MỤC LỤC

DAY 1: INTRODUCTION AND SETUP

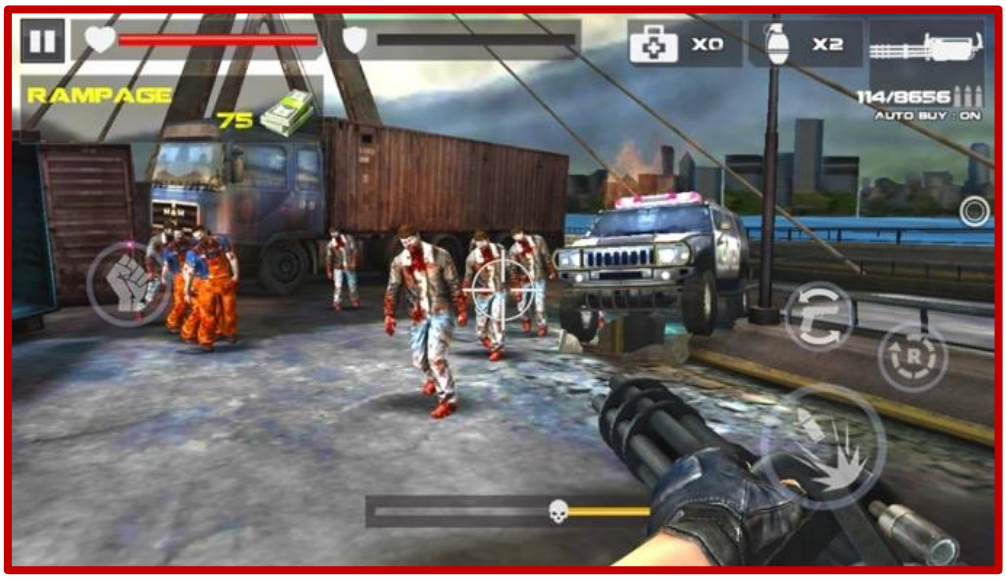
Unity Introduction

Unity Installation

WHAT IS UNITY?

Visual tools

For 3D / 2D game creation

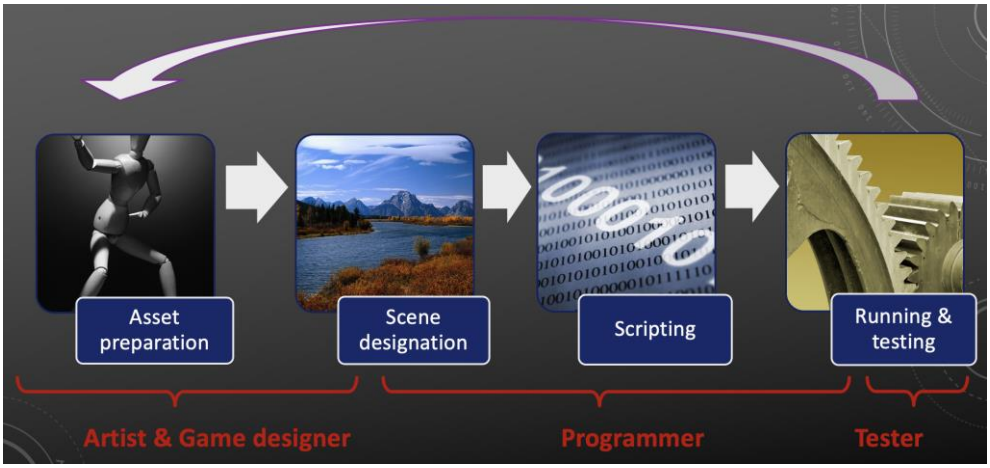


On multiplatform

Unity Advantages

- Create games belong to almost genres
- Support most of popular platforms
- Simple, visual and easy to learn
- Large community
- Free with full features
- Regularly upgrade

Game Development Process



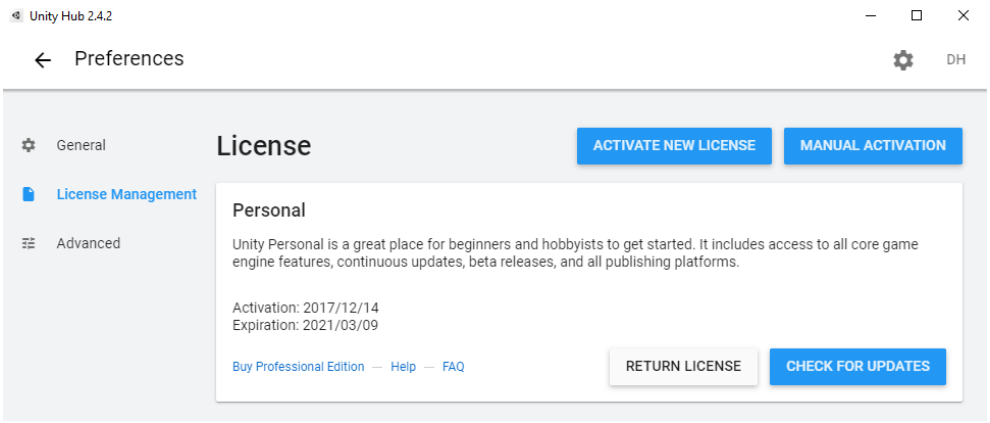
Full Information: Official website at http://unity.com

- Download
- References
- Learning
- Reporting
- Forum

Unity Installation

- Via Unity Hub https://unity3d.com/get-unity/update

- All versions http://unity3d.com/get-unity/download/archive
- REQUEST A LICENSE



Learning Materials

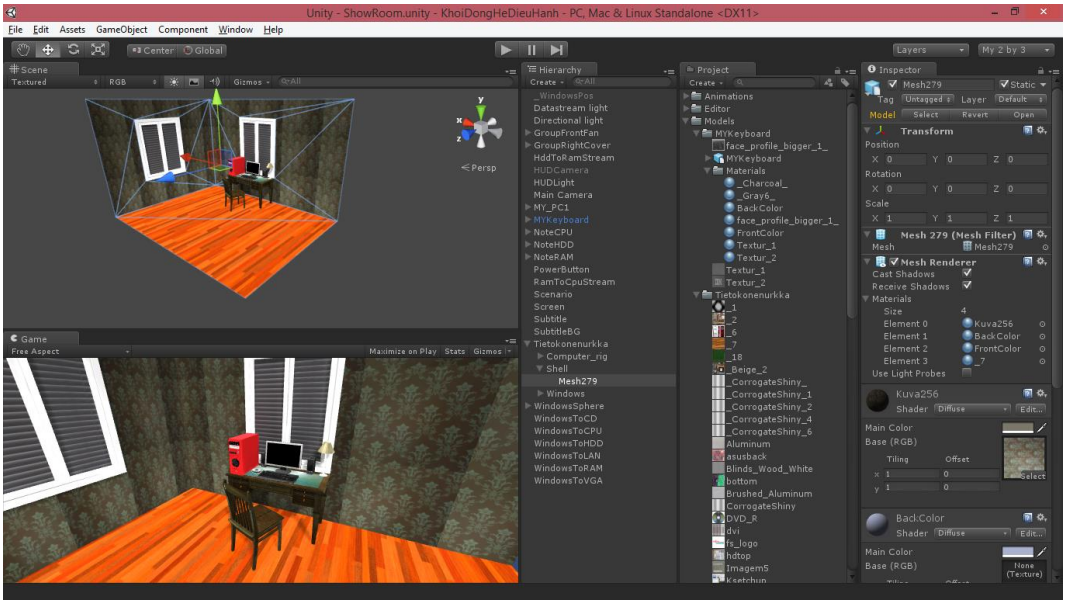
- Unity Learn: https://learn.unity.com/
- Udemy Videos: https://www.udemy.com/courses/search/?src=ukw&q=Unity
- Packt Pub Videos: https://www.packtpub.com/catalogsearch/result/?q=Unity&product_type_filter=Video&released=Available

DAY 2: WORKING ENVIRONMENT

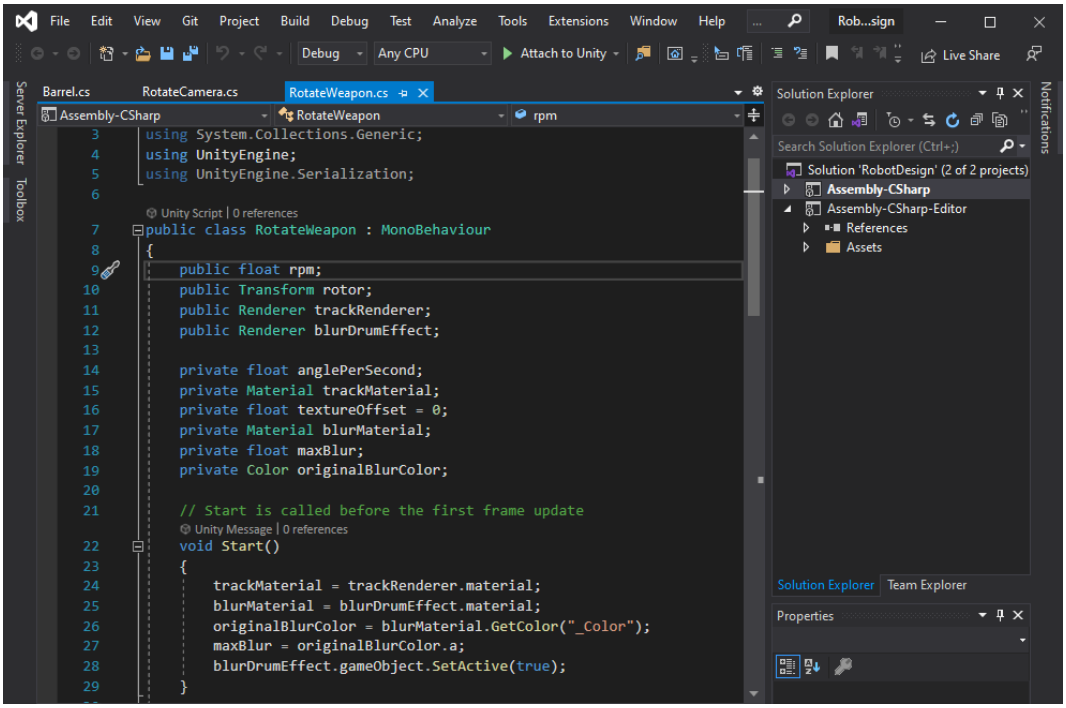
Main Editor

Project Structure

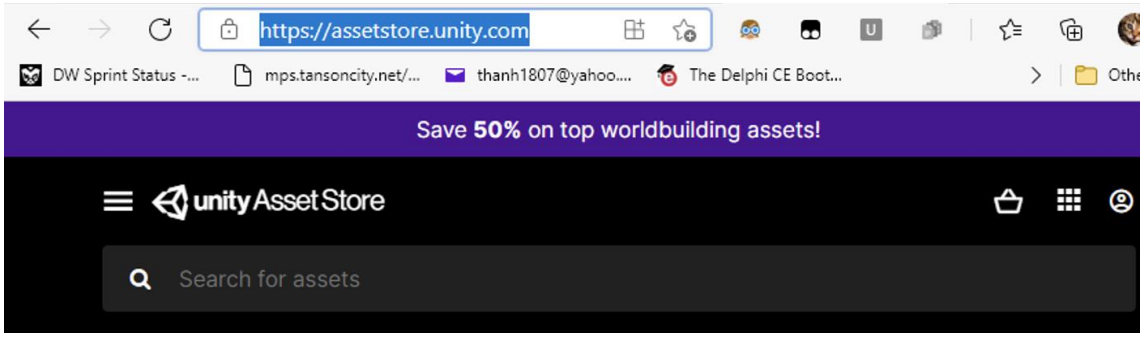
MAIN EDITOR



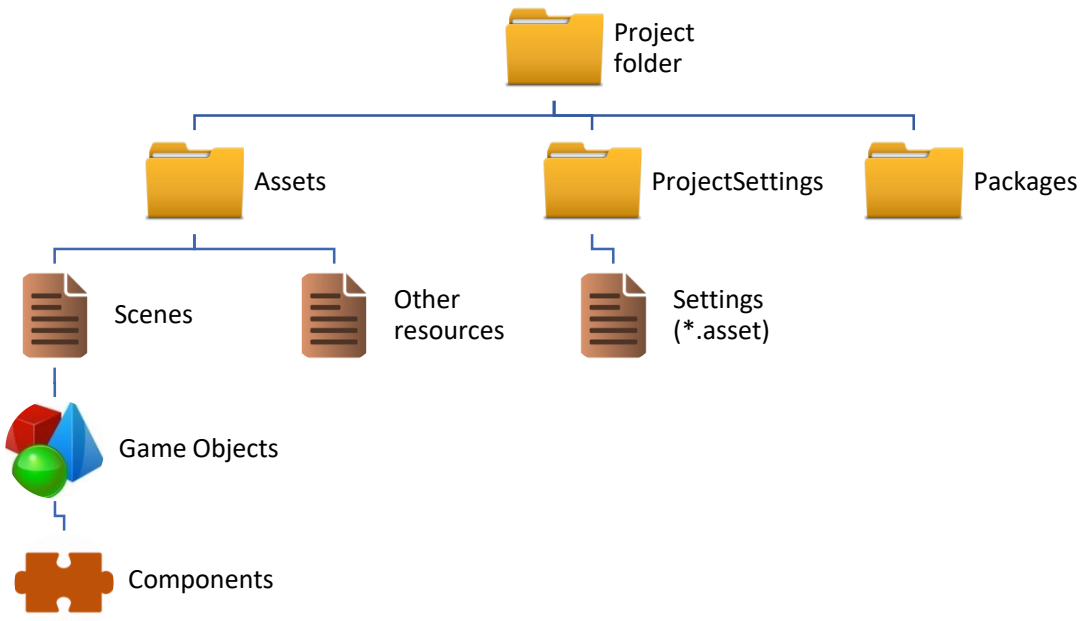
VISUAL STUDIO FOR SCRIPTING



ASSET STORE



PROJECT STRUCTURE

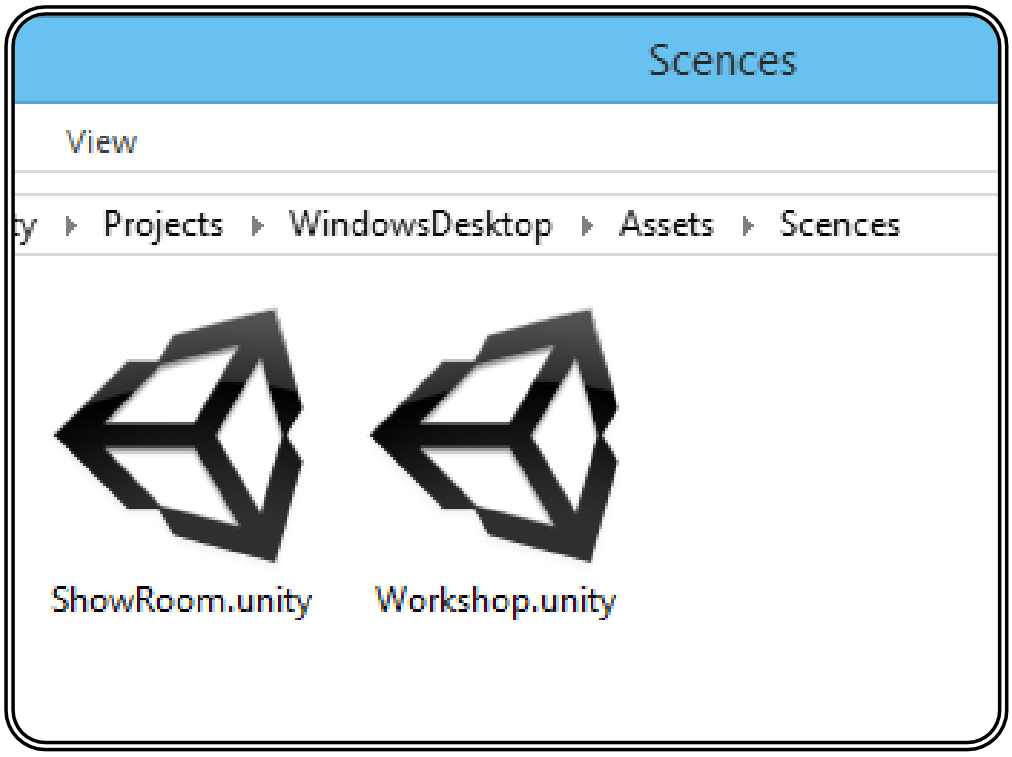


PROJECT ASSETS

- Each asset is a file.
- Can be used for all scene.
- General (imported): model, texture, sound…
- Unity specific (generated): material, shader, script, animation, prefabs…
- Asset configurations are contained in meta files (*.meta)

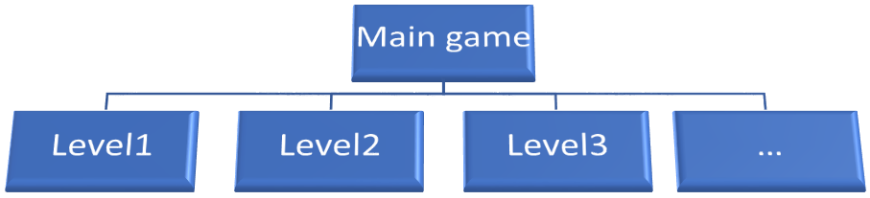
WHAT IS SCENE?

- A Unity file (*.unity) served as main workspace
- A 3D world that contains game objects



WHAT A SCENE CAN BE?

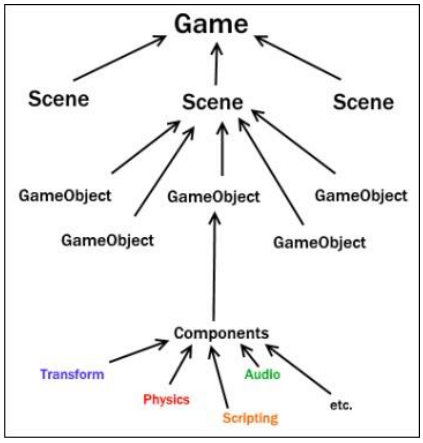
- A state (Main Menu, Action Phase…)
- A map / environment
- A mini game
- A level

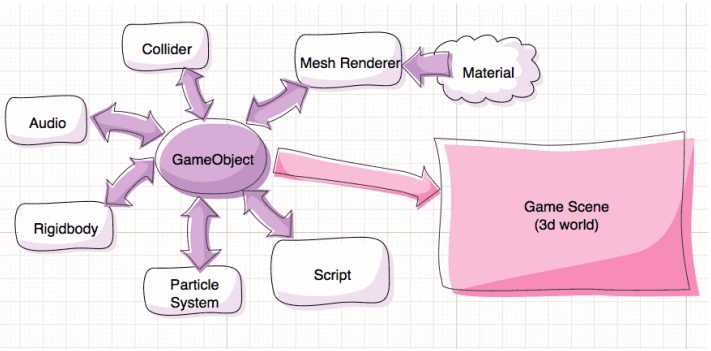


WHAT IS GAME OBJECT?

- Game objects are single instances defined in scenes.
- Game object can be a camera, light source, terrain, models, main character, NPC…
- Game objects are built up from assets.
- Game objects are stored in memory at runtime only.

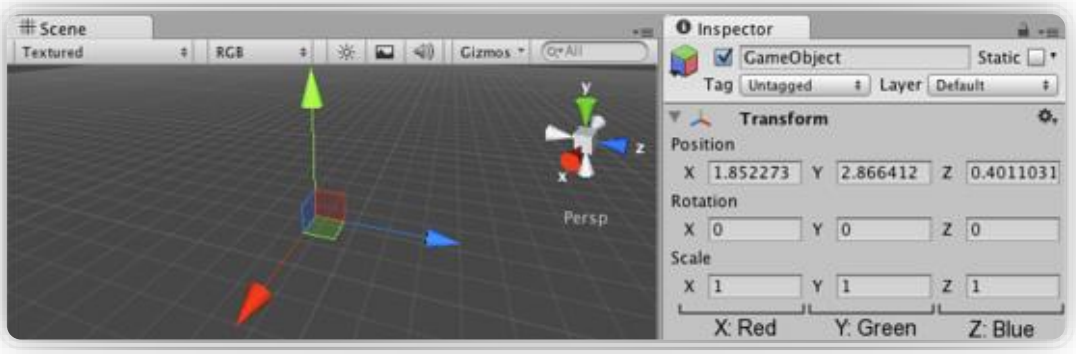
COMPONENTS



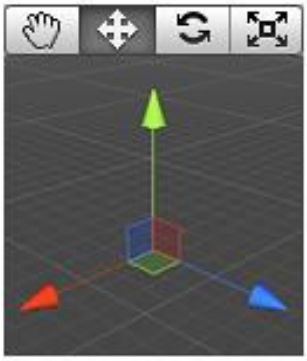


TRANSFORM COMPONENT

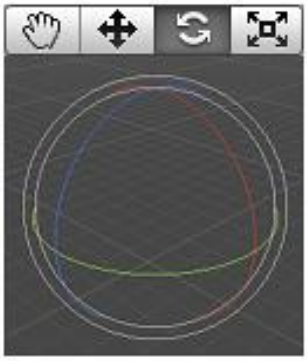
- Position: in X, Y and Z coordinates
- Rotation: around X, Y, and Z axes, measured in degrees
- Scale: along X, Y, and Z axes Value “1” the original size.



