the Hierarchy
- Split Bone
  - This splits a single bone into two connected bones.

Geometry



- Auto Geometry
  - This automatically creates a vertex Mesh for the Sprite, assigns the existing bones to vertices, and generates bone weighting information.
  - Bone weights are set per vertex, and determine how much influence a bone has over a vertex.
- Edit Geometry
  - This allows you to move and delete edges and vertices.
  - If you delete a border edge, causing an open Mesh, Unity only retains the vertices and other border edges.
  - Interior edge information is lost until the surface is closed again or until Auto Geometry is used.
- Create Vertex
  - This allows you to create a vertex, automatically creating edges to connect it to neighboring vertices.
- Create Edge
  - This allows you to create an edge between two existing vertices, or one existing vertex and a new one placed where you click.
- Split Edge
  - This splits the edge nearest your mouse by creating a new vertex where you click, connecting it with both endpoints of the original edge.

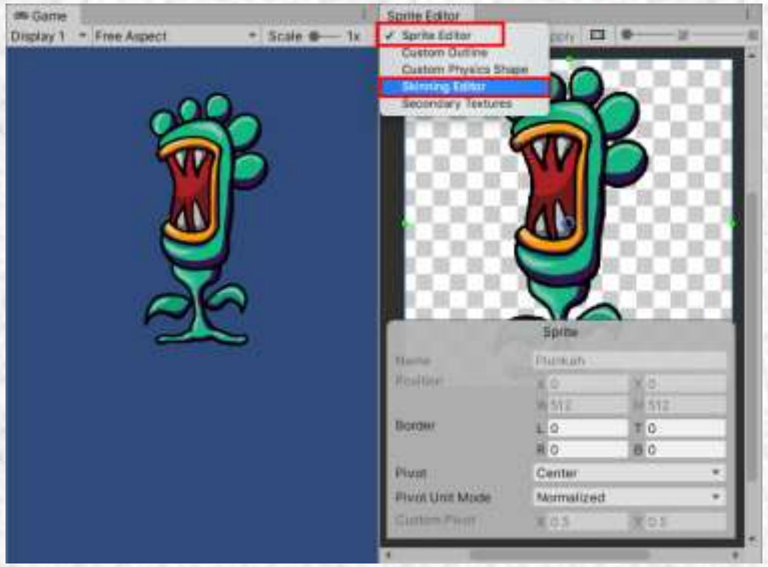
Weights



- Auto Weights
  - This will open a panel that can automatically generate, normalize, or clear bone weights
- Weight Slider
  - Selected vertices can be adjusted via a set of sliders. Select vertices one at a time by clicking on them.
- Weight Brush
  - This allows you to select a bone and paint its area of influence using a traditional digital paintbrush

Using the Skinning Editor

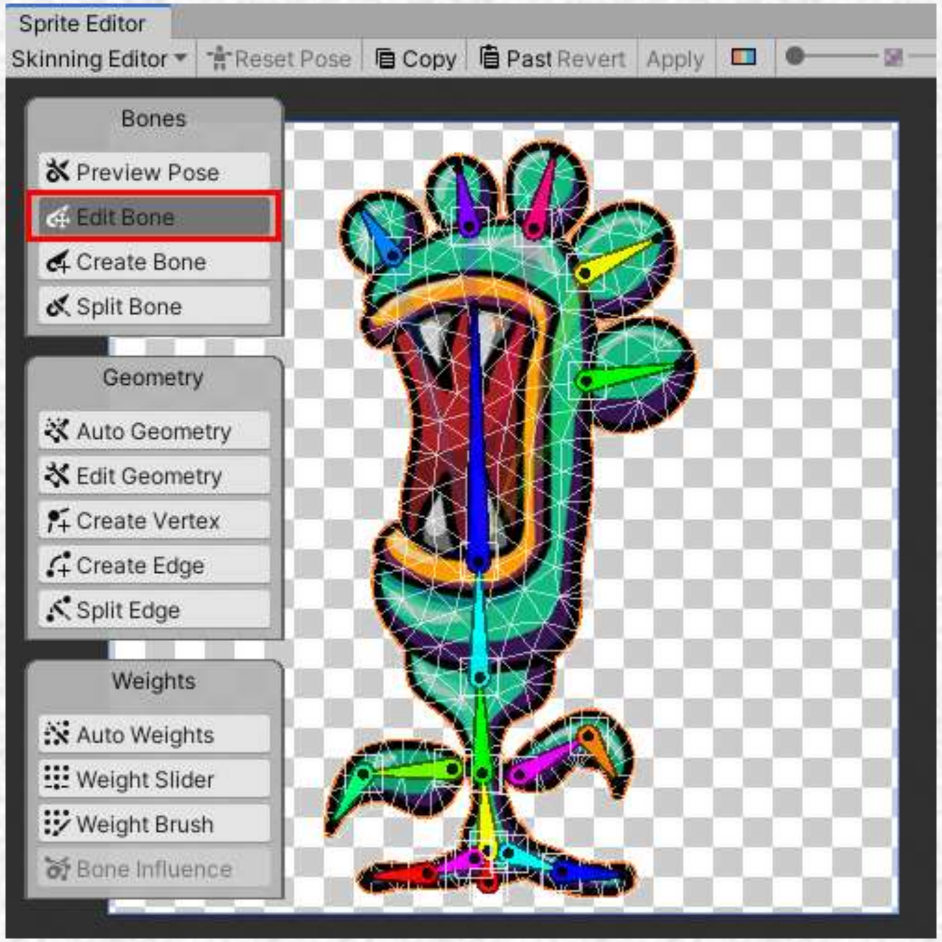
- To start rigging a 2D Sprite, select the Sprite added to your project in the Project Window, and from the dropdown menu select
  - Window > 2D > Sprite Editor
  - Select the dropdown triangle to enter the Skinning Editor



Name Plunkah
Position X 0 Y 0
W 512 H 512
Border L 0 T 0
R 0 B 0
Pivot Center
Pivot Unit Mode Normalized
Custom Pivot X 0.5 Y 0.5

Creating Bone Skeleton Cont.

- Select the Edit Bone Button, and double-click on the Character Sprite to make the bones visible

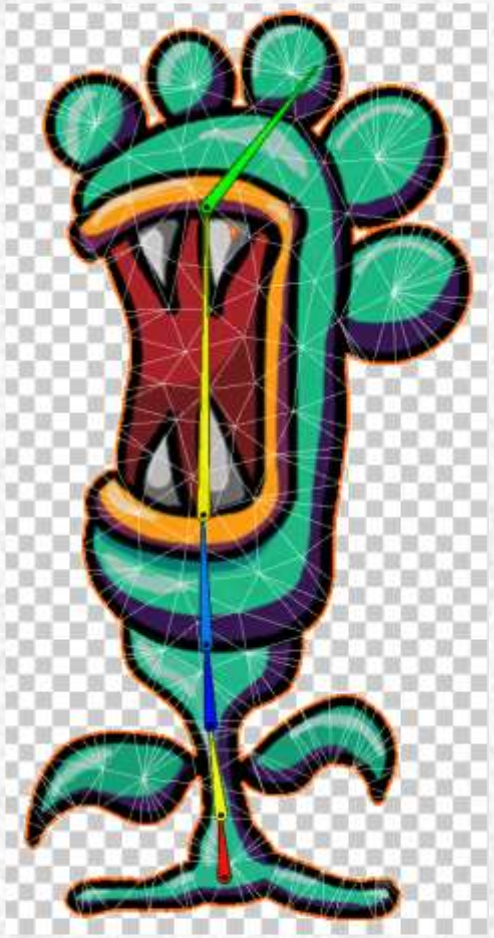


Creating Bone Skeleton Cont.

- Select the Create Bone button
- Click and drag to create new bones, clicking along the character’s skin, pressing the ESC key when finished
- If clicking doesn’t place the root of a bone, double-click your Sprite to enable editing mode

Creating Bone Skeleton Cont.