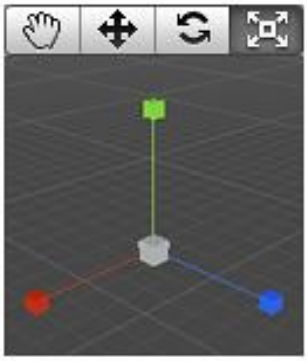
OBJECT TRANSFORMATION



Translate (W)



Rotate (E)



Scale (R)

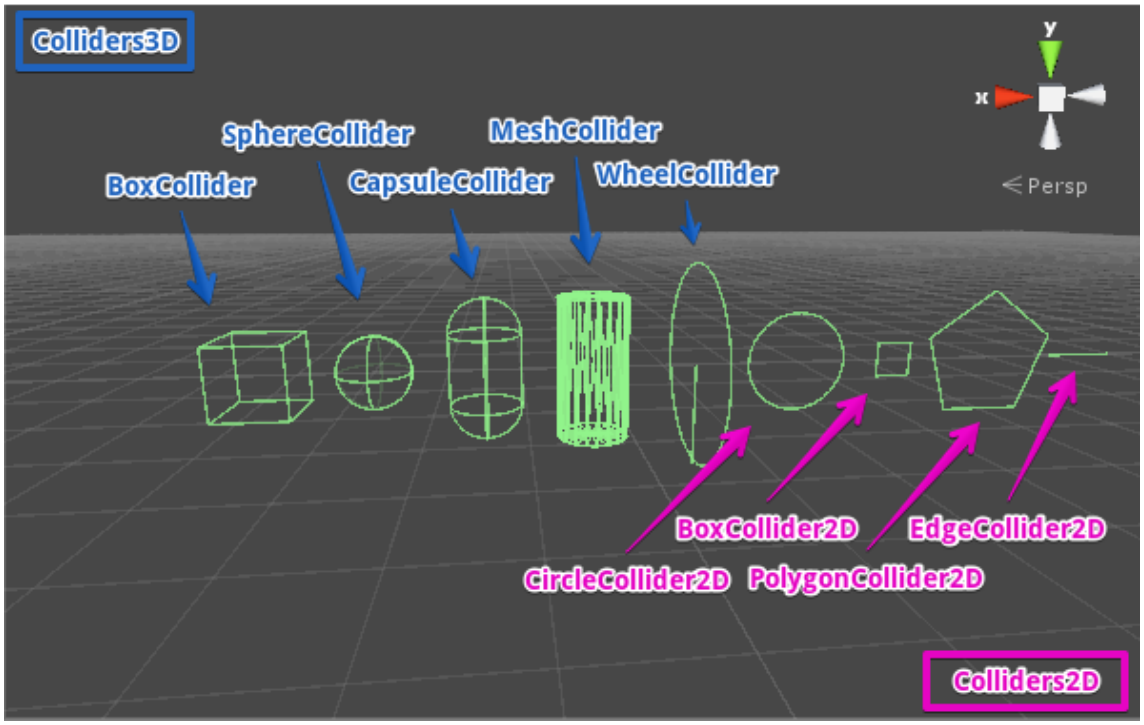
DAY 3: PHYSICS AND ANIMATIONS

Collider2D

Rigidbody2D

Effector2D

UNITY COLLIDER SHAPES



COLLIDER2D TYPES:

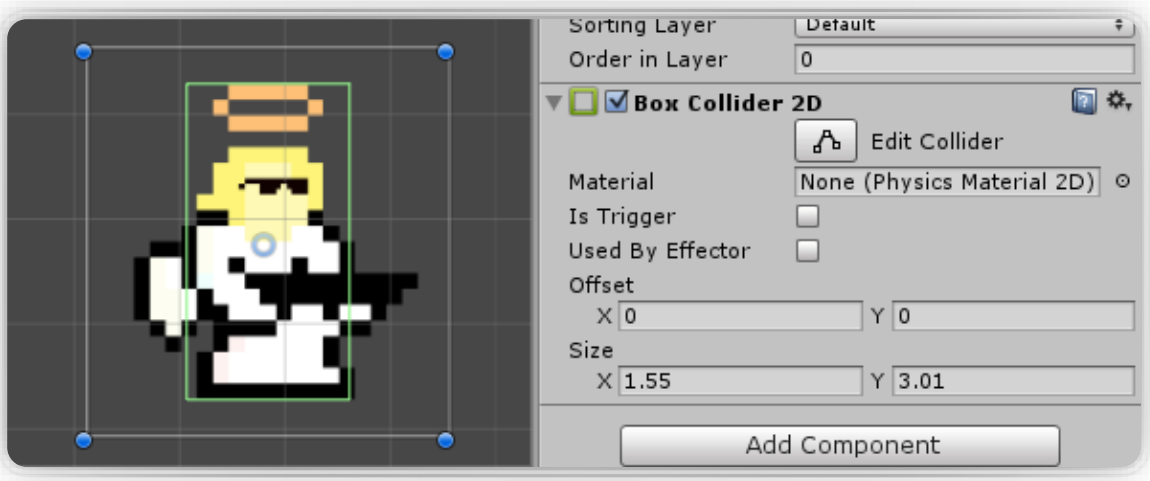
The Collider 2D types that can be used with Rigidbody2D are

- Circle Collider 2D for circular collision areas.
- Box Collider 2D for square and rectangle collision areas.
- Polygon Collider 2D for freeform collision areas.
- Edge Collider 2D for freeform collision areas and areas which aren't completely enclosed (such as rounded convex corners).
- Capsule Collider 2D for circular or lozenge-shaped collision areas.
- Composite Collider 2D for merging Box Collider2Ds and Polygon Collider 2D

BOX COLLIDER 2D

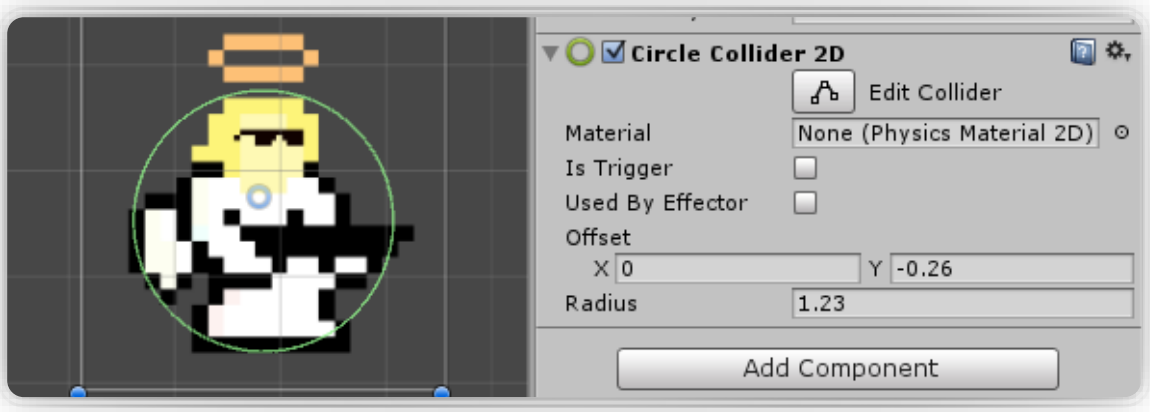
- BoxCollider2D define a 2D Rectangle.

- They are commonly used in most of 2D game objects: ground, obstacles, characters bodies, etc..



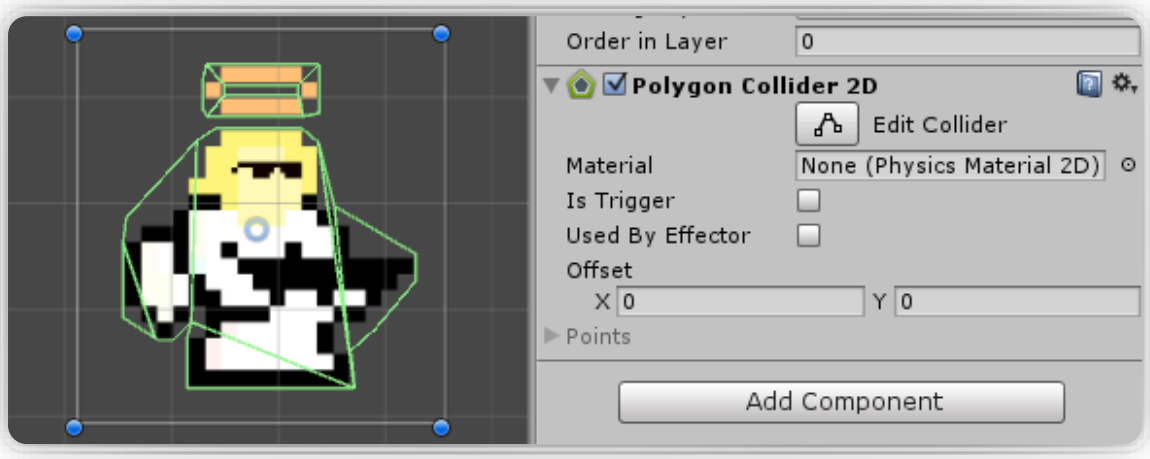
CIRCLE COLLIDER 2D

- CircleCollider2D is a flat sphere-shaped collider.
- It is often used for character's legs part to be responsible for collisions with the ground.



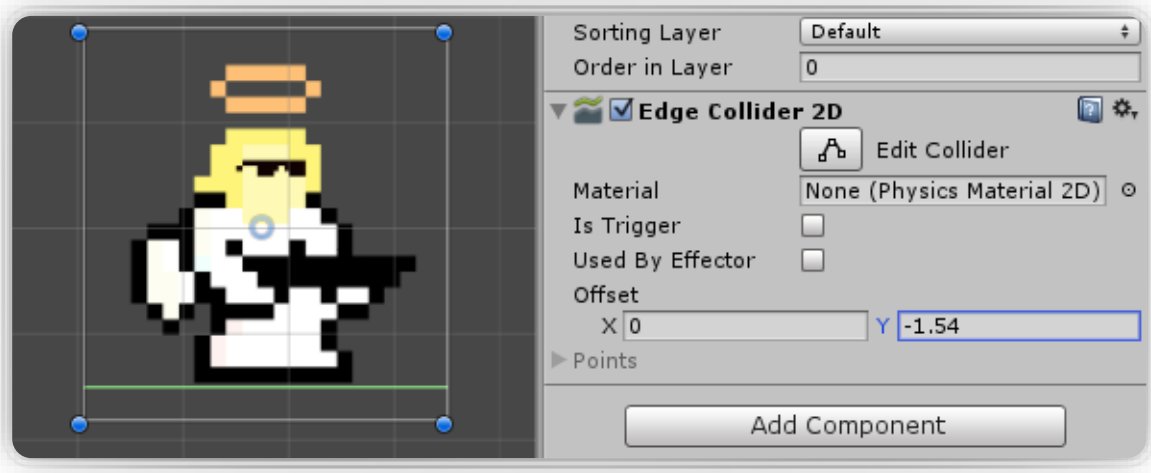
POLYGON COLLIDER 2D

- PoligonCollider2D takes the detailed shape of and object which can be edited to fit the perfect values.



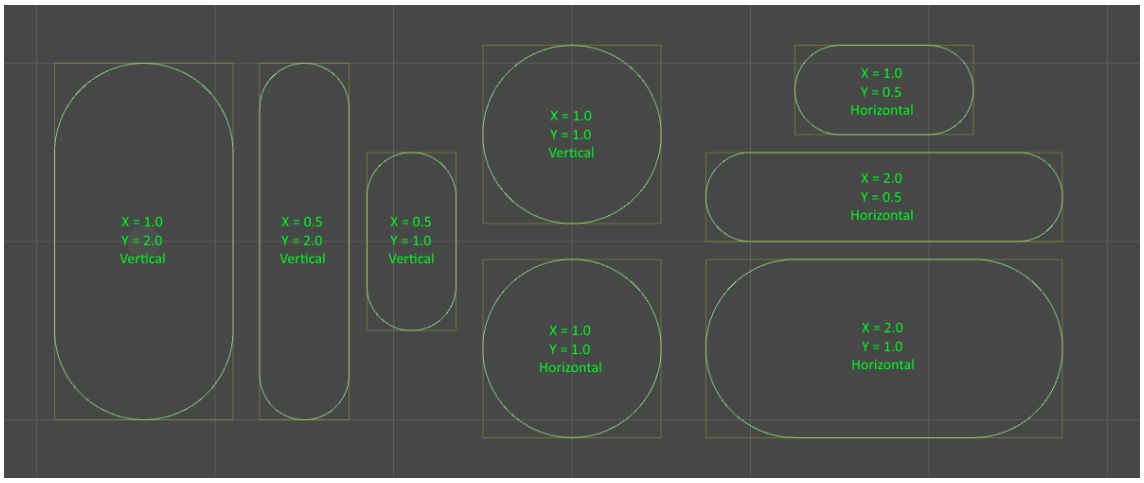
EDGE COLLIDER 2D

- Edge collider is a Unity3d collider type which is used to define object's edge that should collide with other objects.



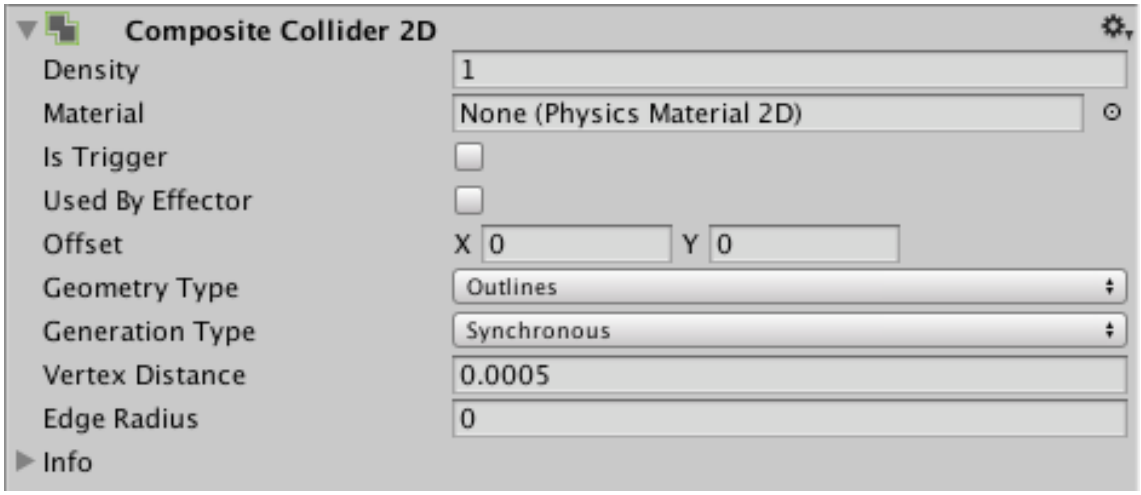
CAPSULE COLLIDER 2D

- The Capsule Collider 2D component is a 2D physics primitive that you can elongate in either a vertical or horizontal direction.
- The capsule shape has no vertex corners; it has a continuous circumference, which doesn't get easily caught on other collider corners.
- The capsule shape is solid, so any other Collider 2Ds that are fully inside the capsule are considered to be in contact with the capsule and are forced out of it over time.



COMPOSITE COLLIDER 2D

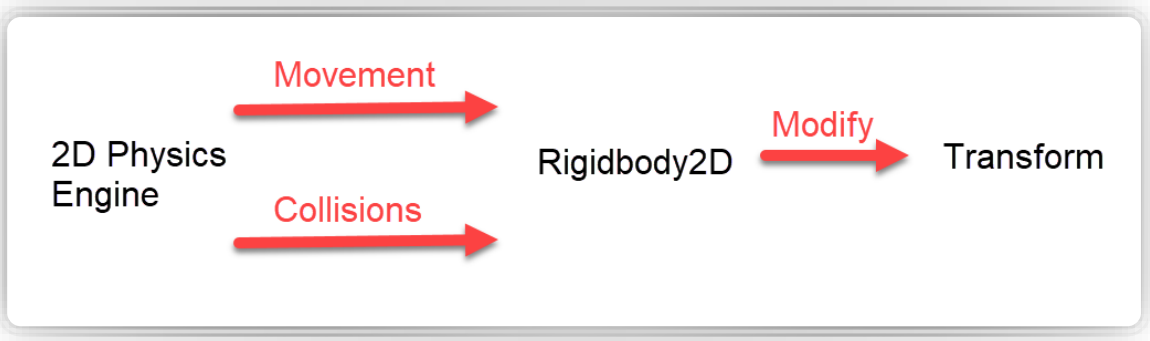
- The Composite Collider 2D merges the shapes of any Box Collider 2D or Polygon Collider 2D. It uses the vertices (geometry) from any of these Colliders, and merges them together into new geometry controlled by itself.
- The Box Collider 2D and Polygon Collider 2D components have a Used By Composite checkbox. Tick this checkbox to attach them to the Composite Collider 2D.
- These Colliders also have to be attached to the same Rigidbody 2D as the Composite Collider 2D.



RIGIDBODY 2D

- Rigidbody 2D component is for communicating Physics 2D movement of colliders back to the Transform components
- Collisions are possible even without rigidbodies, but rigidbodies will make the gameObjects behave like physical objects (gravity, forces...).

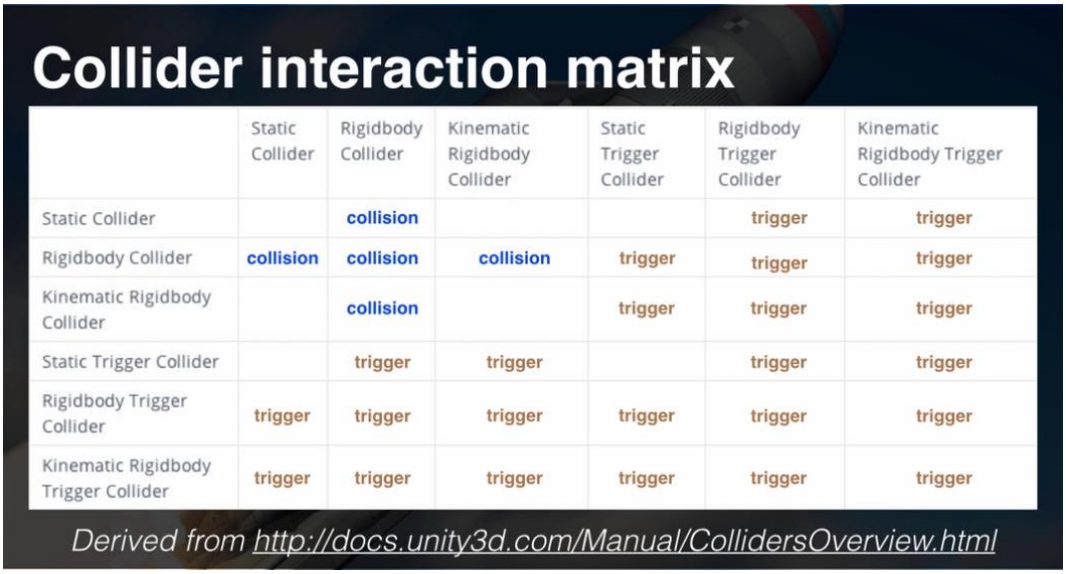
- In general if the object moves it should always have a rigidbody, because Unity will make internal optimizations.



COLLISION EVENTS

- Physics3D and Physics2D are totally independent.
- Collisions between 2 Colliders produce "OnCollisionEnter2D(Collision2D collision)"
- Collisions between 2 Colliders when at least one of them is marked as a "Trigger" will produce "OnTriggerEnter2D(Collider2D collider)"
- Additionally, if none of the game objects has a rigidbody attached you will not receive any event!

COLLIDER INTERATION MATRIX



	Static Collider	Rigidbody Collider	Kinematic Rigidbody Collider	Static Trigger Collider	Rigidbody Trigger Collider	Kinematic Rigidbody Trigger Collider
Static Collider		collision			trigger	trigger
Rigidbody Collider	collision	collision	collision	trigger	trigger	trigger
Kinematic Rigidbody Collider		collision		trigger	trigger	trigger
Static Trigger Collider		trigger	trigger		trigger	trigger
Rigidbody Trigger Collider	trigger	trigger	trigger	trigger	trigger	trigger
Kinematic Rigidbody Trigger Collider	trigger	trigger	trigger	trigger	trigger	trigger

EFFECTORS 2D

Use Effector 2D components with Collider 2D components to

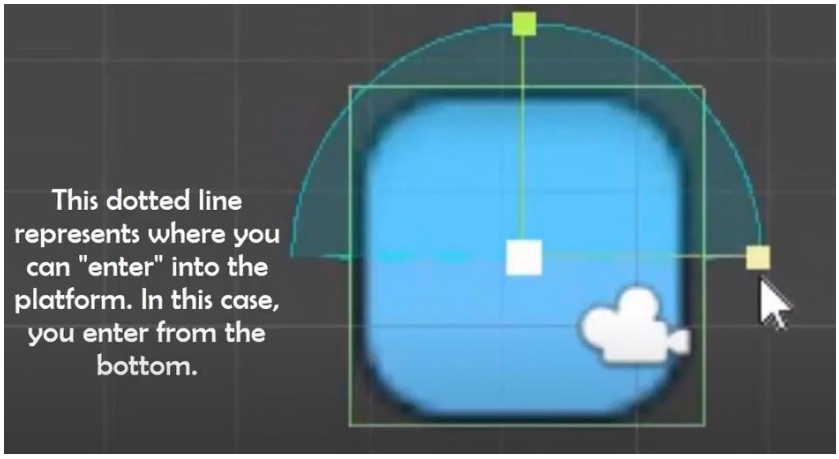
- direct the forces of physics
- when GameObject colliders come into contact with each other.