- Place the root of the first bone in the bottom center of your character’s body
- This bone will serve as the Sprite’s handle
- Continue up the central column, with the last bone roughly in the center leaf on the character’s head

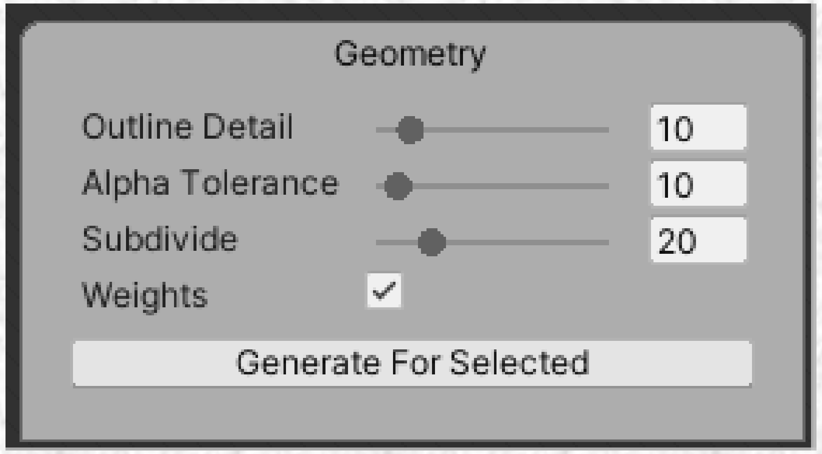


Creating Bone Skeleton Cont.

- Right-click or press the ESC key to break the chain, and click in another section of your character’s pose, such as an arm, leg, leaf or tendril, and beginning near the central column, add a couple of more bones to add to the skeleton.

Creating the Bone Geometry

- Select the Auto Geometry button
- In the popup Geometry panel, select Generate for Selected
- Select Apply at the top of the Sprite / Skinning Editor window to apply these changes