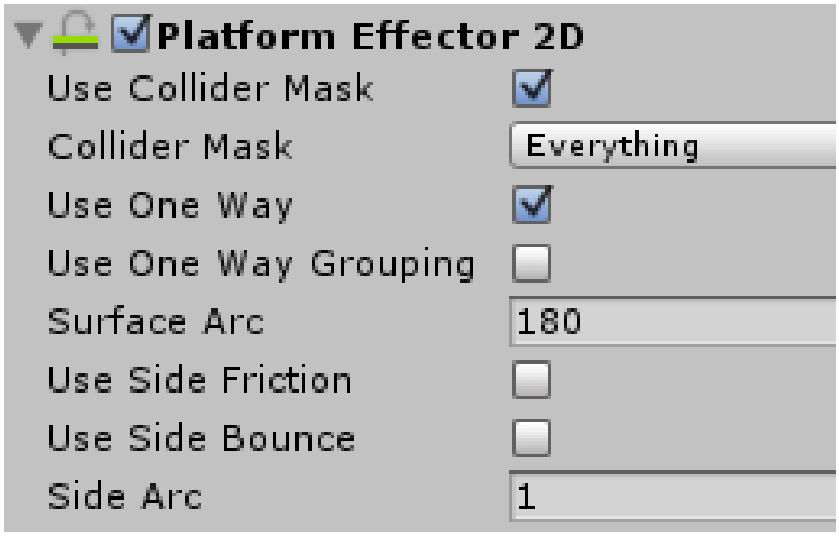
You can use Effector 2D components to create the following effects:

- Create ‘platform’ behaviour such as one-way collisions (PlatformEffector2D)
- Create conveyor belts (SurfaceEffector2D)
- Attract or repulse against a given source point (PointEffector2D)
- Make GameOjects float (BuoyancyEffector2D)
- Randomly vary force and angle magnitude (class-AreaEffector2D)

PLATFORM EFFECTOR

- Use One Way: Should one-way collision behaviour be used?
- Surface Arc: The angle of an arc centered on the local ‘up’ the defines the surface which doesn’t allow colliders to pass. Anything outside of this arc is considered for one-way collision.





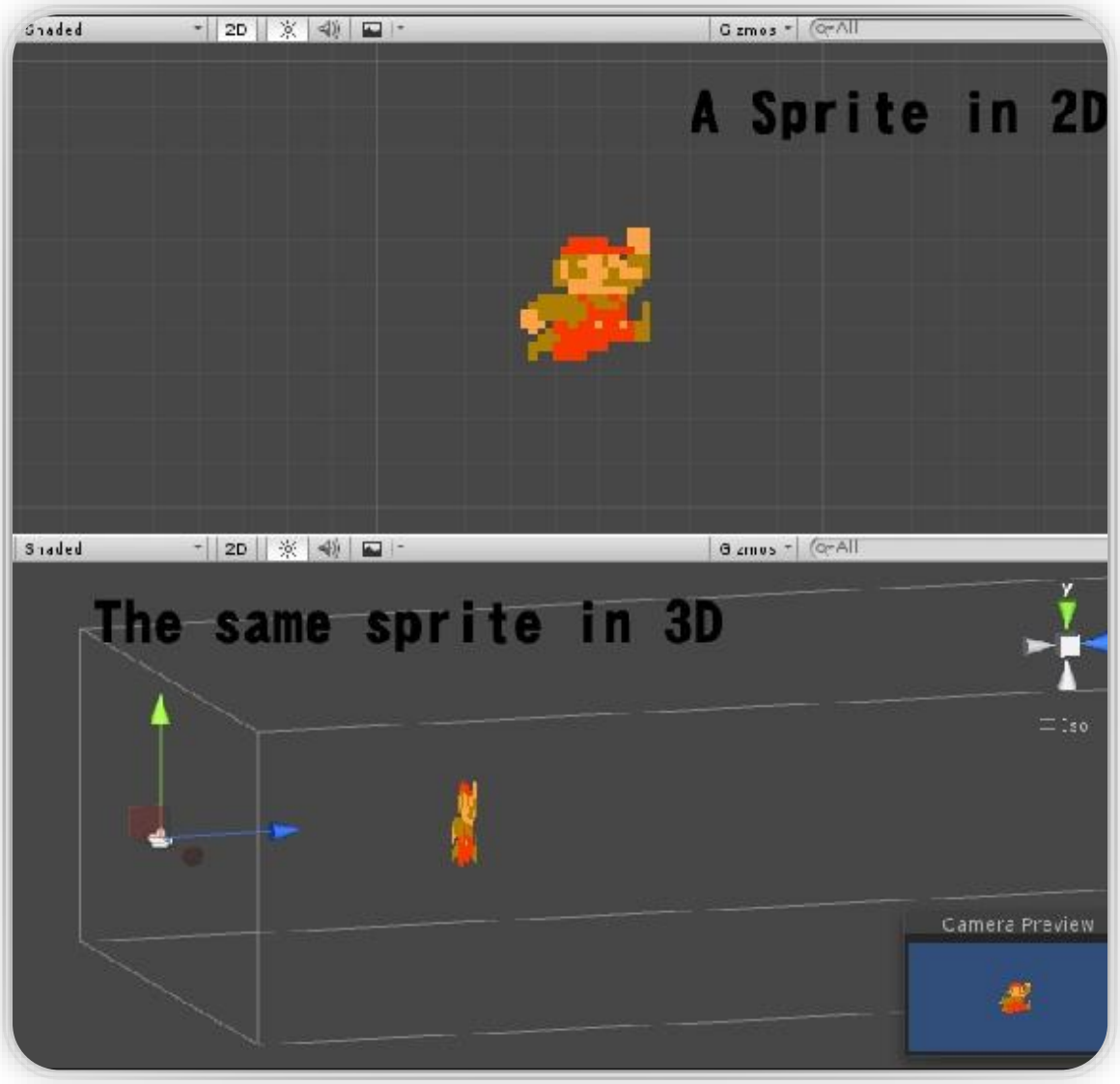
DAY 4: SPRITES AND ANIMATIONS

Sprite

Animation

SPRITE

- Sprites are simple 2D objects that have graphical images (called textures) on them.
- When viewed in 3D space, sprites will appear to be paper-thin, because they have no Z-width.

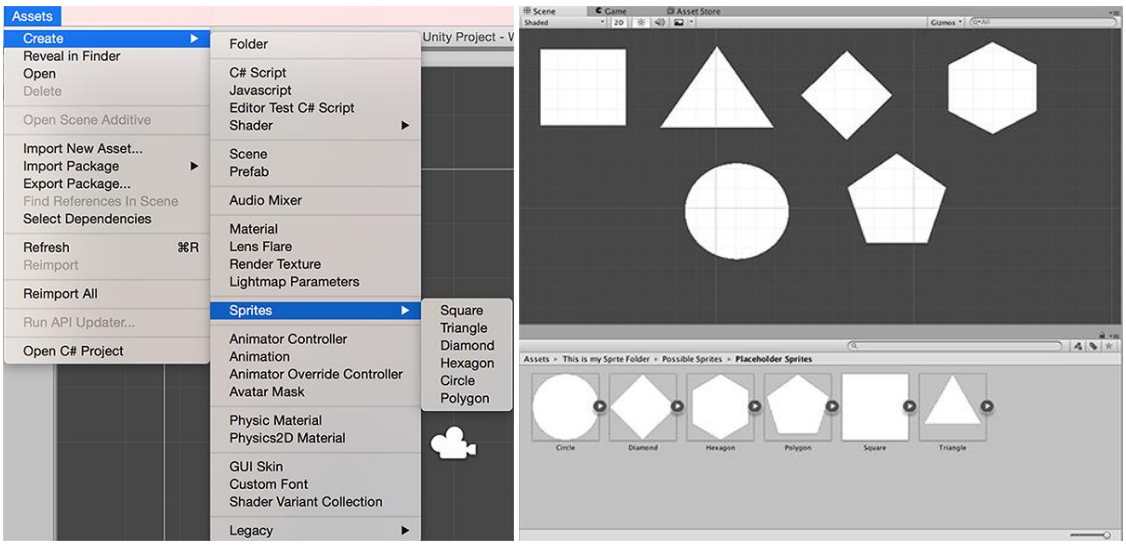


SPRITE TOOLS

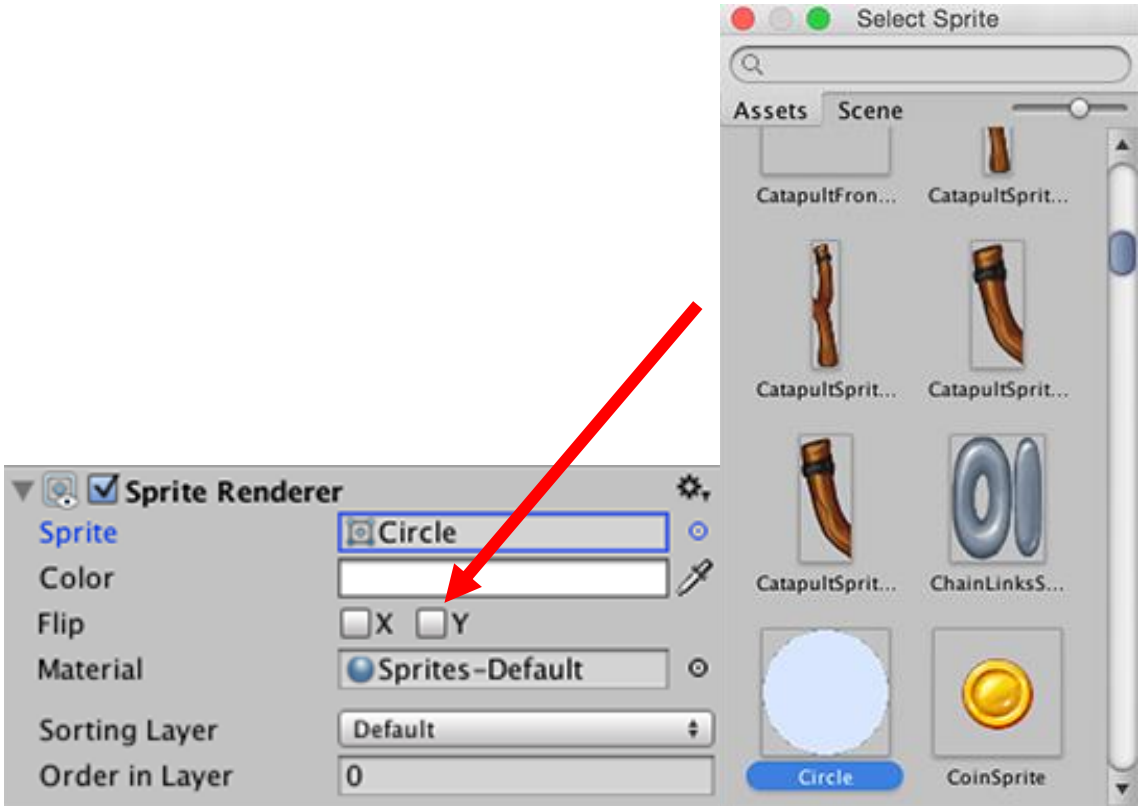
- Sprite Creator: create placeholder sprites in your project

- Sprite Editor: extract sprite graphics from a larger image and edit a number of component images within a single texture in your image editor.
- Sprite Renderer: display images as Sprites for use in both 2D and 3D scenes.
- Sprite Packer: optimize the use and performance of video memory by your project.

SPRITE CREATOR

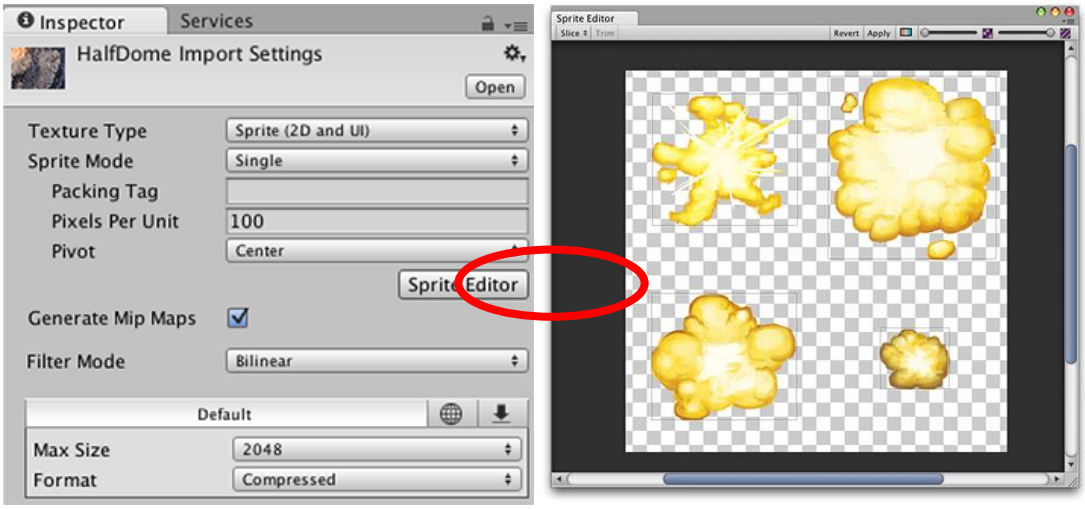


SPRITE PLACE HOLDER



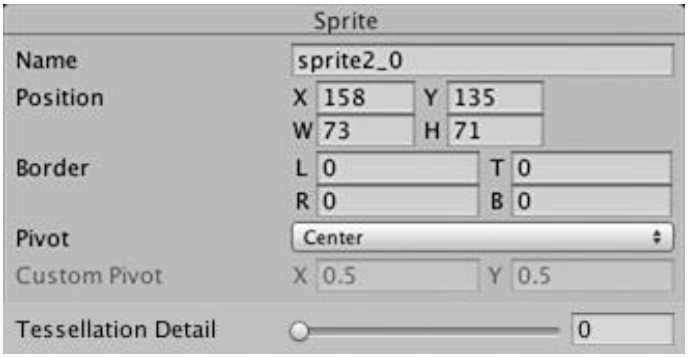
SETTING IMAGE AS SPRITE

- Click on the asset to see its Import Inspector.
- Set the Texture Type to Sprite (2D and UI)



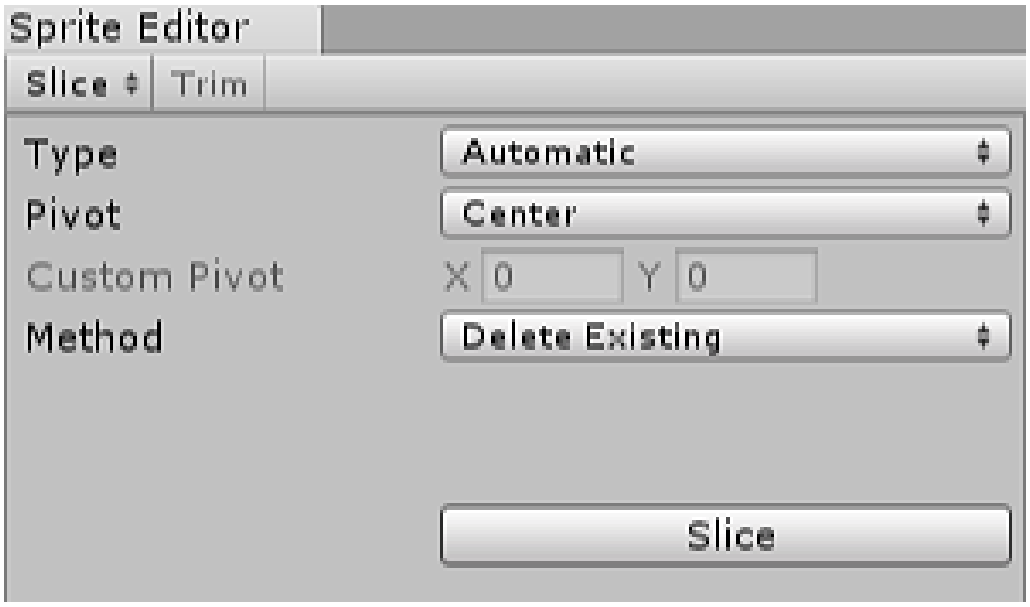
USING SPRITE EDITOR

- Click on the image, you will see a rectangular selection area appear with handles in the corners.
- You can drag the handles or the edges of the rectangle to resize it around a specific element.
- Having isolated an element, you can add another by dragging a new rectangle in a separate part of the image.
- Click on the image, you will see a rectangular selection area appear with handles in the corners.
- You can drag the handles or the edges of the rectangle to resize it around a specific element.
- Having isolated an element, you can add another by dragging a new rectangle in a separate part of the image.

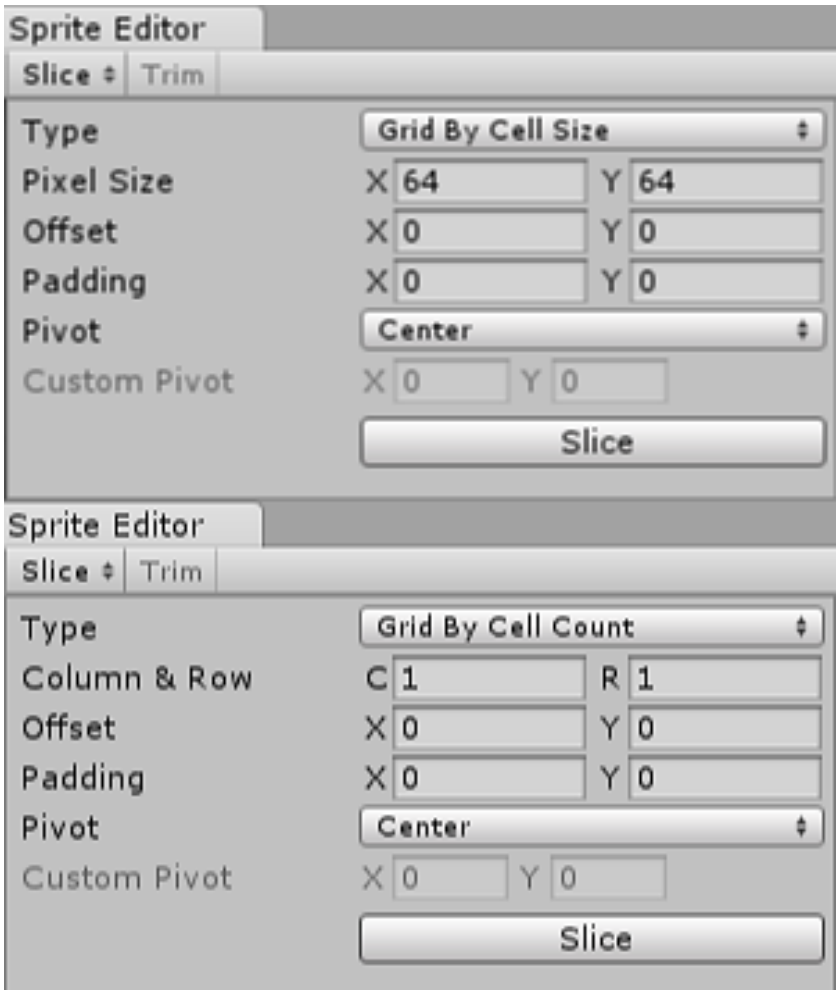


AUTOMATIC SLICING

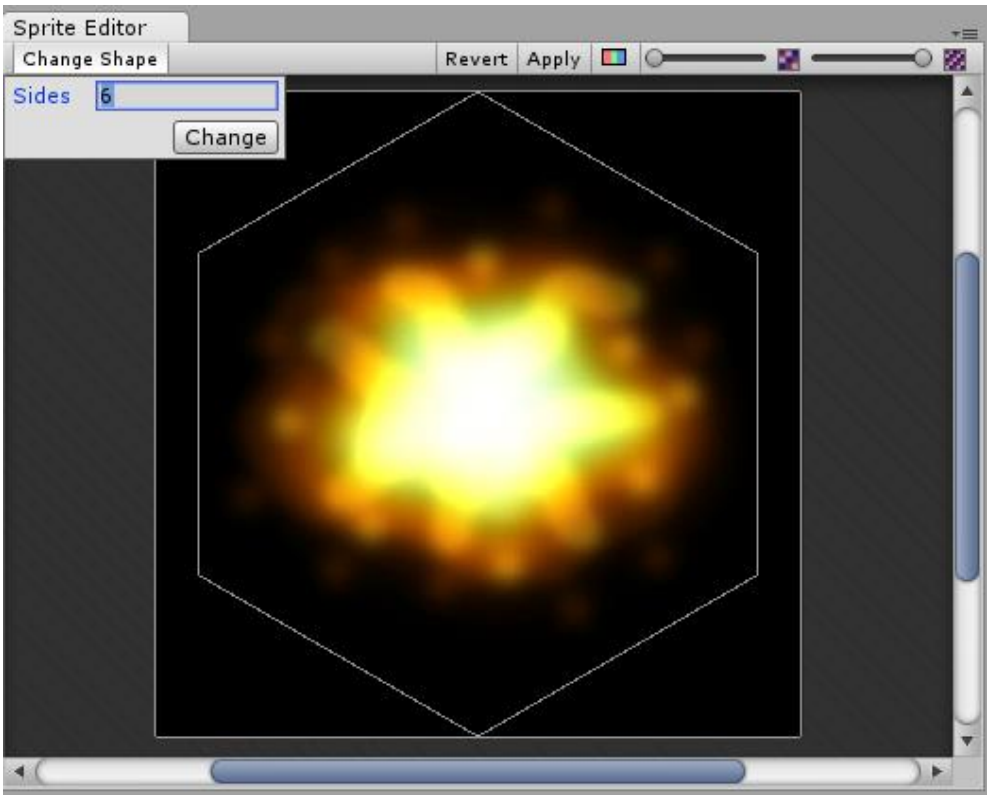
- FULL AUTOMATIC: The editor will attempt to guess the boundaries of sprite elements by transparency.



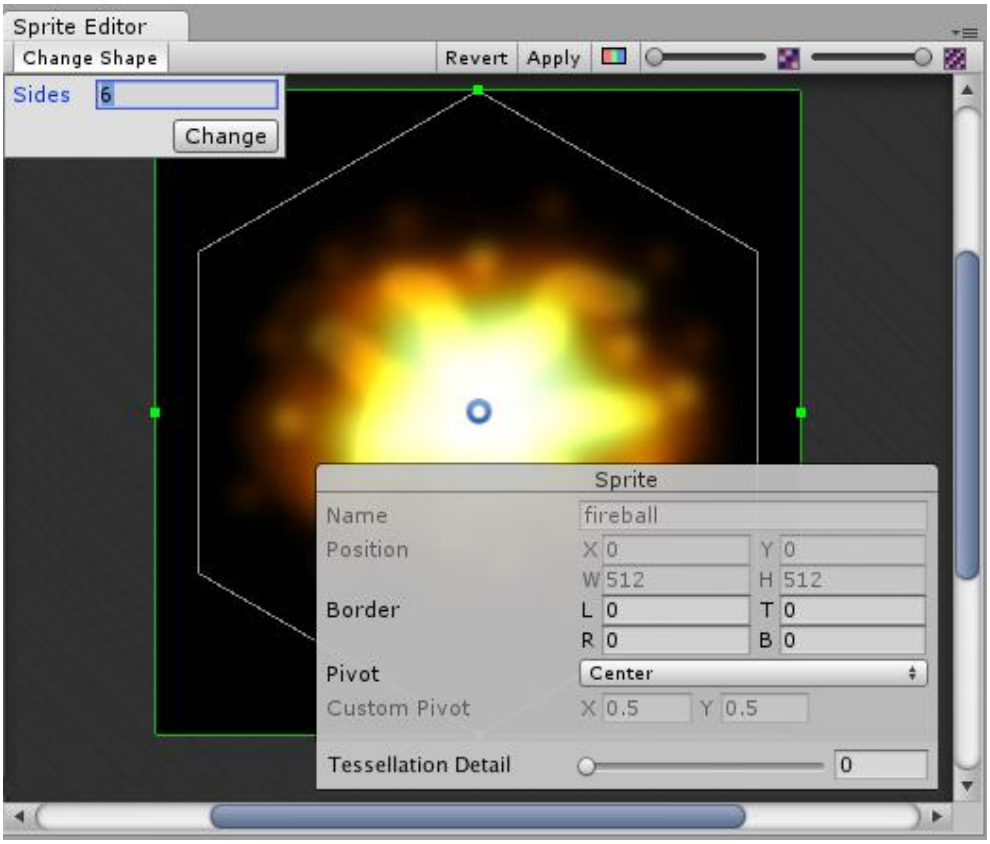
- GRID: By Cell Size / By Cell Count



POLYGON RESIZING: SHAPE



POLYGON RESIZING: SIZE & PIVOT



EDIT OUTLINE

- Use the Sprite Editor's Edit Outline option to edit the generated Mesh for a Sprite, effectively editing its outline.

TRANSPARENCY SORT MODE

- Default: Sorts based on whether the Camera's Projection mode is set to Perspective or Orthographic
- Perspective: Sorts based on perspective view. Perspective view sorts Sprites based on the distance from the Camera's position to the Sprite's center.
- Orthographic: Sorts based on orthographic view. Orthographic view sorts Sprites based on the distance along the view direction.
- Custom Axis: Sorts based on the given axis set in Transparency Sort Axis

ANIMATION SOURCE

Made by Unity (*.ANIM)

- Simple transformation (Move, Rotate and Scale)
- Simple 2D/Humanoid IK animation
- Component property modification

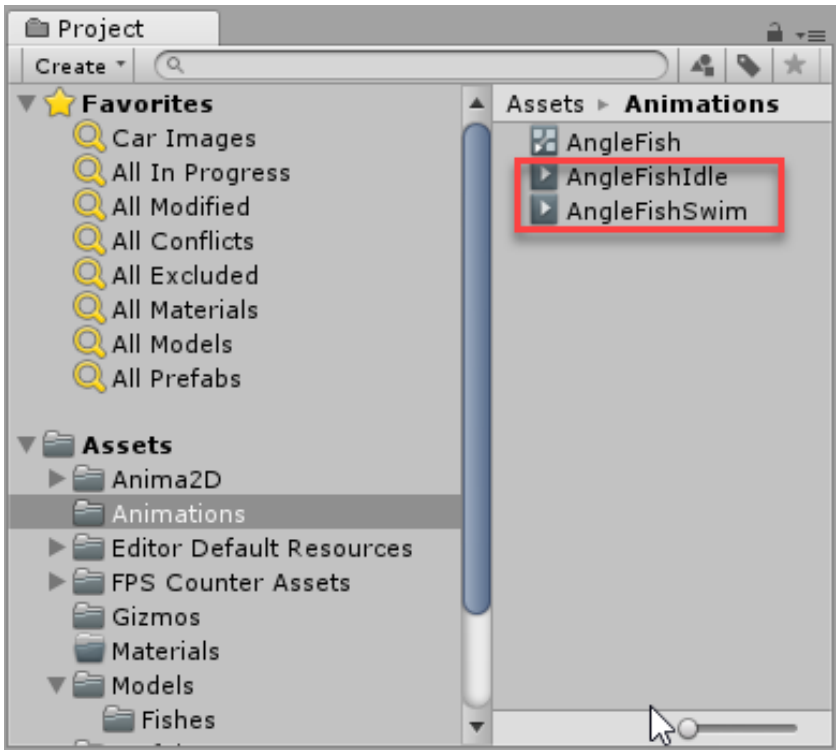
- Camera animation

Made by other software (*.FBX)

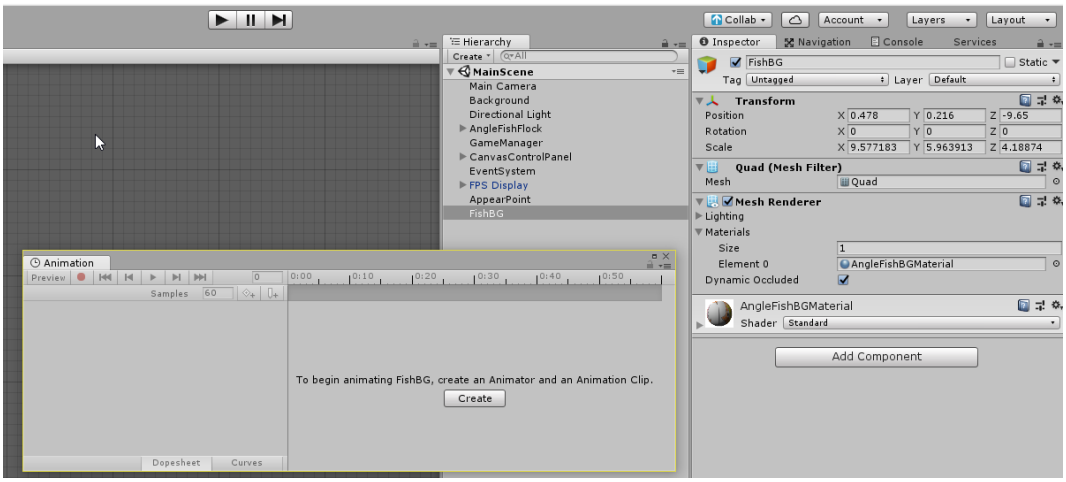
- Complicated transformation
- Generic Bones & IK animation
- Motion capture

Made by Scripting (*.cs)

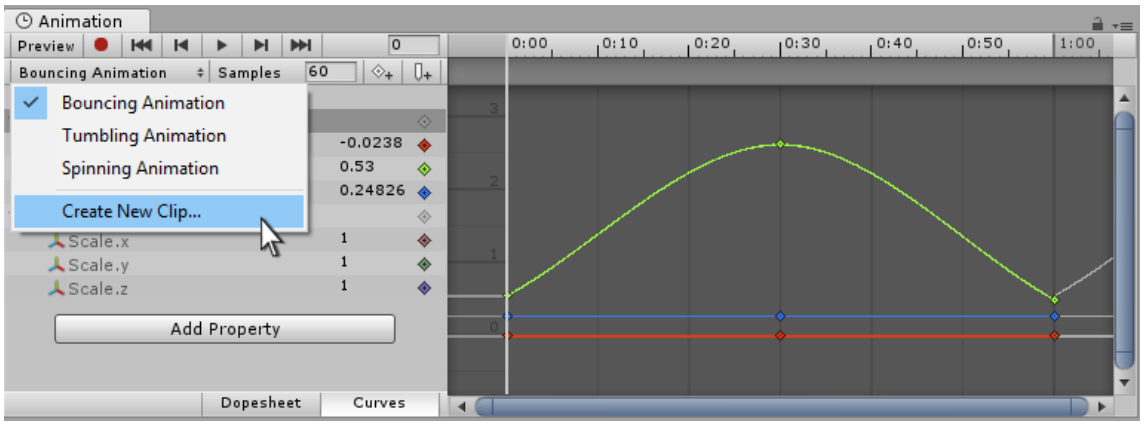
- Real-time adaption
- Based on Mathematic



CREATE FIRST ANIMATION



ADD MORE ANIMATION



ANIMATION CLIP & CONTROLLER

Animation Clip:

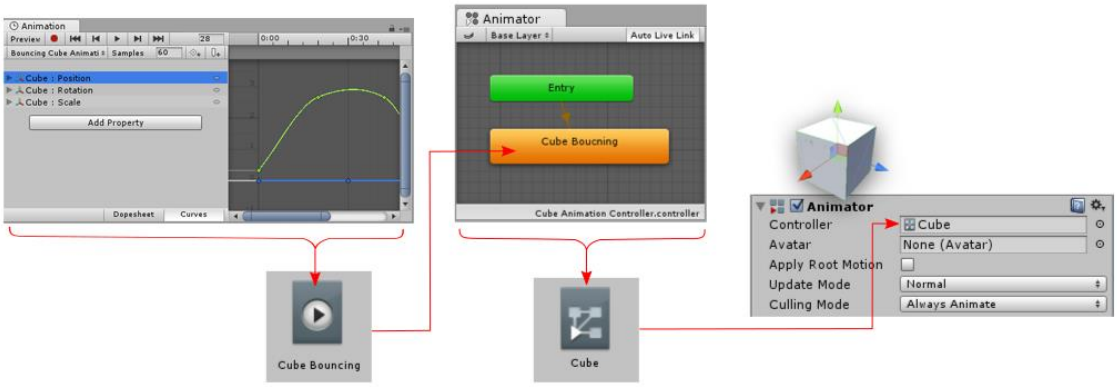
- Asset that store information of the animation (timeline, keyframe…)
- File: *.anim

Animation Controller:

- Assets manage animation states & transitions
- File: *.controller

Animator:

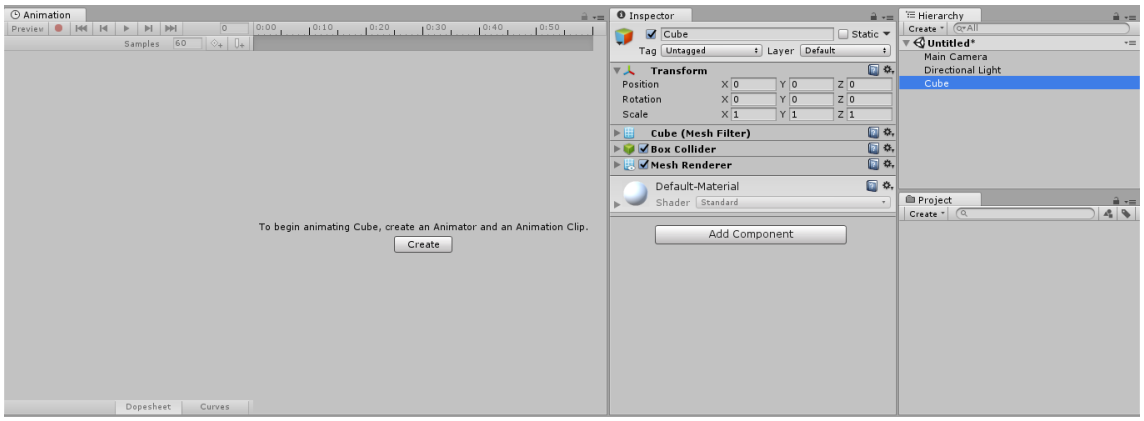
- Component that link to and play Animation Controller



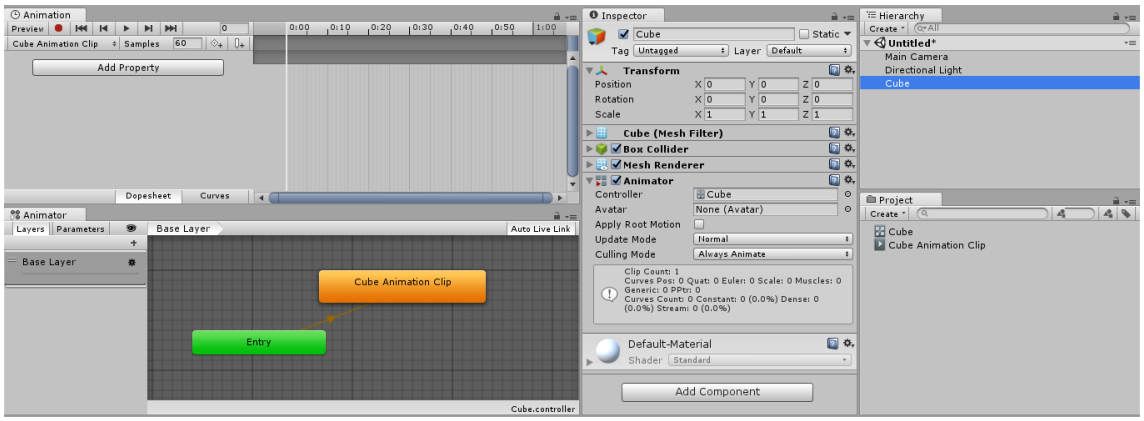
ANIMATION CLIP & CONTROLLER

- A GameObject must have an Animator component
- The Animator component must have an Animator Controller asset assigned
- The Animator Controller asset must have one or more Animation Clips assigned

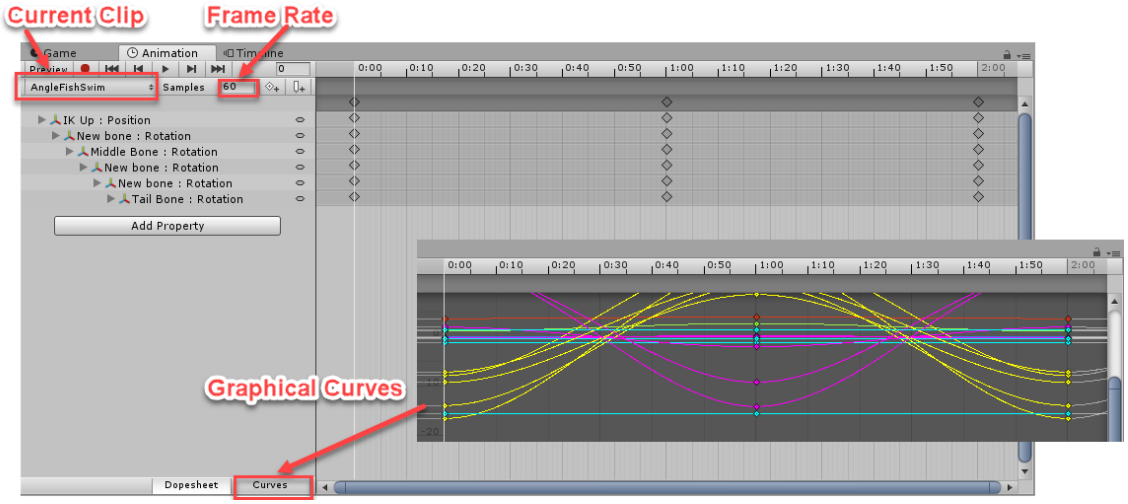
BEFORE ADDNING ANIMATION



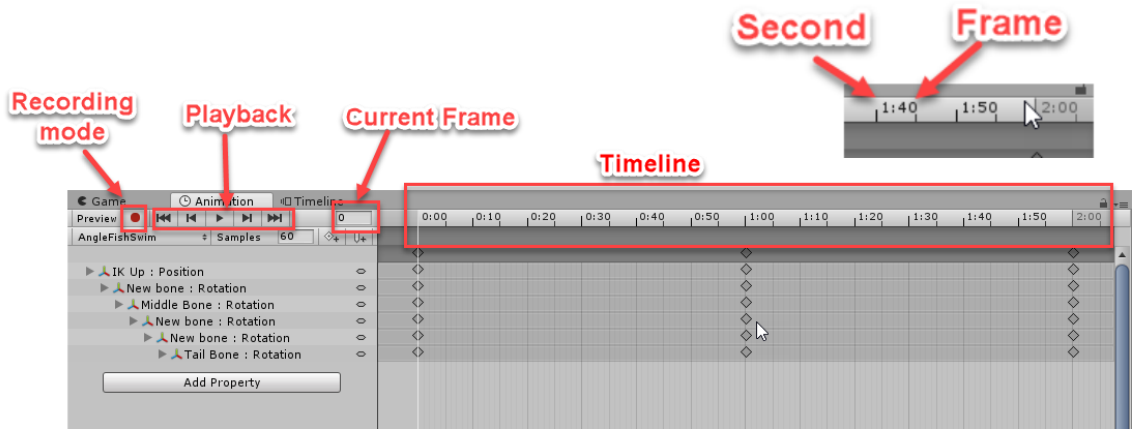
AFTER ADDING ANIMATION



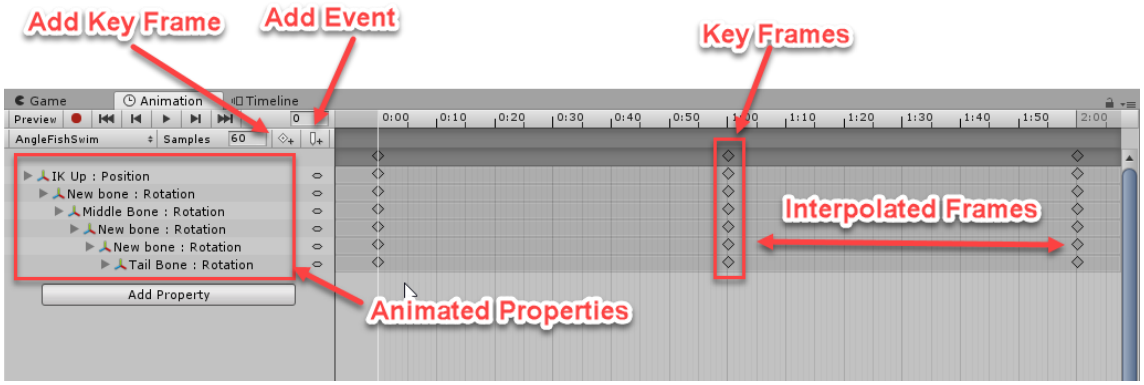
ANIMATION WINDOW



ANIMATION TIME LINE



PROPERTIES & KEY FRAMES



Animating A Game Object

- Record Mode: Unity automatically creates keyframes at the playback head when you move, rotate, or otherwise modify any animatable property on your animated GameObject.

- Preview Mode: You must manually create keyframes each time you modify your GameObject to a desired new state.

Recoring Key Frames

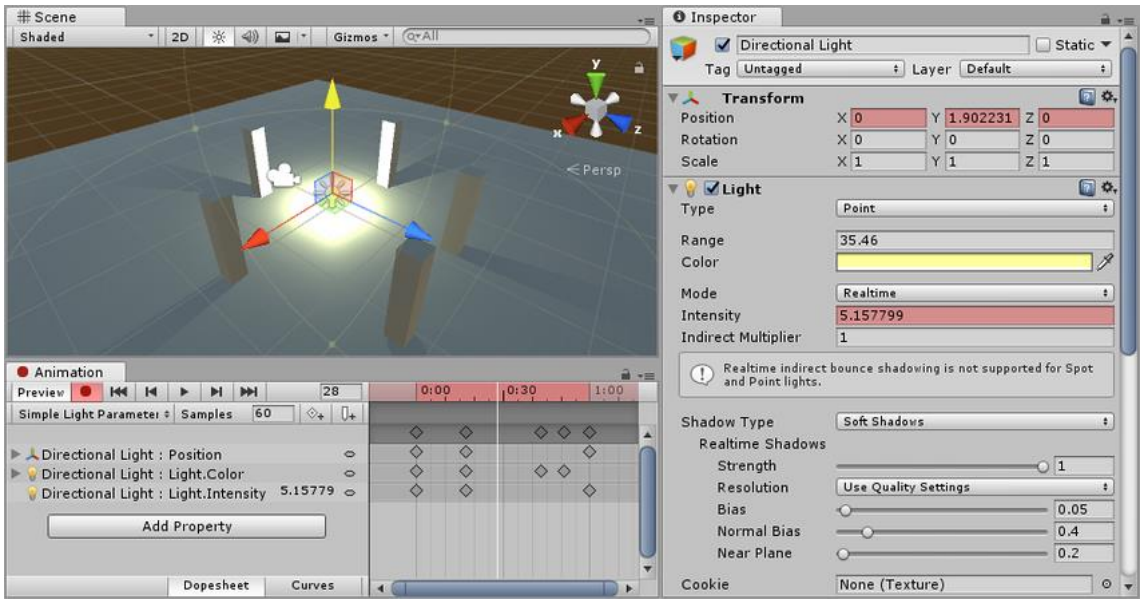
- In Record mode, you can create keyframes by
    - Setting the white Playback head to the desired time
- Then modify your GameObject

- The changes you make to the GameObject are recorded as keyframes at the current time shown by the white line.
- Any change to an animatable property will cause a keyframe for that property to appear.

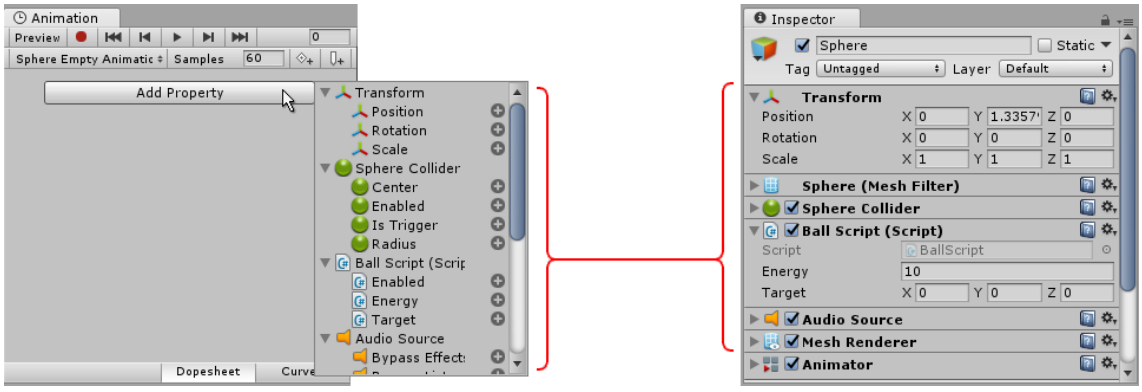
Clicking or dragging in the time line bar will

Moves the playback head

Shows the state of the animation at the playback head's current time.