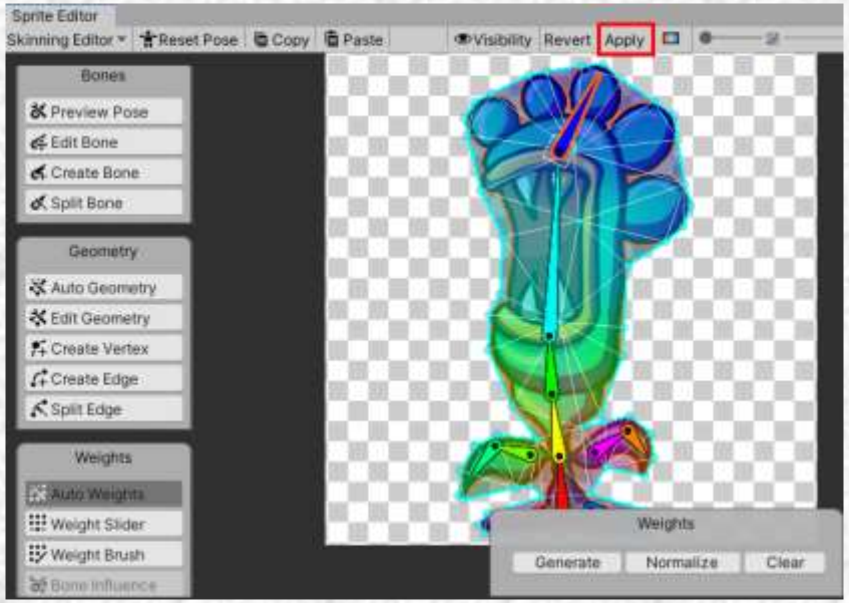
Geometry

Outline Detail 10

Alpha Tolerance 10

Subdivide 20

Weights ✓

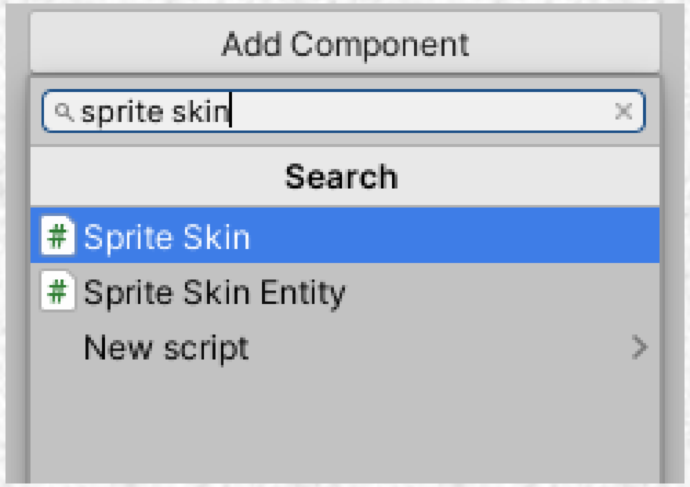


Using Weights

- Try using the Weight Slider controls to adjust vertex weights.
- Remember that the more a vertex is influenced by a bone, the more it will take on that bone’s color in the weight map.
- Any changes made since the last enabling of Apply can be undone by selecting Revert

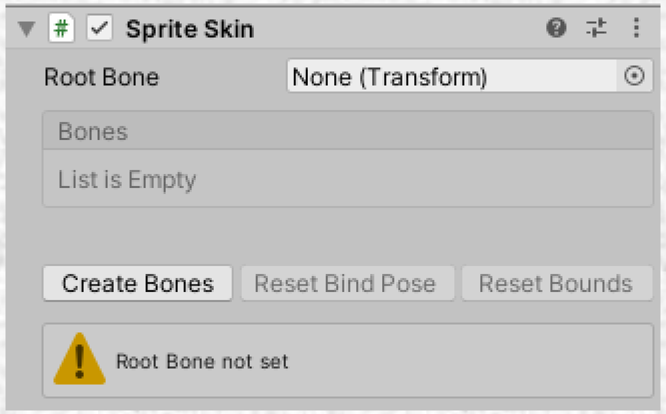
Using Rig in the Game Scene

- Drag your rigged Sprite into the Hierarchy view to create a GameObject.
- Add a Sprite Skin component to your sprite, by first selecting your Sprite in the Hierarchy Window, and then selecting the Add Component Button, and then searching for Sprite Skin



Using Rig in the Game Scene Cont.

- The Sprite Skin component will detect the joints as configured in the Skinning Editor, but there are no corresponding bones



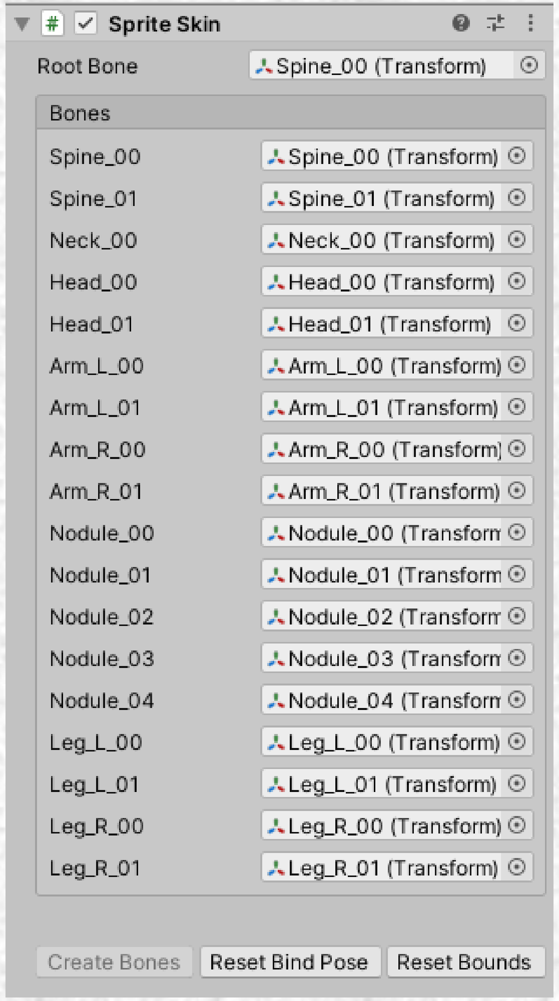
Sprite Skin

Root Bone: None (Transform)