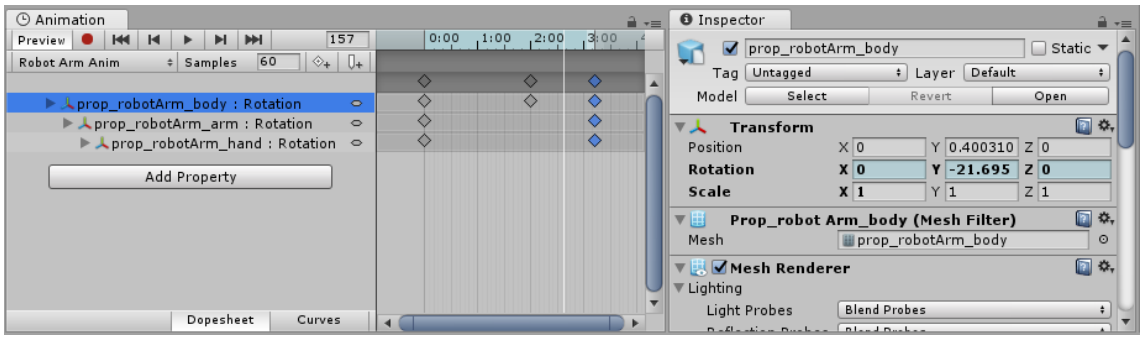


ADD ANIMATABLE PROPERTIES



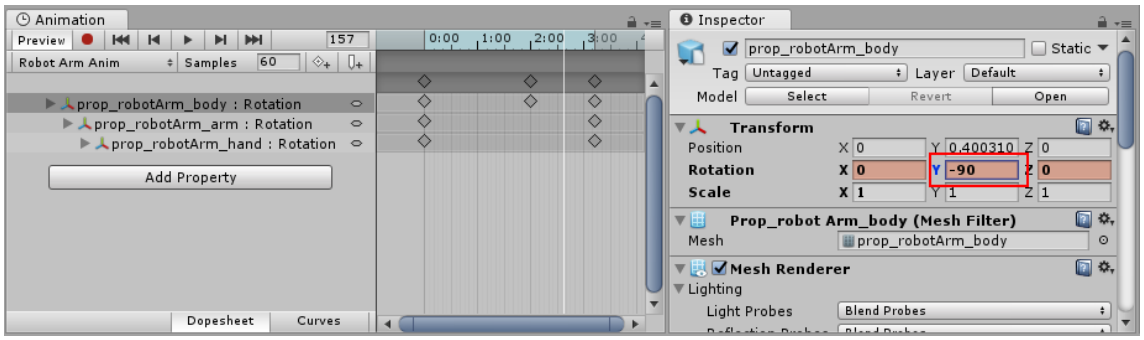
PREVIEW MODE

- In preview mode, animated properties are tinted blue in the Inspector window.



CHANGES IN PREVIEW MODE

- If you modify any of these blue-tinted properties while previewing, the GameObject is now in a **modified** animation state.



- Because you are not in record mode, your modification is not yet saved as a keyframe.
- You must manually create a keyframe to “save” this modification.

DAY 5: USER INTERFACE (UI)

Canvas

RectTransform

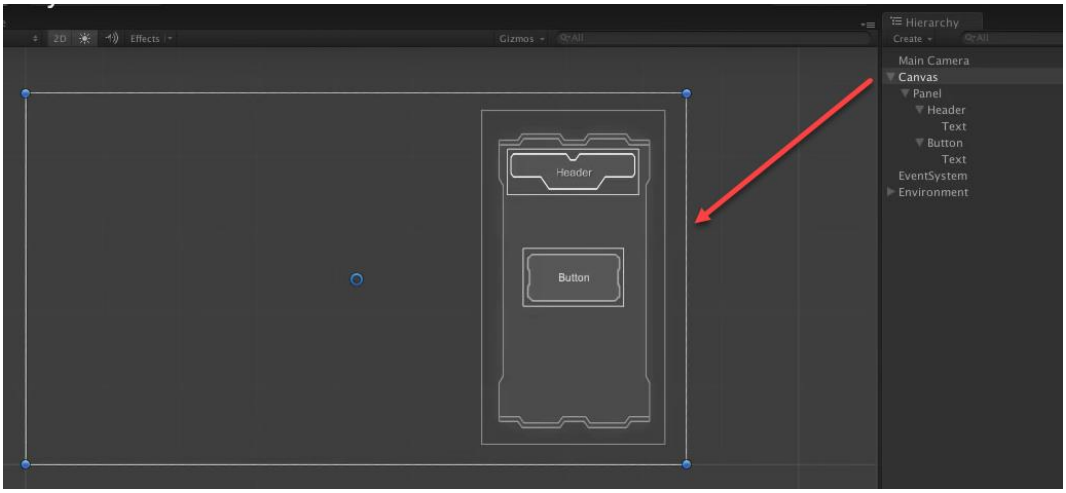
Text

Image

Button

CANVAS COMPONENT

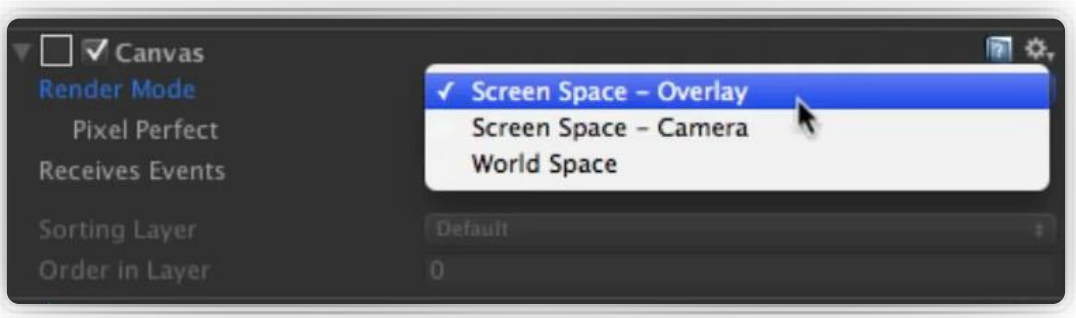
- Canvas is the container for any UI component
- No UI component can be rendered outside canvas.



RENDER MODE

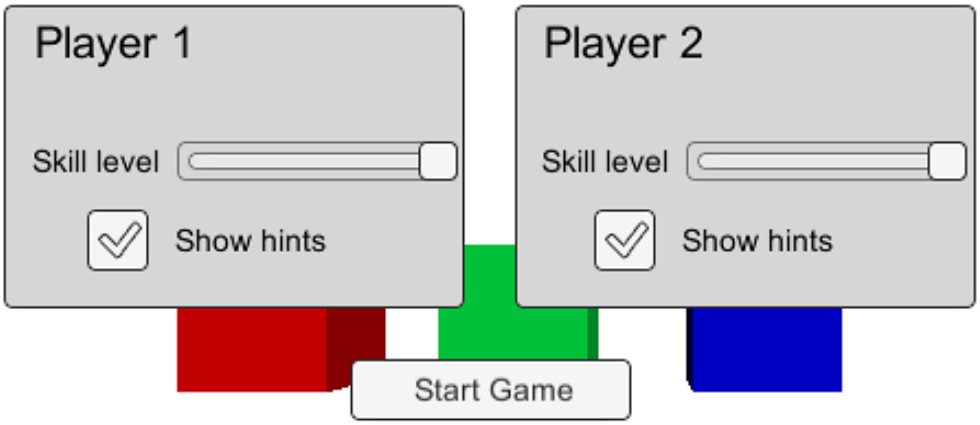
Canvas has 3 render modes:

- Screen Space - Overlay
- Screen Space - Camera Space
- World Space



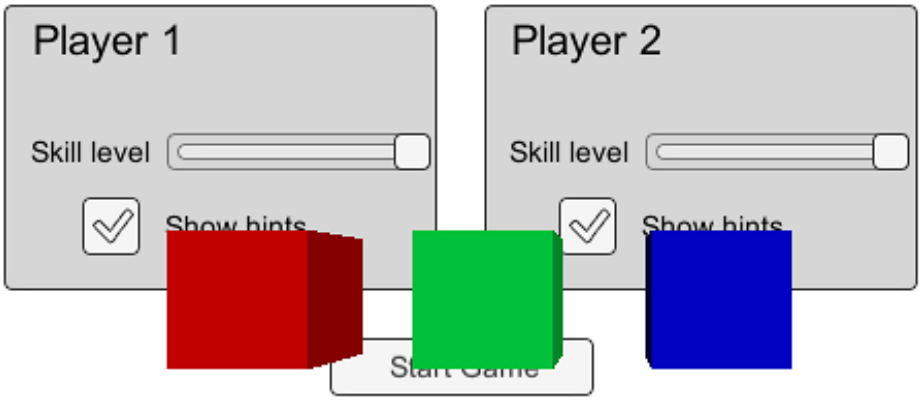
SCREEN SPACE – OVERLAY

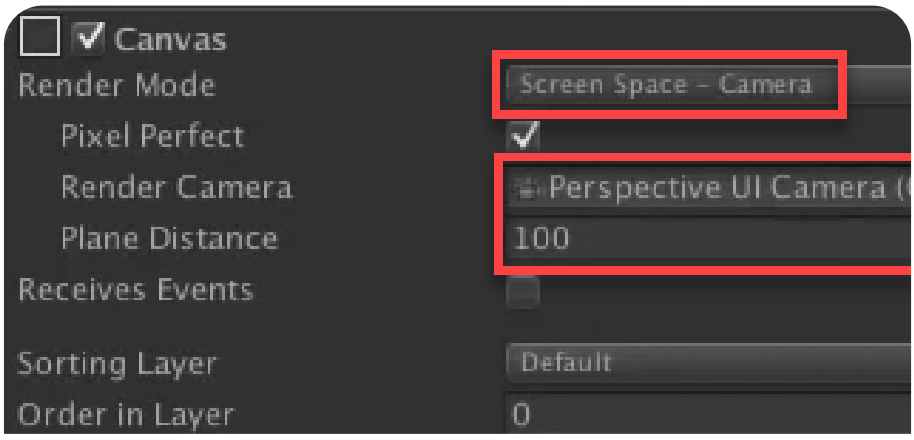
- In Screen space mode, canvas will be auto resized to fit entire game screen.
- In Overlay mode, canvas will be drawn after all cameras has been rendered.
- So, canvas is always on the top layer.



SCREEN SPACE – CAMERA

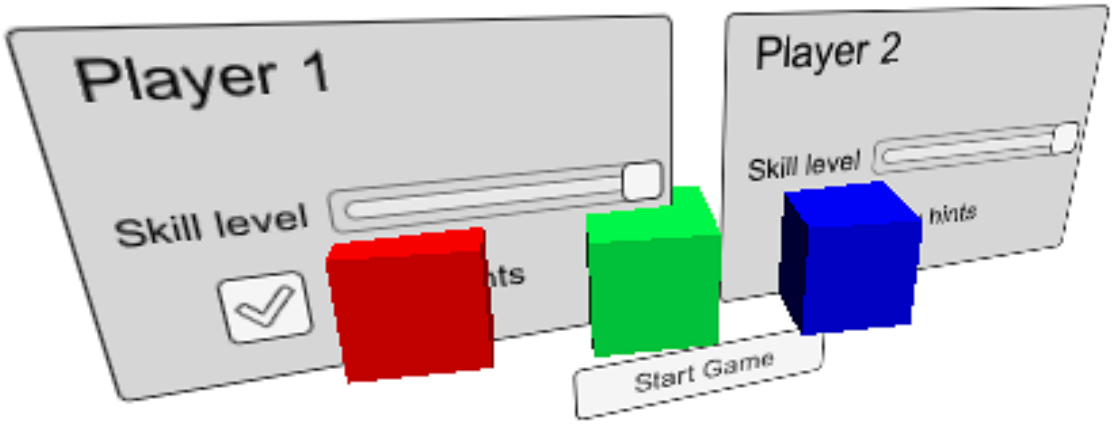
- Screen space - Camera allows rendering via a specific camera
- So, we can blend 2D with 3D objects

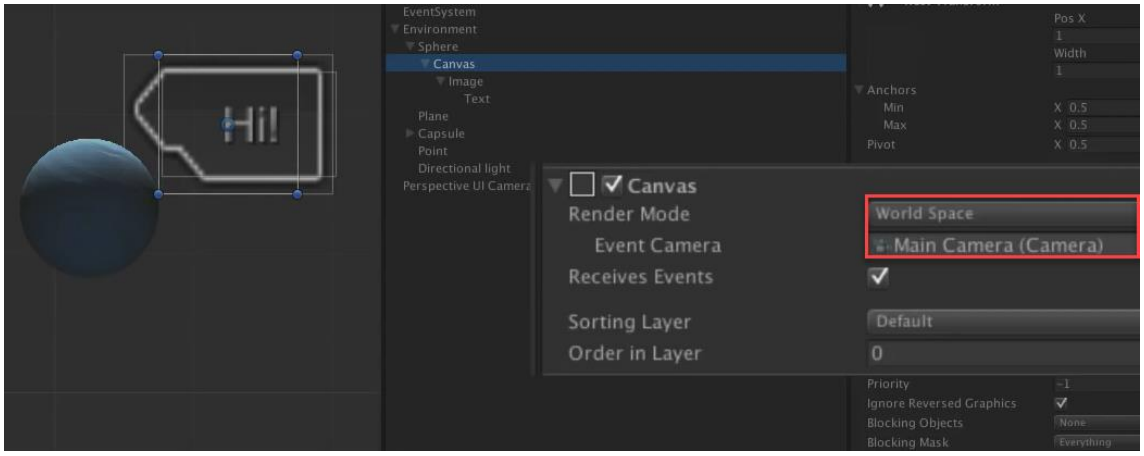




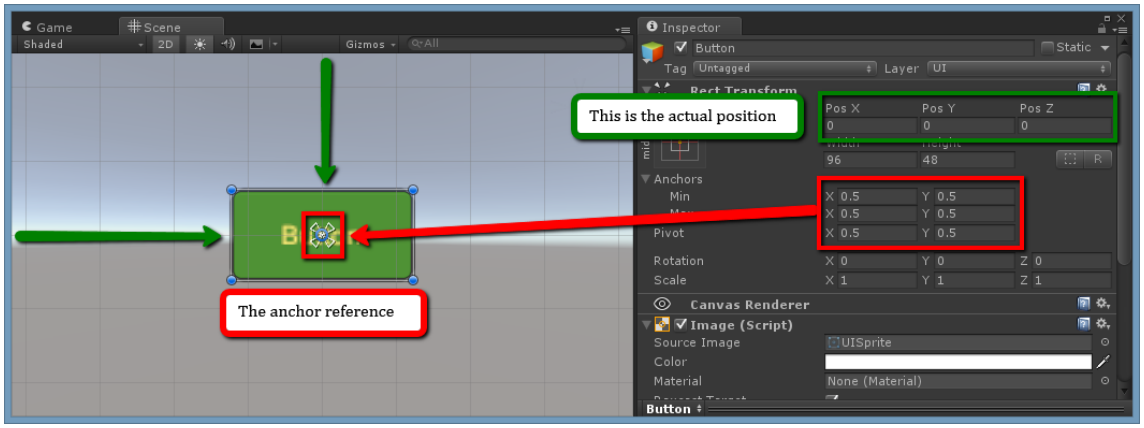
WORLD SPACE

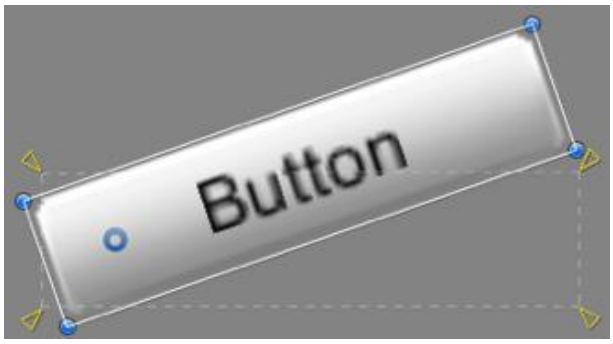
- In World Space mode, you can resize the canvas and treat it like any 3D plan.
- In VR game, all canvas should be in World Space mode.

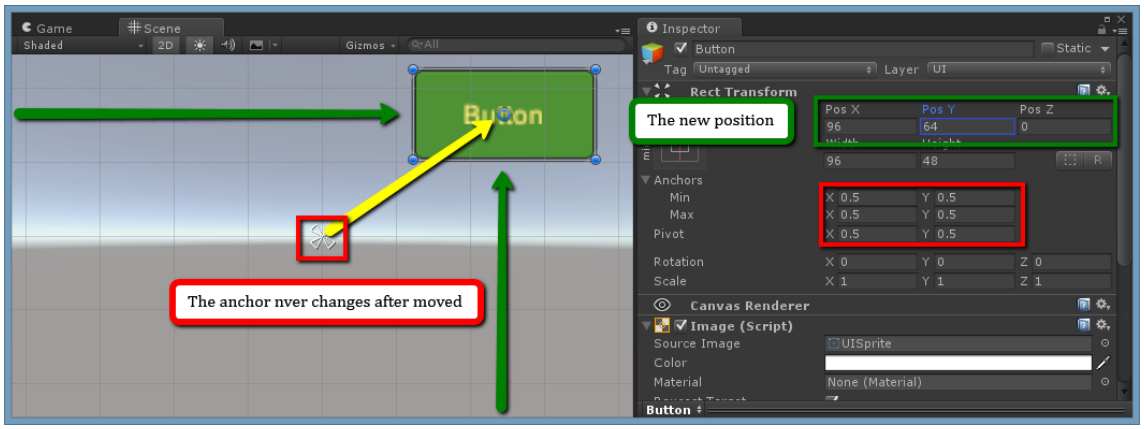


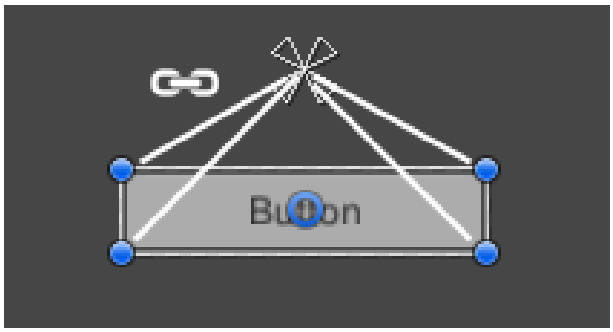


RECT TRANSFORM



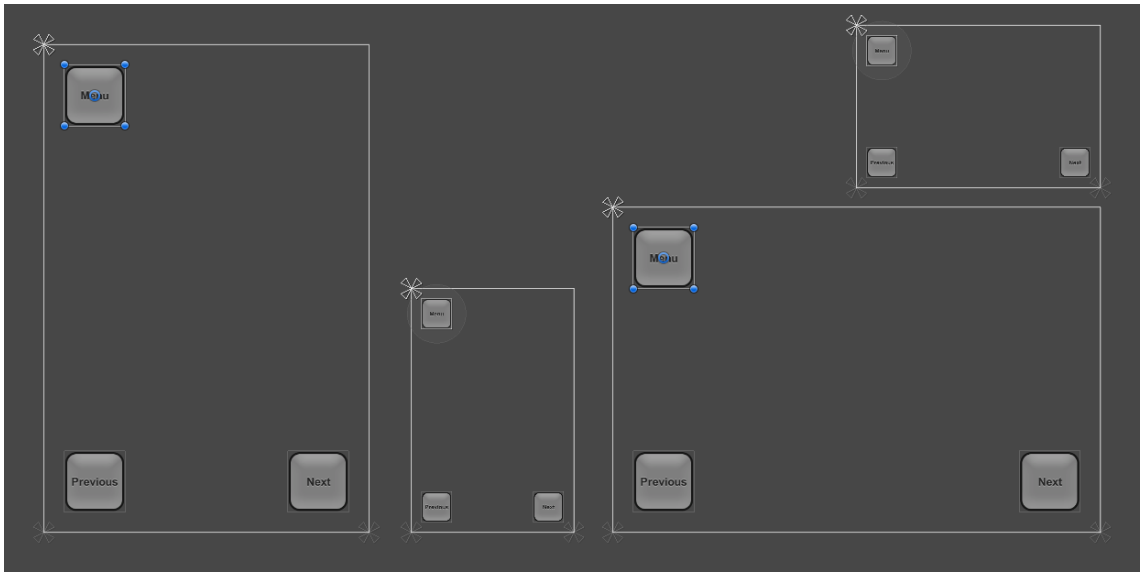






DESIGN FOR MULTIPLE RESOLUTIONS

- Anchors help UI layout be more consistent on different screen resolutions



UI IMAGE

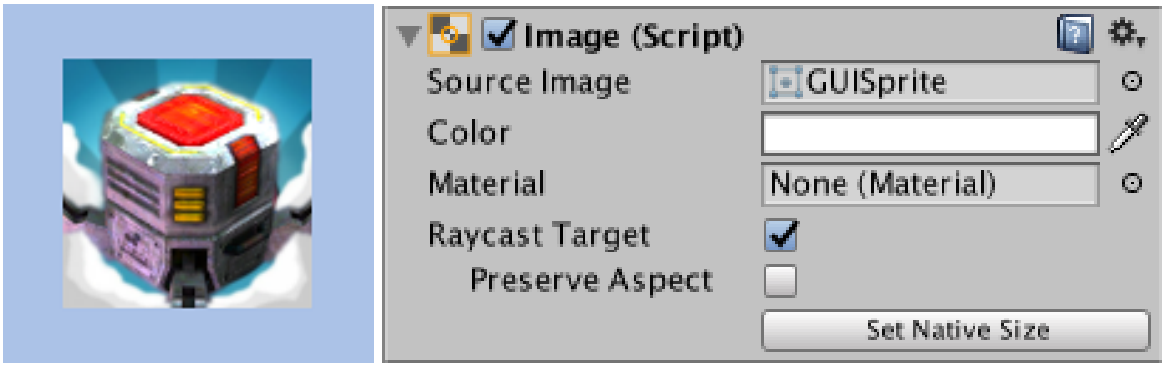
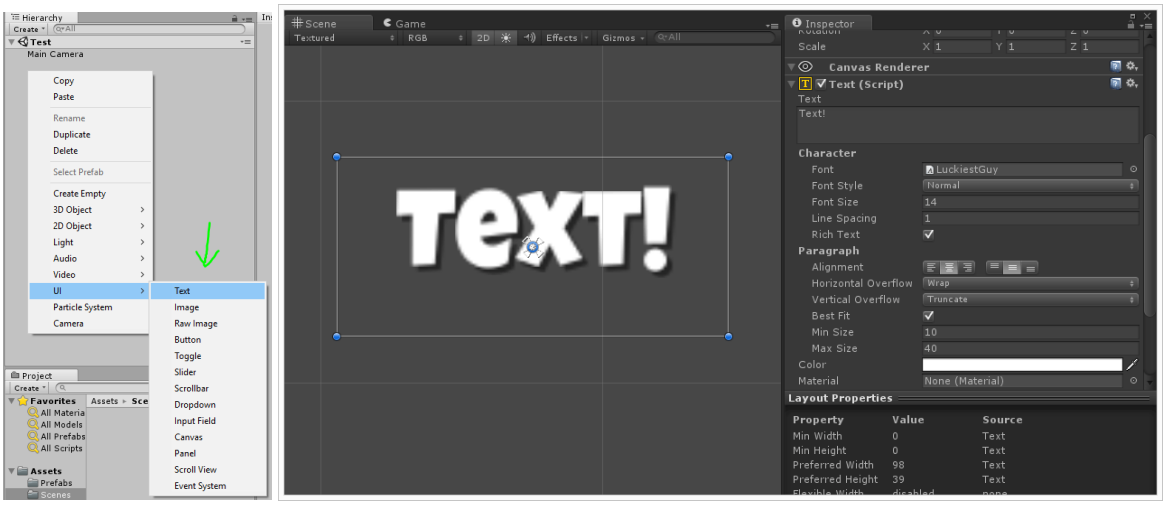


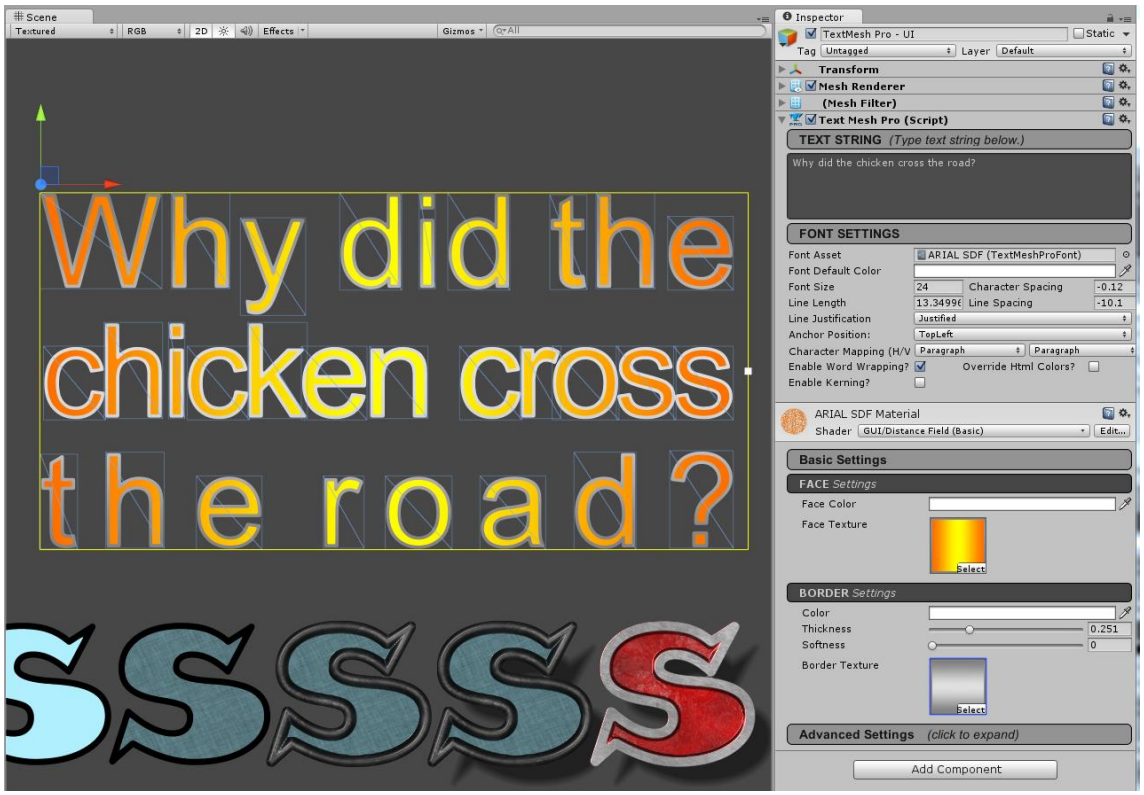
IMAGE VS SPRITE

- Image uses sprite as rendering source
- Image must be in a Canvas while Sprite is natively rendered by 3D Camera
- Image works with Rect Transform => easier layout

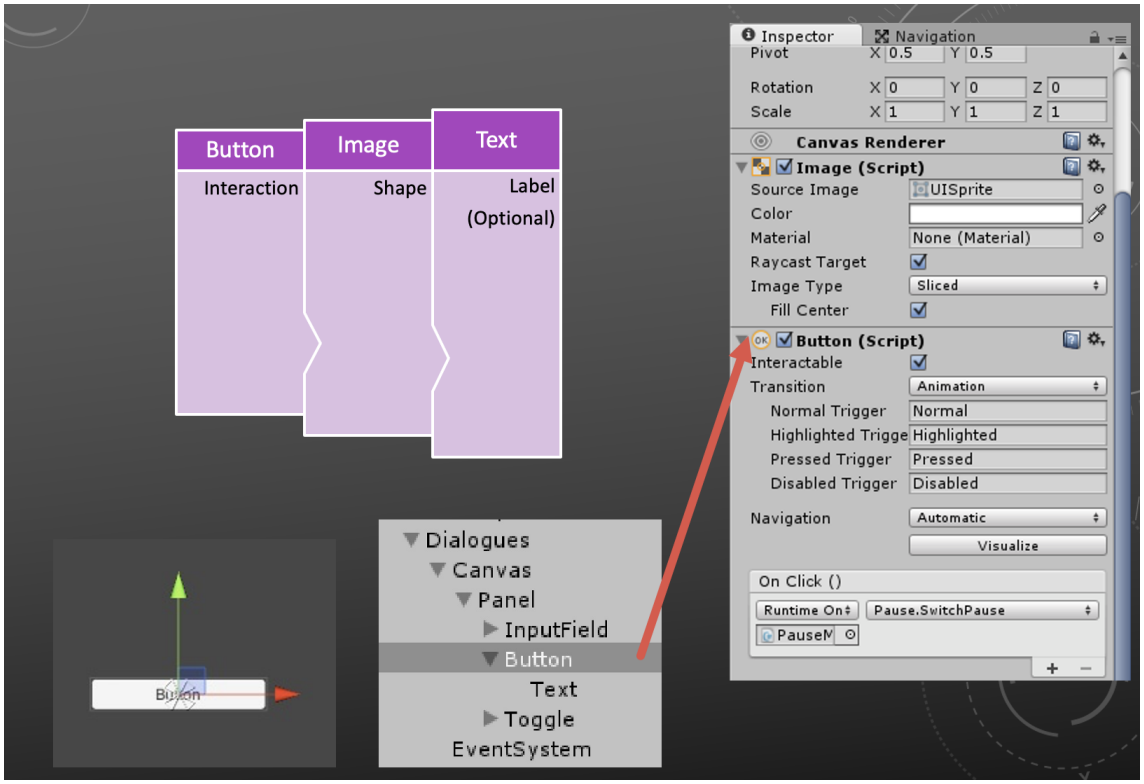
TEXT



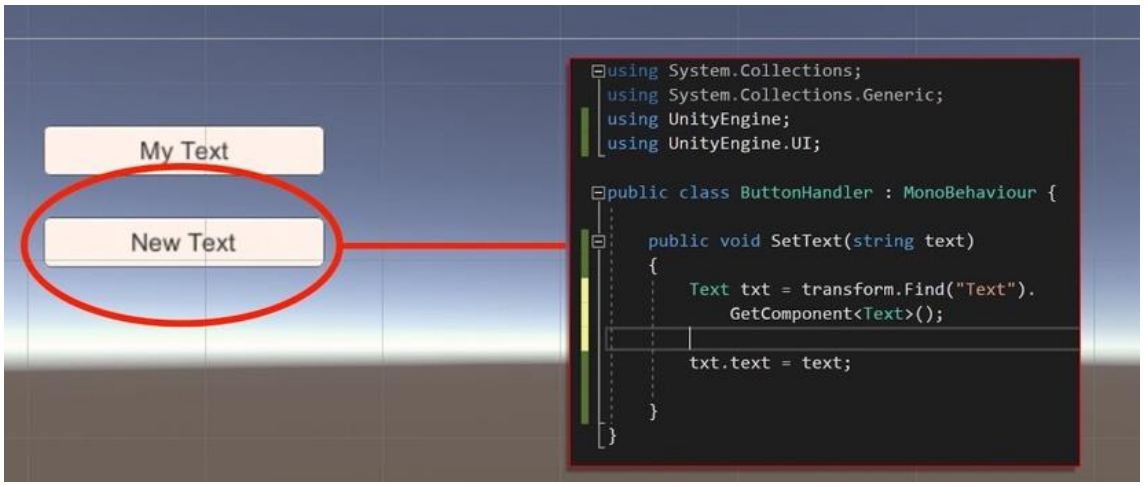
TEXT MESH PRO

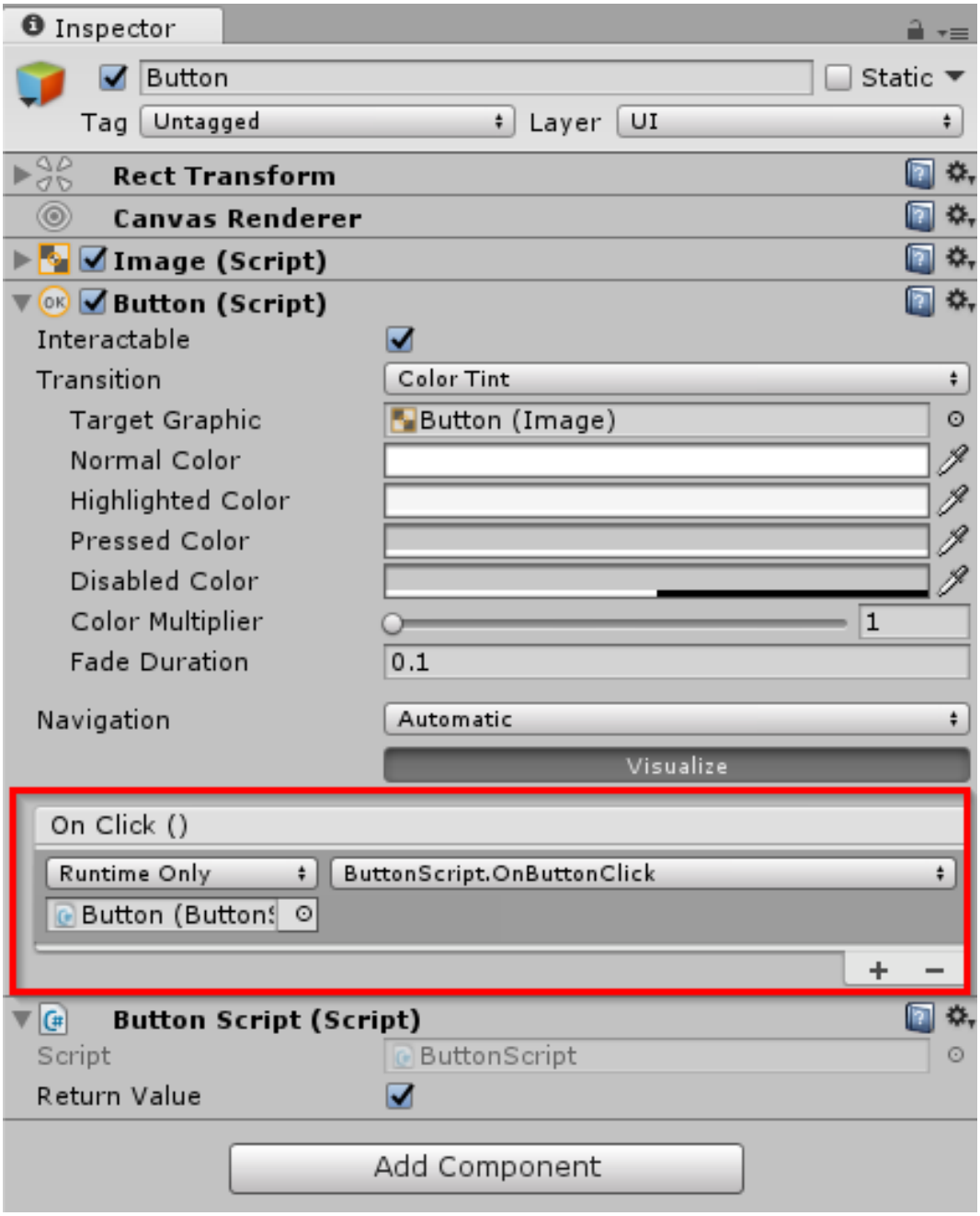


BUTTON

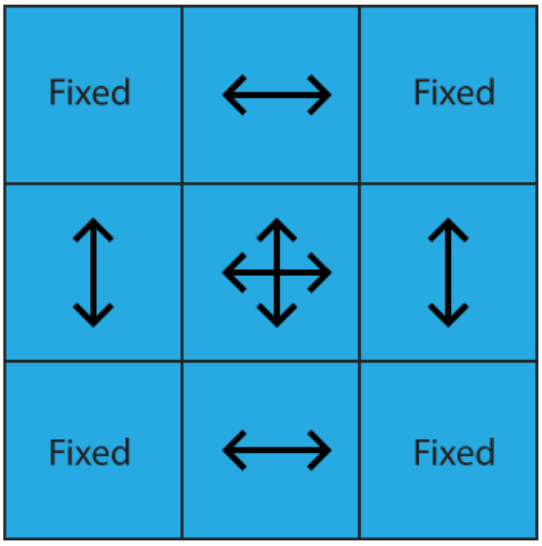


BUTTON HANDLER


My Text
New Text
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;
public class ButtonHandler : MonoBehaviour {
public void SetText(string text)
{
Text txt = transform.Find("Text").
GetComponent<Text>();
txt.text = text;
}
}


Inspector
Button
Static
Tag
Untagged
Layer
UI
Rect Transform
Canvas Renderer
Image (Script)
Button (Script)
Interactable
Transition
Color Tint
Target Graphic
Button (Image)
Normal Color
Highlighted Color
Pressed Color
Disabled Color
Color Multiplier
1
Fade Duration
0.1
Navigation
Automatic
Visualize
On Click ()
Runtime Only
ButtonScript.OnButtonClick
Button (Button
Button Script (Script)
Script
ButtonScript
Return Value
Add Component