Bones

List is Empty

Root Bone not set



Sprite Skin

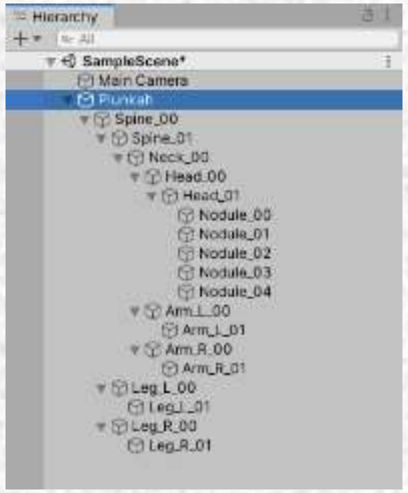
Root Bone: Spine_00 (Transform)

Bones

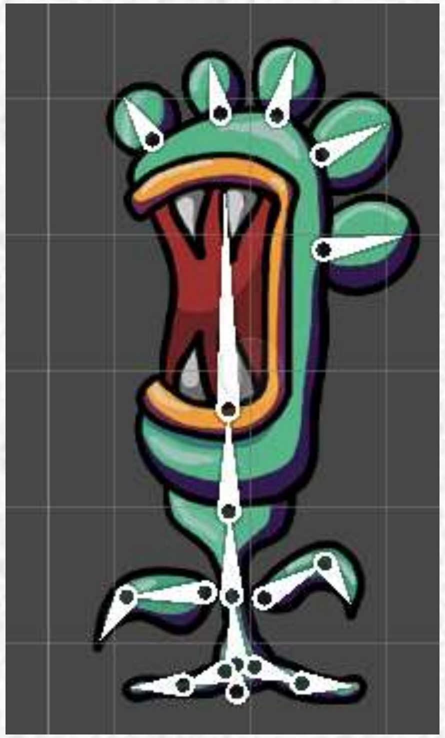
Spine_00	Spine_00 (Transform)
Spine_01	Spine_01 (Transform)
Neck_00	Neck_00 (Transform)
Head_00	Head_00 (Transform)
Head_01	Head_01 (Transform)
Arm_L_00	Arm_L_00 (Transform)
Arm_L_01	Arm_L_01 (Transform)
Arm_R_00	Arm_R_00 (Transform)
Arm_R_01	Arm_R_01 (Transform)
Nodule_00	Nodule_00 (Transform
Nodule_01	Nodule_01 (Transform
Nodule_02	Nodule_02 (Transform
Nodule_03	Nodule_03 (Transform
Nodule_04	Nodule_04 (Transform
Leg_L_00	Leg_L_00 (Transform)
Leg_L_01	Leg_L_01 (Transform)
Leg_R_00	Leg_R_00 (Transform)
Leg_R_01	Leg_R_01 (Transform)

Using Rig in the Game Scene Cont.

- Select the Create Bones button to populate the Root and Bones of the Sprite Skin



```
SampleScene*
  Main Camera
  Plunkah
    Spine_00
      Spine_01
        Neck_00
          Head_00
            Head_01
              Nodule_00
              Nodule_01
              Nodule_02
              Nodule_03
              Nodule_04
          Arm_L_00
            Arm_L_01
          Arm_R_00
            Arm_R_01
      Leg_L_00
        Leg_L_01
      Leg_R_00
        Leg_R_01
```



References

- https://learn.unity.com/tutorial/rigging-a-sprite-with-the-2d-animation-package?uv=2019.4#
- https://docs.unity3d.com/Packages/com.unity.2d.animation@9.0/manual/PreparingArtwork.html
- https://www.youtube.com/watch?v=EZtpACxCTEE&t=73s

Computer Game Programming (SE3173)

Ibrar Arshad
Ibrar.arshad@cust.edu.pk

Animations using Animator (Animation Controller)

- Create an animations (states)
- Design the sequence of animation (states) and transition between them using Animator
- Add parameters (triggers/variables)
- Assign triggers to transitions from one state to another
- Add script that assigns keys to specific triggers