BUTTON STRETCH



DAY 6: SCRIPTING

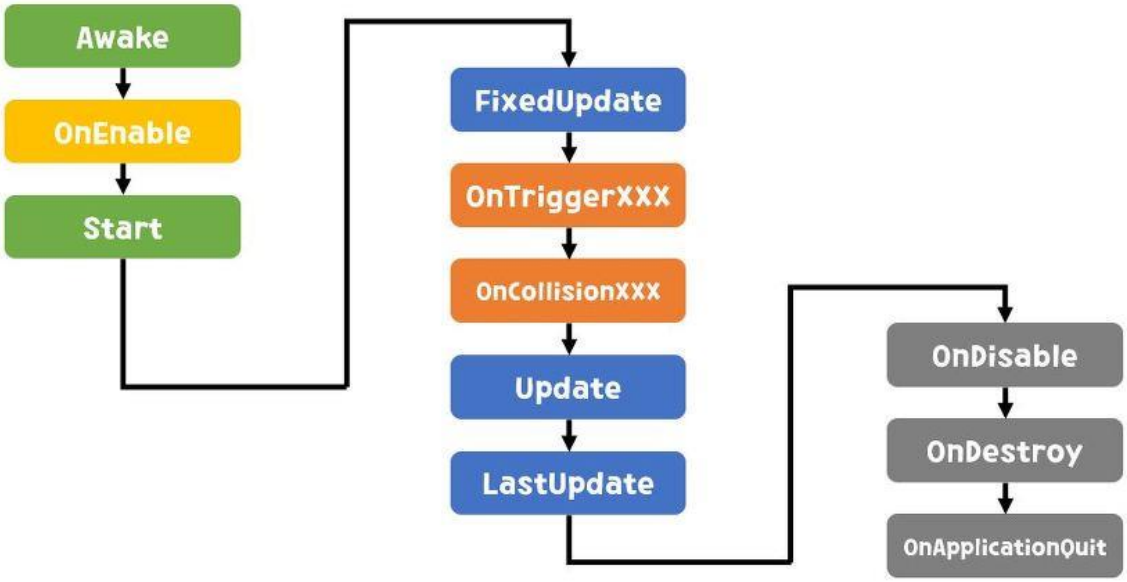
Component lifecycle

Vector and Quaternion

Interpolate Movement

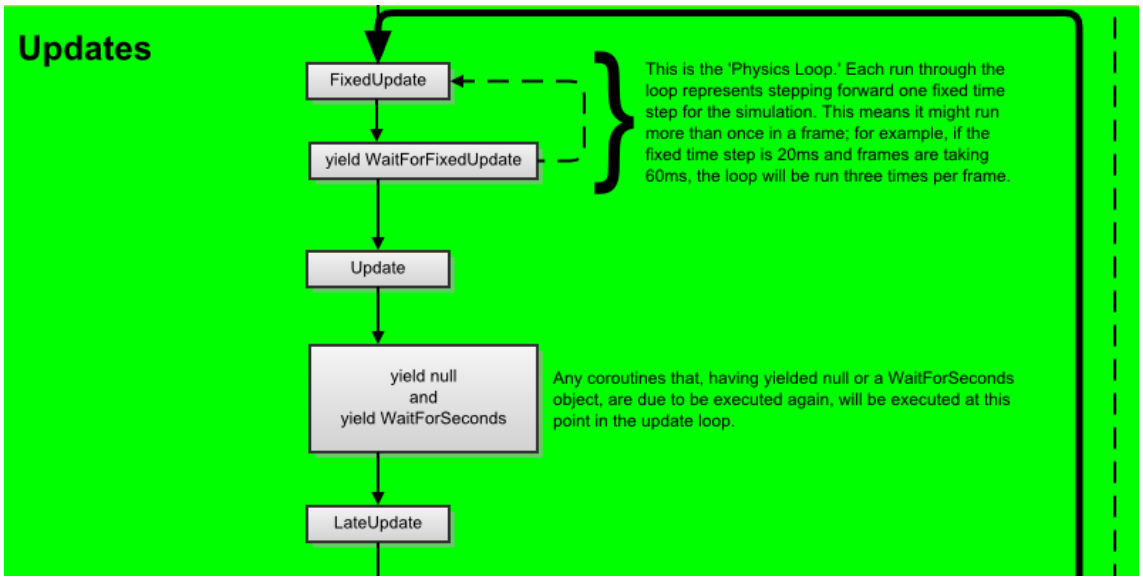
Rotation API

COMPONENT LIFECYCLE

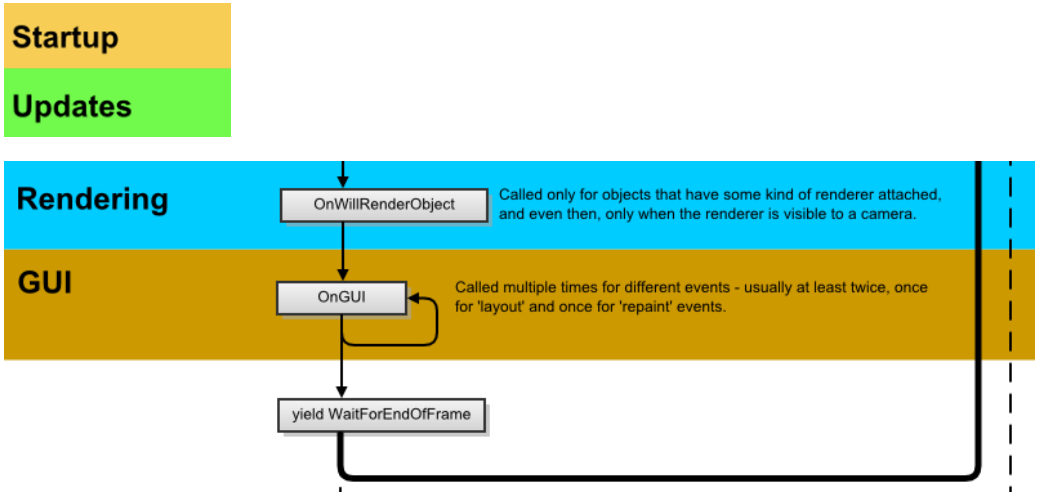


GAME OBJECT INITIALIZATION

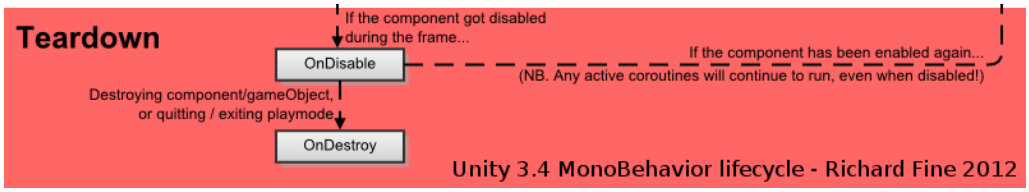
GAME OBJECT UPDATE



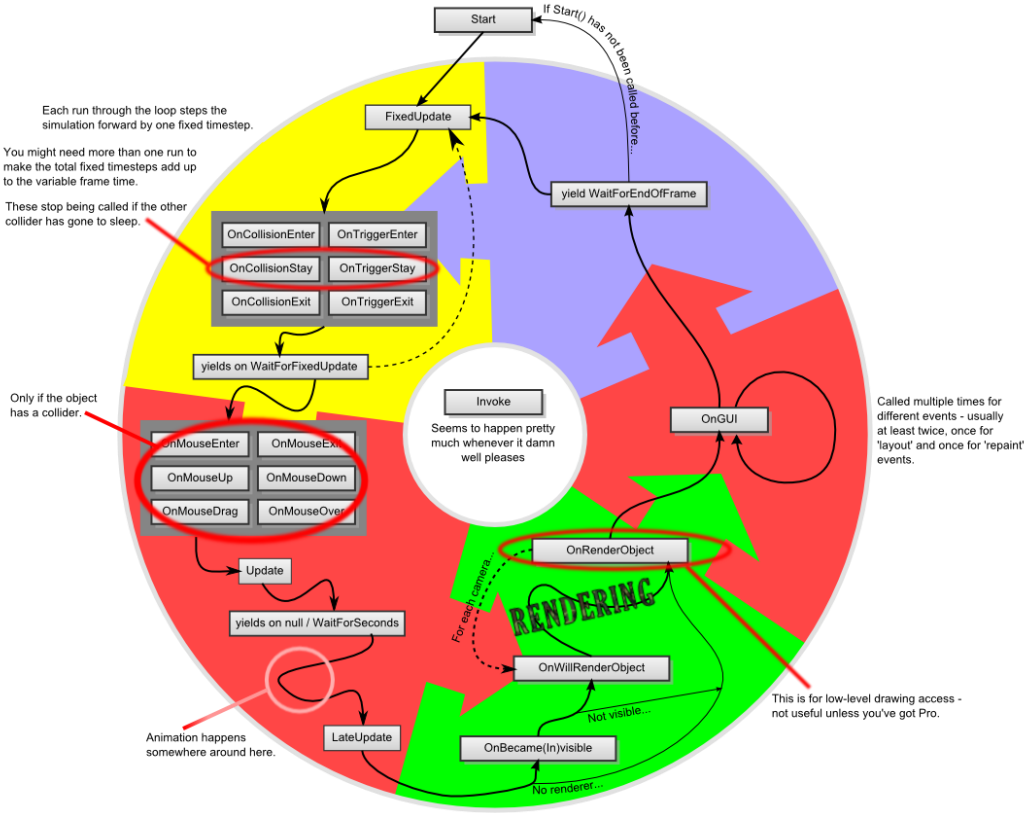
GAME OBJECT RENDERING



GAME OBJECT FINALIZATION

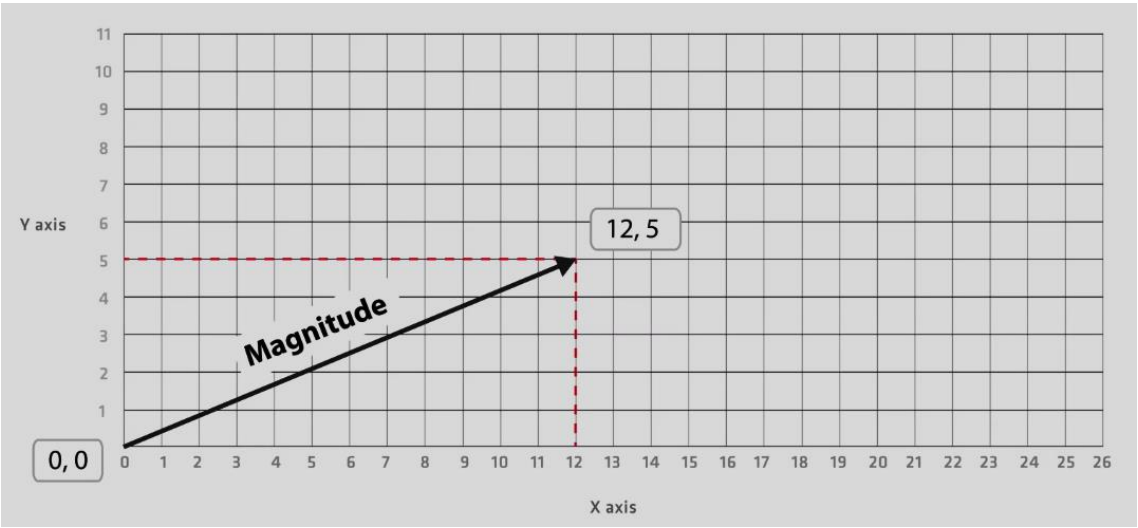


PHYSICS AND MOUSE EVENTS

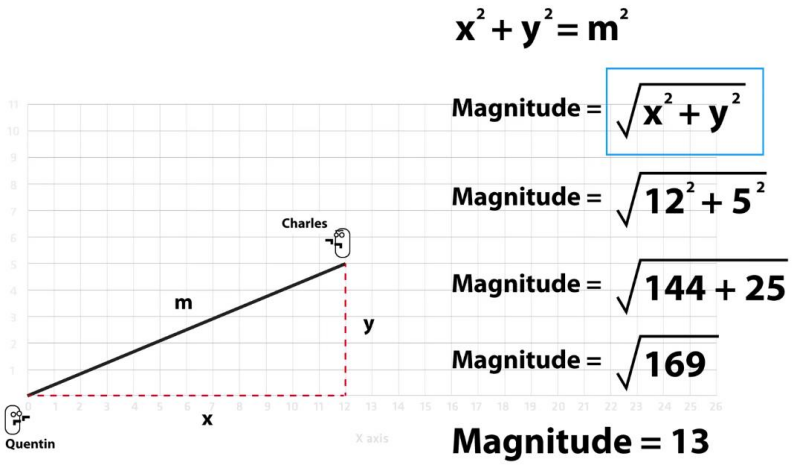


VECTOR & QUATERNION

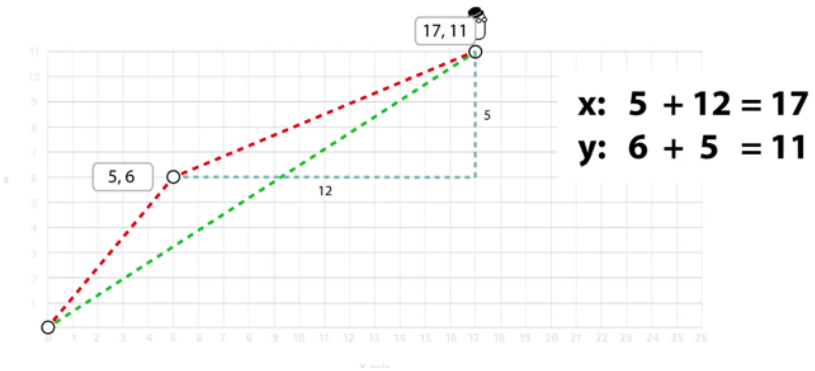
- 2D Vector



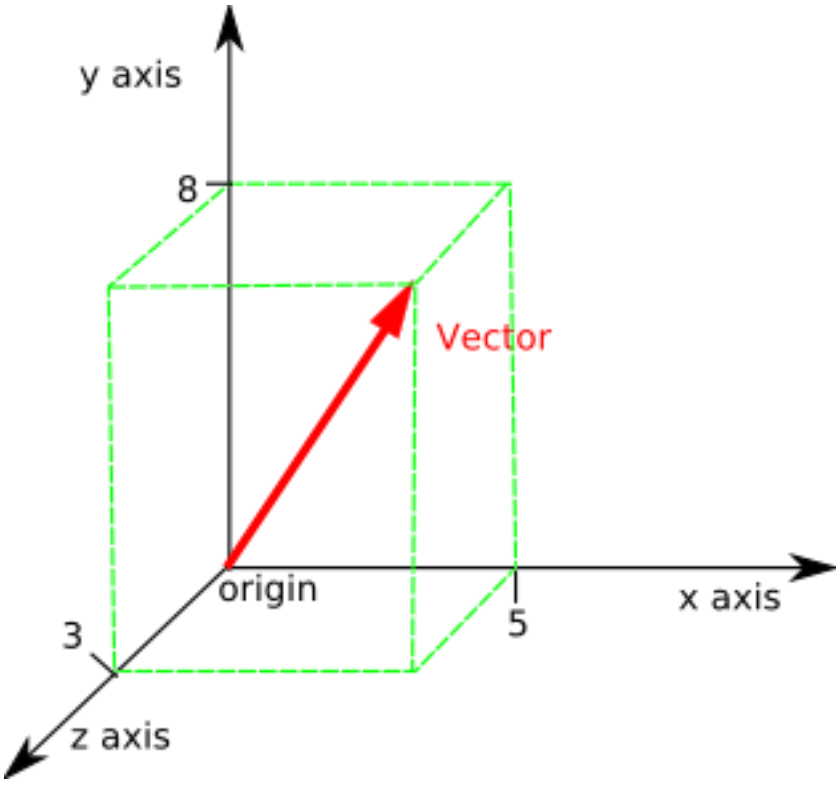
- Vector Magnitude



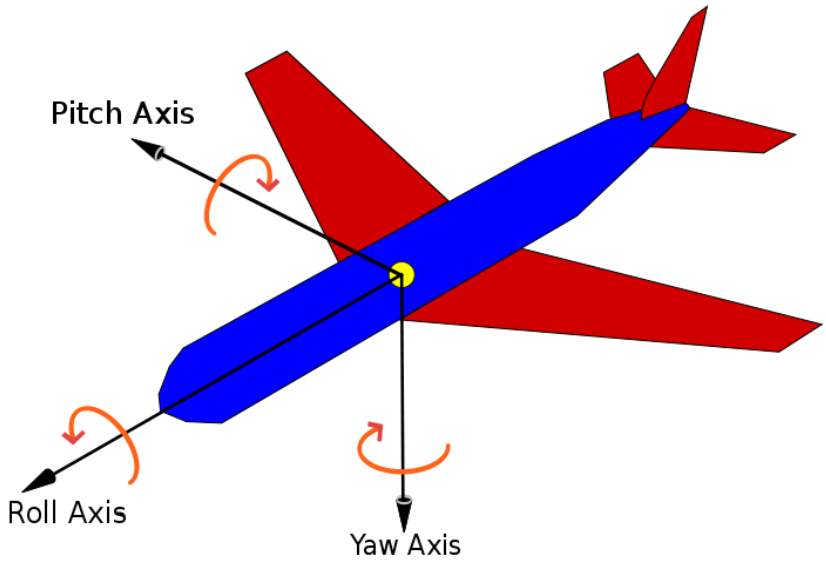
- Vector Addition



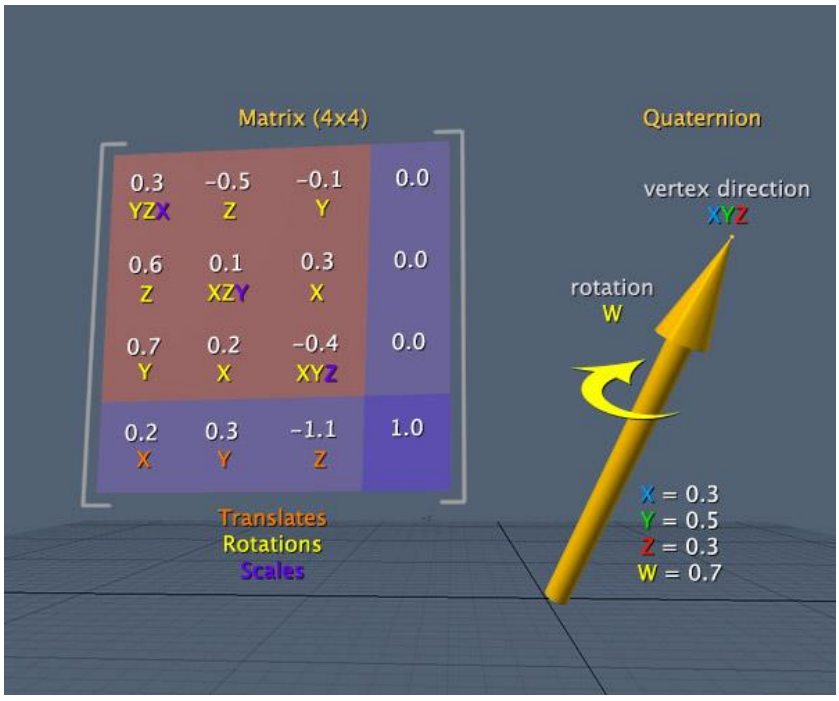
- 3D Vector



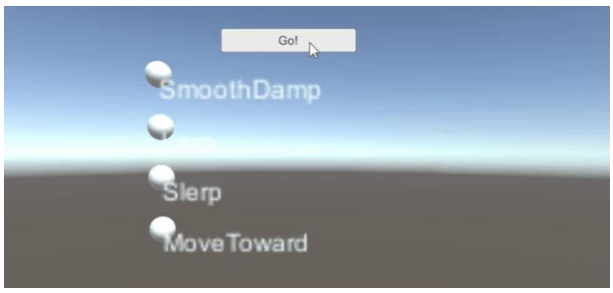
- Local Rotation



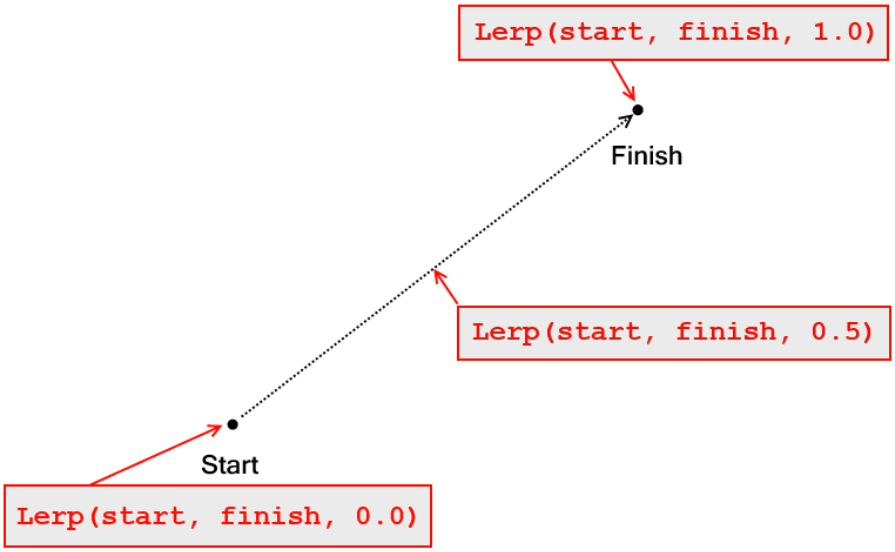
- Quarternion



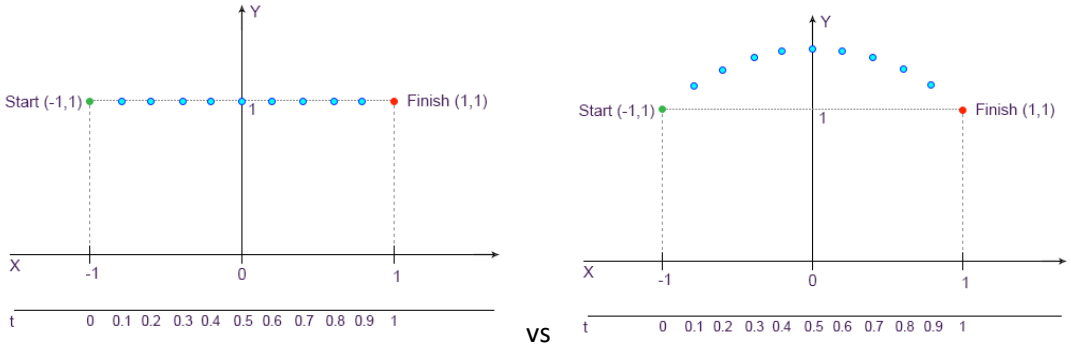
INTERPOLATE MOVEMENT



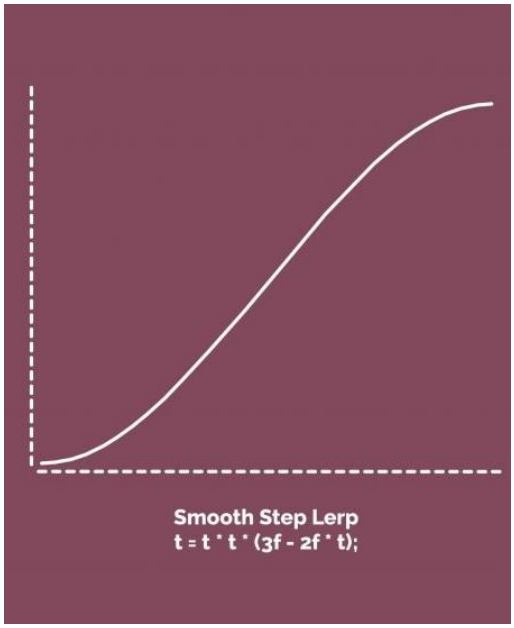
UNITY LERP



LERP VS SLERP



SMOOTH STEP




```
float t = time / duration;

t = t * t * (3f - 2f * t);

transform.position =
Vector3(startPosition, endPosition,
t);
```

SMOOTH DAMP

Vector3.SmoothDamp

public static Vector3 **SmoothDamp**(Vector3 **current**, Vector3 **target**, ref Vector3 **currentVelocity**, float **smoothTime**, float **maxSpeed** = Mathf.Infinity, float **deltaTime** = Time.deltaTime);

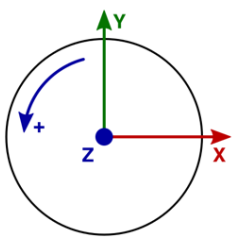
Parameters

current	The current position.
target	The position we are trying to reach.
currentVelocity	The current velocity, this value is modified by the function every time you call it.
smoothTime	Approximately the time it will take to reach the target. A smaller value will reach the target faster.
maxSpeed	Optionally allows you to clamp the maximum speed.
deltaTime	The time since the last call to this function. By default Time.deltaTime.

```
public class ExampleClass : MonoBehaviour {

    public Transform target;

    public float smoothTime = 0.3F;

    private Vector3 velocity = Vector3.zero;

    void Update()

    {

        // Define a target position above and behind the target transform

        Vector3 targetPosition = target.TransformPoint(new Vector3(0, 5,
-10));

        // Smoothly move the camera towards that target position

        transform.position = Vector3.SmoothDamp(transform.position,
targetPosition, ref velocity, smoothTime);

    }

}
```

2D ROTATION AXIS

```
// Update is called once per frame
void Update()
{
    var dir = Input.mousePosition - Camera.main.WorldToScreenPoint(transform.position);
    var angle = Mathf.Atan2(dir.y, dir.x) * Mathf.Rad2Deg + angleOffset;
    transform.rotation = Quaternion.AngleAxis(angle, Vector3.forward);
}
```



QUATERNION.EULER

public static Quaternion **Euler**(float **x**, float **y**, float **z**);

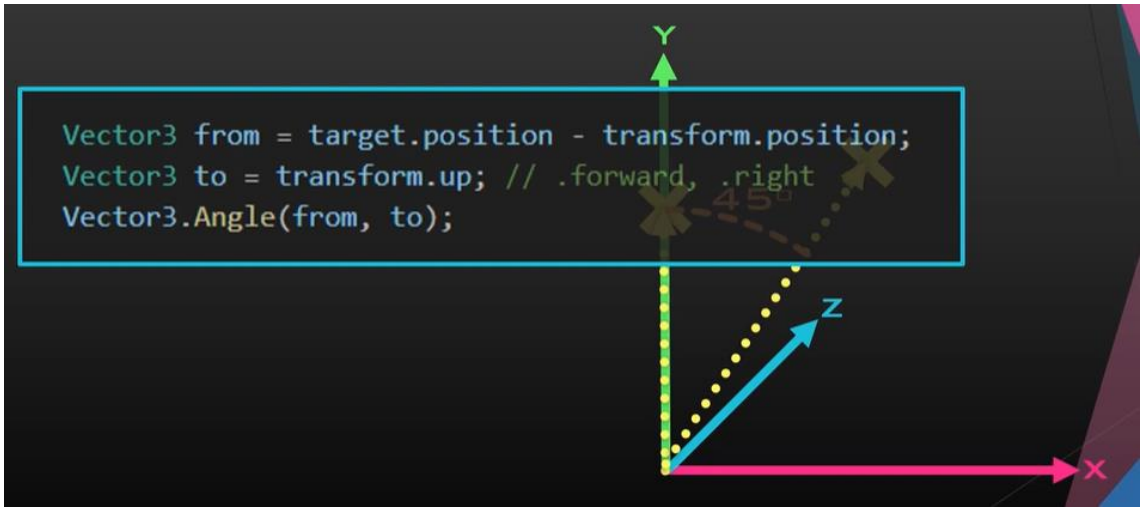
Description

Returns a rotation that rotates z degrees around the z axis, x degrees around the x axis, and y degrees around the y axis; applied in that order.