Script

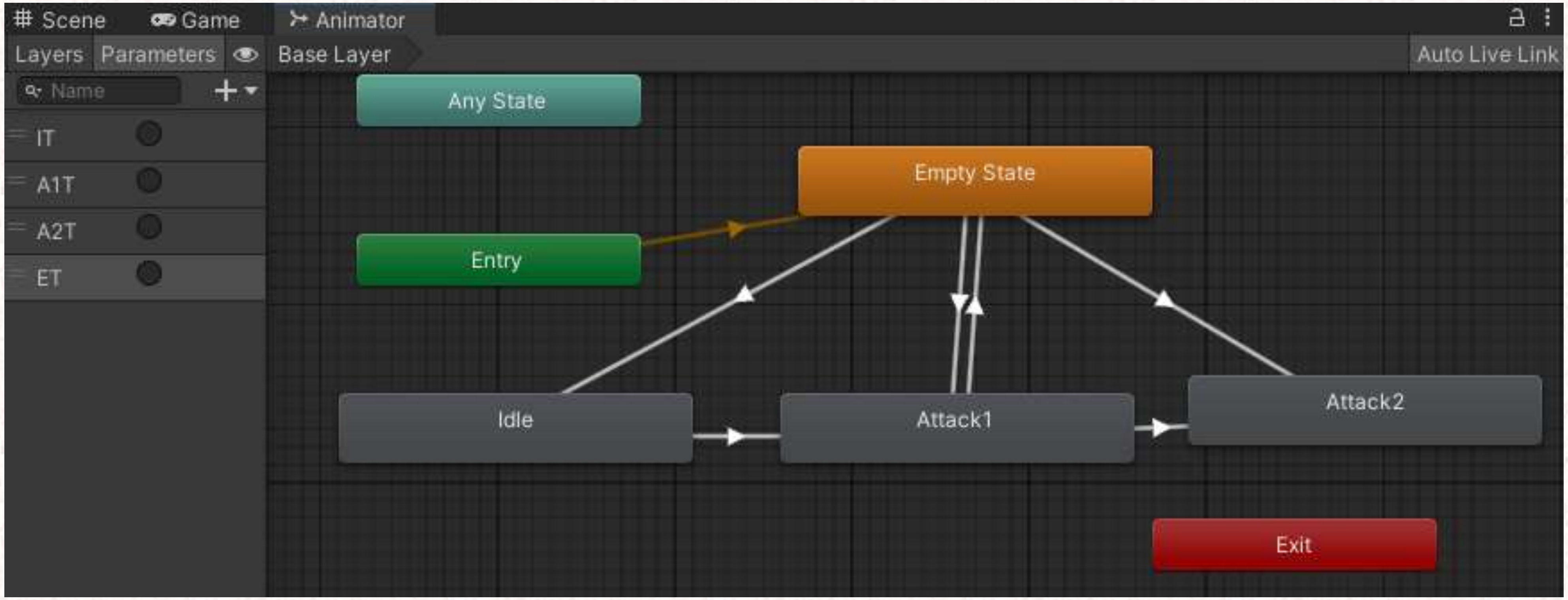
- Get the Animation Controller object

```
anim_Controller = GetComponent<Animator>();
//where anim_Controller is object of type Animator
```

- Get KeyCode and assign it to specific trigger

```
    if(Input.GetKeyDown("a"))
         anim_Controller.SetTrigger("IdleTrigger");
//where IdleTrigger is a trigger assigned to transition
between a state to Idle State
```

Example


Scene
Game
Animator
Layers
Parameters
Base Layer
Auto Live Link
Name
IT
A1T
A2T
ET
Any State
Empty State
Entry
Idle
Attack1
Attack2
Exit

Script for controlling animation transitions

```
public class TController : MonoBehaviour
{
    private Animator anim_Control;
    // Start is called before the first frame update
    void Start()
    {
       anim_Control = GetComponent<Animator> ();
    }

    // Update is called once per frame
    void Update(){
        if(Input.GetKeyDown("a"))
            anim_Control.SetTrigger("IT");

        if(Input.GetKeyDown("s"))
            anim_Control.SetTrigger("A1T");

        if(Input.GetKeyDown("d"))
            anim_Control.SetTrigger("A2T");

        if(Input.GetKeyDown("w"))
            anim_Control.SetTrigger("ET");
    }
}
```