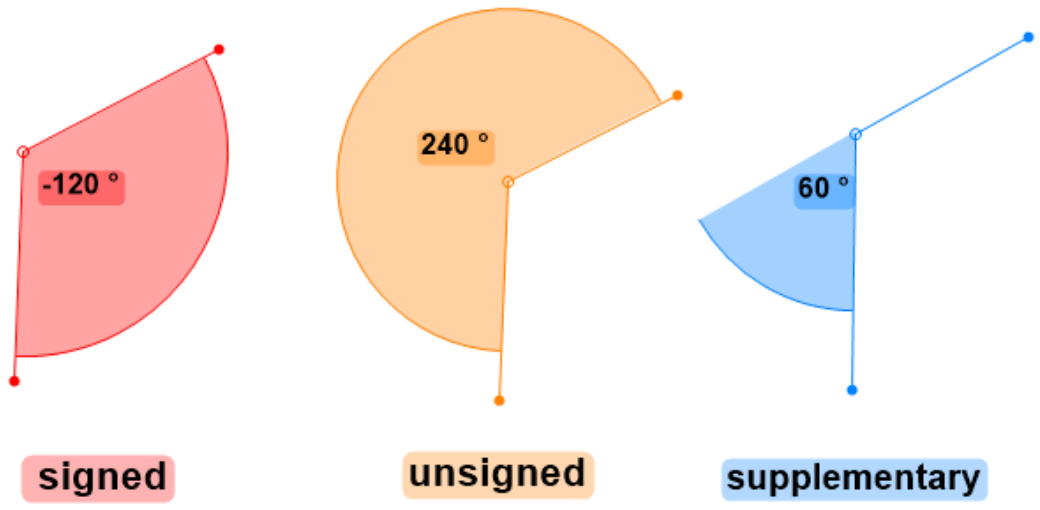
For more information, see Rotation and Orientation in Unity.

```
using UnityEngine;

public class Example : MonoBehaviour
{
    void Start()
    {
        // A rotation 30 degrees around the y-axis
        Quaternion rotation = Quaternion.Euler(0, 30, 0);
    }
}
```

VECTOR3.ANGLE



SIGNED ANGLE



VECTOR3.SIGNEDANGLE

public static float **SignedAngle**(Vector3 **from**, Vector3 **to**, Vector3 **axis**);

Parameters

from	The vector from which the angular difference is measured.
to	The vector to which the angular difference is measured.
axis	A vector around which the other vectors are rotated.

Description

Returns the signed angle in degrees between `from` and `to`.

The smaller of the two possible angles between the two vectors is returned, therefore the result will never be greater than 180 degrees or smaller than -180 degrees.

DAY 7: DATA STORING

Passing Data

PlayerPrefs

Json Format

Scriptable Object

File

PASSING DATA BETWEEN DIFFERENT SCENES

There are many ways to pass data from one scene to another inside Unity.

- Using Static Keywords
- Using DontDestroyOnLoad function
- Saving to local storage(file) and load inside another scene.
- Using ScriptableObject
- Using PlayerPrefs

PLAYERPREFS

- “PlayerPrefs” is a class that stores Player preferences between game sessions. It can store string, float and integer values into the user’s platform registry.
- Unity stores “PlayerPrefs” data differently based on which operating system the application runs on. In the file paths given on this page, the company name and product name are the names you set in Unity’s Player Settings.

PLAYERPREFS LOCATION

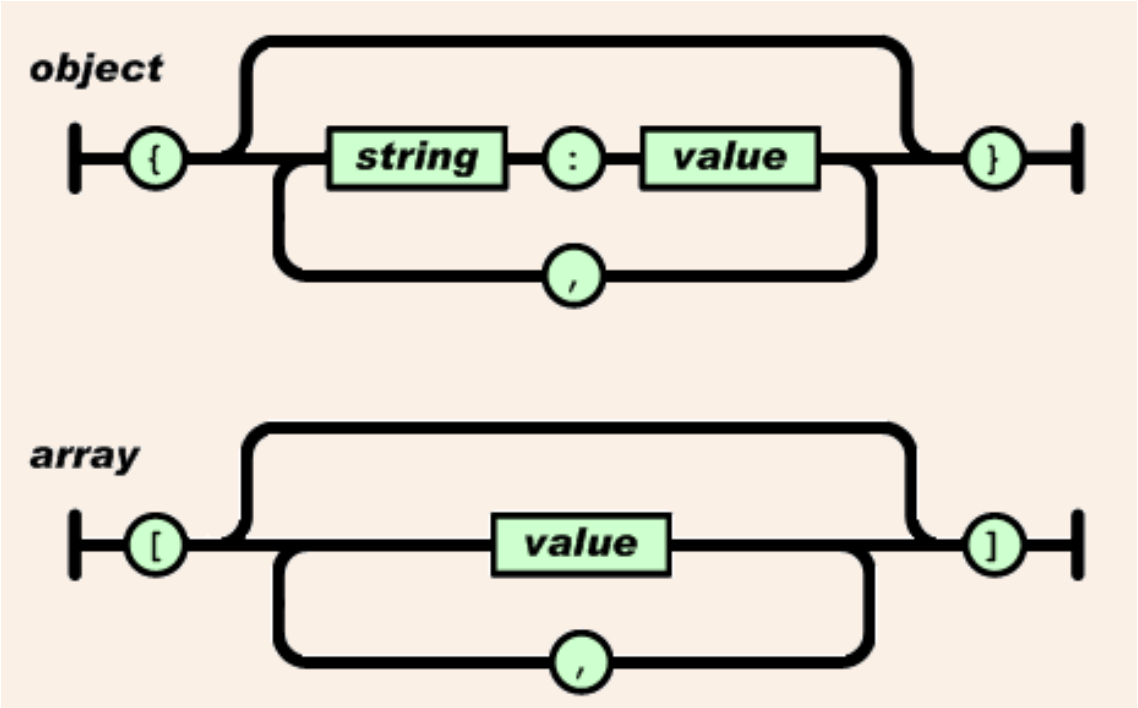
- On macOS, PlayerPrefs are stored in ~/Library/Preferences/com.ExampleCompanyName.ExampleProductName.plist. Unity uses the same .plist file for projects in the Editor and standalone Players.
- On Windows, PlayerPrefs are stored in HKCU\Software\ExampleCompanyName\ExampleProductName key.
- On Android, PlayerPrefs are stored in /data/data/pkg-name/shared_prefs/pkg-name.v2.playerprefs.xml. Unity stores PlayerPrefs data on the device, in SharedPreferences.
- On WebGL, Unity stores PlayerPrefs data using the browser's IndexedDB API.

PLAYERPREFS EXAMPLE

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

4 references
public class PrefsControl : MonoBehaviour
{
    1 reference
    public void SetFloatPlayerPrefs(float value){
        PlayerPrefs.SetFloat("test_float", value);
    }

    2 references
    public float GetFloatPlayerPrefs(){
        return PlayerPrefs.GetFloat("test_float", 0);
    }
}
```

JSON FORMAT

- JSON (JavaScript Object Notation) is a lightweight data-interchange format.
- It is easy for humans to read and write, and easy for machines to parse and generate.
- A JSON object may contain:
    - { "key1": "value1", "key2": "value2", ... } - a hash with key/value pairs; the value can be another JSON object.
    - ["va1", "val2", "val3", ...] - an array with values; the value can be another JSON object.
    - "String" - a text string enclosed in double quotes.



JSON SERIALIZATION

```
[Serializable]
public class MyClass
{
    public int level;
    public float timeElapsed;
    public string playerName;
}
string json = JsonUtility.ToJson(myObject);
// json now contains: '{"level":1,"timeElapsed":47.5,"playerName":"Dr Charles Francis"}'
myObject = JsonUtility.FromJson<MyClass>(json);
JsonUtility.FromJsonOverwrite(json, myObject);
```

SCRIPTABLE OBJECT

- ScriptableObject is a data container that you can use to save large amounts of data, independent of class instance.
- Data that you save from Editor Tools to ScriptableObjects as an asset is written to disk and is therefore persistent between sessions.
- The main use cases for ScriptableObjects are:
    - Saving and storing data during an Editor session
    - Saving data as an Asset in your Project to use at run time

SCRIPTABLE OBJECT EXAMPLE

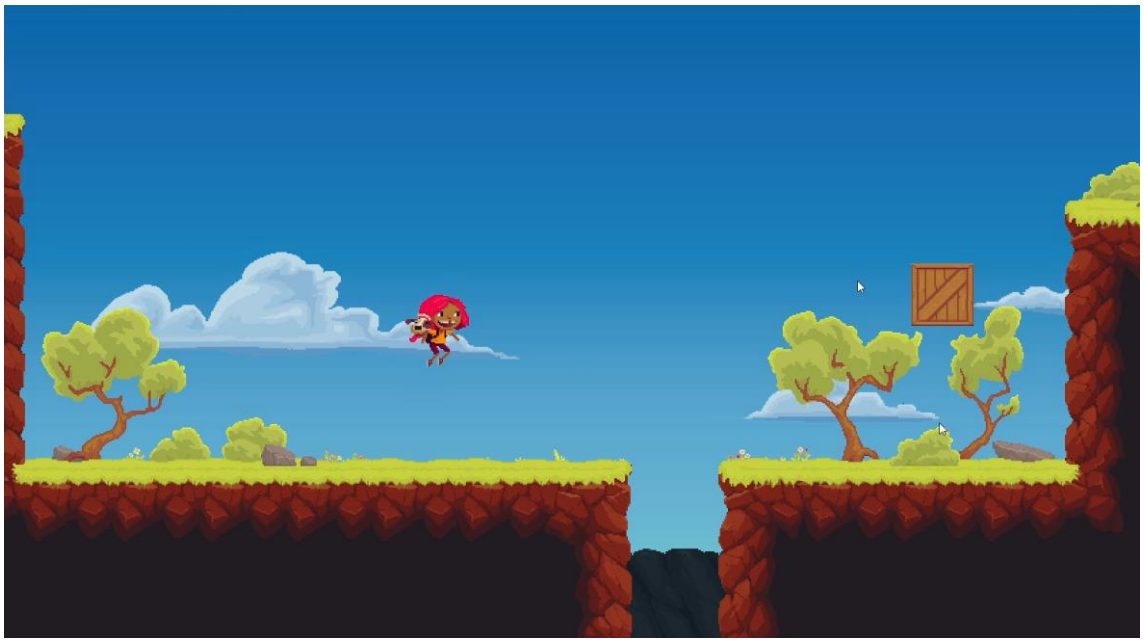
```
[CreateAssetMenu]
public class FloatVariable : ScriptableObject, ISerializationCallbackReceiver
{
    public float InitialValue;
    [NonSerialized]
    public float RuntimeValue;
    public void OnAfterDeserialize() {
        RuntimeValue = InitialValue;
    }
    public void OnBeforeSerialize() { }
}
```

STORING DATA USING FILE

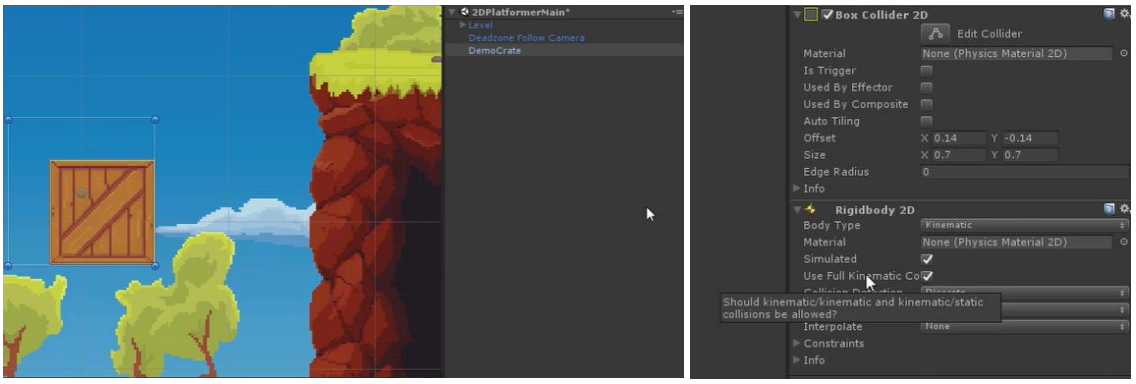
```
using System.IO;
string filePath = Application.persistentDataPath + "/settings.txt";
string[] scores;
File.WriteAllLines(filePath, scores);
string[] levelsInfo = File.ReadAllLines(fileName);
```

DAY 8: PLATFORMER GAME

OVERVIEW



RIGIDBODY 2D



KINEMATIC RIGIDBODY 2D

- Kinematic Rigidbody 2D is designed to be repositioned explicitly via Rigidbody2D.MovePosition or Rigidbody2D.MoveRotation.
- While a Dynamic Rigidbody 2D is affected by gravity and forces, a Kinematic Rigidbody 2D isn't.
- A Kinematic Rigidbody 2D does still move via its velocity, but the velocity is not affected by forces or gravity.
- A Kinematic Rigidbody 2D does not collide with other Kinematic Rigidbody 2Ds or with Static Rigidbody 2Ds; it only collides with Dynamic Rigidbody 2Ds.

FULL KINEMATIC CONTACTS

- Enable this setting if you want the Kinematic Rigidbody 2D to collide with all Rigidbody 2D Body Types.
- This is like a Dynamic Rigidbody 2D, except the Kinematic Rigidbody 2D is not moved by the physics engine when contacting another Rigidbody 2D component.
- When Use Full Kinematic Contacts is disabled, the Kinematic Rigidbody 2D only collides with Dynamic Rigidbody 2Ds.

STATIC RIGIDBODY

- There are two ways to mark a Rigidbody 2D as Static:
- For the GameObject with the Collider 2D component not to have a Rigidbody 2D component at all. All such Collider 2Ds are internally considered to be attached to a single hidden Static Rigidbody 2D component.
- For the GameObject to have a Rigidbody 2D and for that Rigidbody 2D to be set to Static.
- A Static body only collides with Dynamic Rigidbody 2Ds. Static Rigidbody 2Ds are designed not to move, and collisions between two Static Rigidbody 2D objects that intersect are not registered.
- However, Static Rigidbody 2Ds and Kinematic Rigidbody 2Ds will interact with each other if one of their Collider 2Ds is set to be a trigger.

RIGIDBODY CONSTRAINTS

- Define any restrictions on the Rigidbody 2D's motion.
    - Freeze Position: Stops the Rigidbody 2D moving in the world's x & y axes selectively.
    - Freeze Rotation: Stops the Rigidbody 2D rotating around the world's z axis selectively.

MOVING

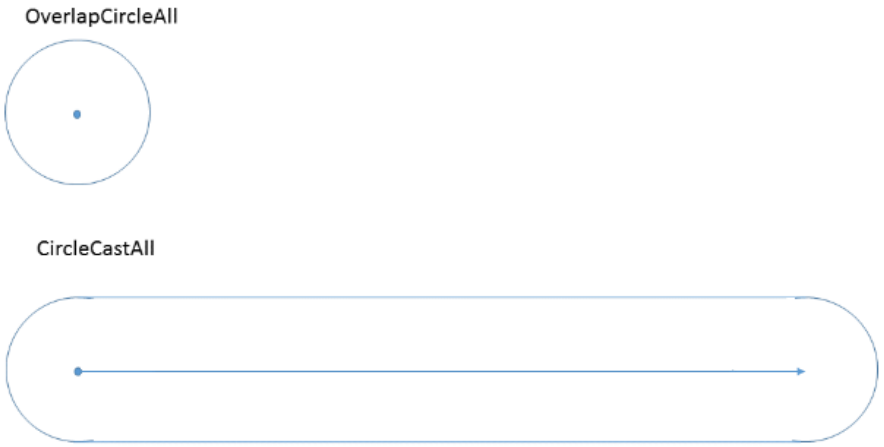
```
public void Move(float move, bool crouch, bool jump)
{
```

```
// Move the character by finding the target velocity
Vector3 targetVelocity = new Vector2(move * 10f, m_Rigidbody2D.velocity.y);
// And then smoothing it out and applying it to the character
m_Rigidbody2D.velocity = Vector3.SmoothDamp(m_Rigidbody2D.velocity, targetVelocity, ref m_Velocity, m_MovementSmoothing);
```

FACING

```
// If the input is moving the player right and the player is facing left...
if (move > 0 && !m_FacingRight)
{
        // ... flip the player.
        Flip();
}
// Otherwise if the input is moving the player left and the player is facing right...
else if (move < 0 && m_FacingRight)
{
        // ... flip the player.
        Flip();
}

private void Flip()
{
        // Switch the way the player is labelled as facing.
        m_FacingRight = !m_FacingRight;

        // Multiply the player's x local scale by -1.
        Vector3 theScale = transform.localScale;
        theScale.x *= -1;
        transform.localScale = theScale;
}
```

CASTING VS OVERLAPPING



CHECK IF ON GROUND

```
[SerializeField] private LayerMask m_WhatIsGround;
[SerializeField] private Transform m_GroundCheck;

 private void FixedUpdate()
 {
        bool wasGrounded = m_Grounded;
        m_Grounded = false;

        // The player is grounded if a circlecast to the groundcheck position hits anything designated as ground
        // This can be done using layers instead but Sample Assets will not overwrite your project settings.
        Collider2D[] colliders = Physics2D.OverlapCircleAll(m_GroundCheck.position, k_GroundedRadius, m_WhatIsGround);
        for (int i = 0; i < colliders.Length; i++)
        {
                if (colliders[i].gameObject != gameObject)
                {
                        m_Grounded = true;
                        if (!wasGrounded)
                                OnLandEvent.Invoke();
                }
        }
 }
```

JUMPING

```
// If the player should jump...
if (m_Grounded && jump)
{
        // Add a vertical force to the player.
        m_Grounded = false;
        m_Rigidbody2D.AddForce(new Vector2(0f, m_JumpForce));
}
```

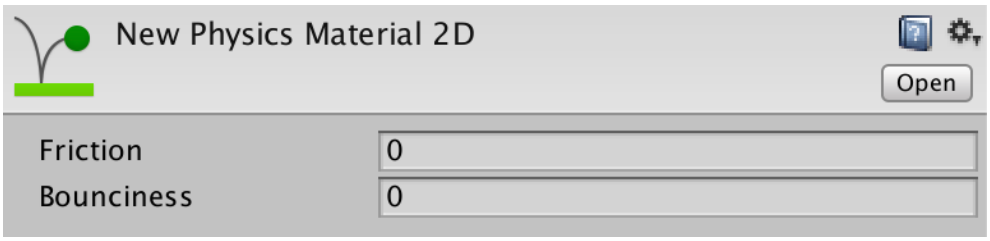
PREVENT DOUBLE ACCELERATE WHEN JUMP

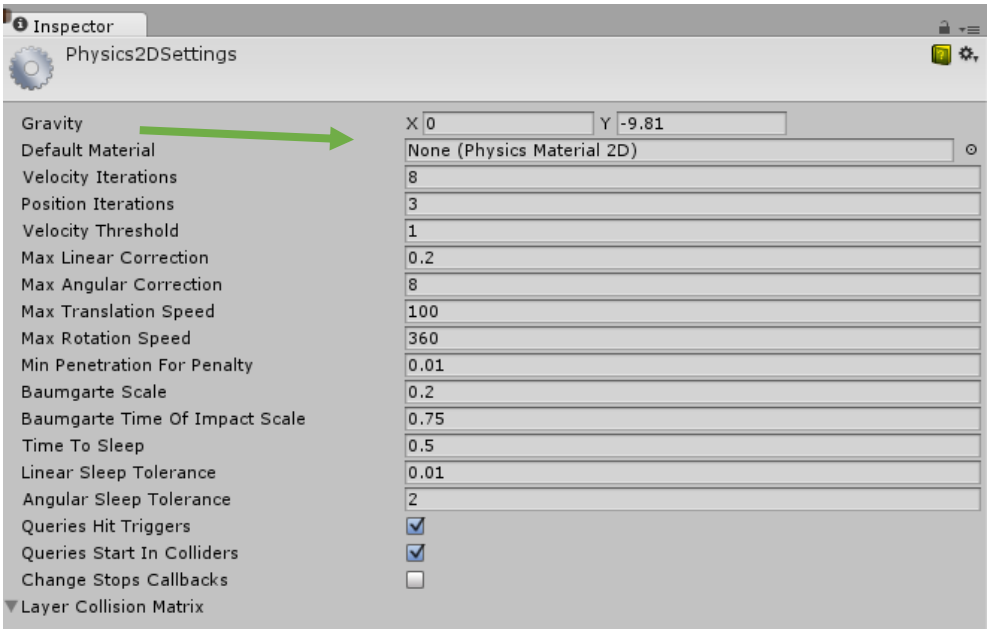
- Try the following solutions
- Implement jumping cool down
- Check if velocity.y > 0 (works only if no climbing)
- Use GetKeyDown

ELIMINATE FRICTION

- Try the following solutions
- Use PhysicsMaterial2d to reduce friction
- Use round-shaped collider on the foot

- Change the default physics material





REFERENCES

https://www.youtube.com/watch?v=dwcT-Dch0bA

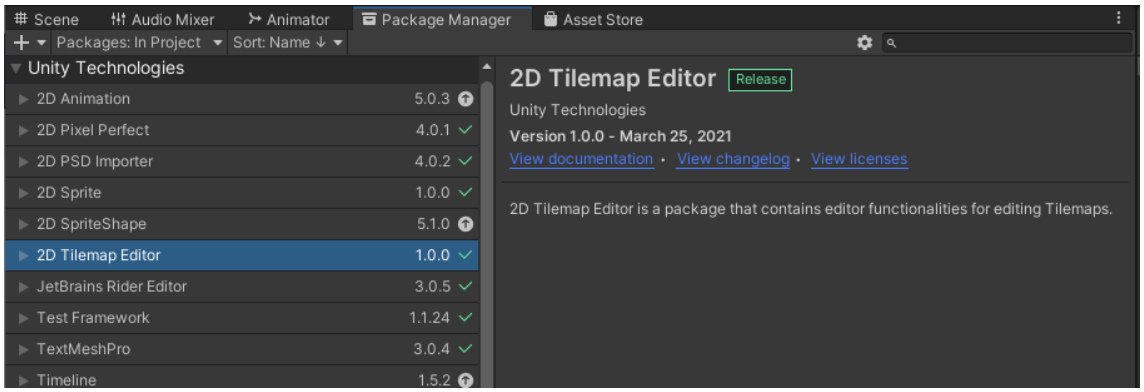
https://roystan.net/articles/character-controller-2d.html

https://learn.unity.com/tutorial/live-session-2d-platformer-character-controller

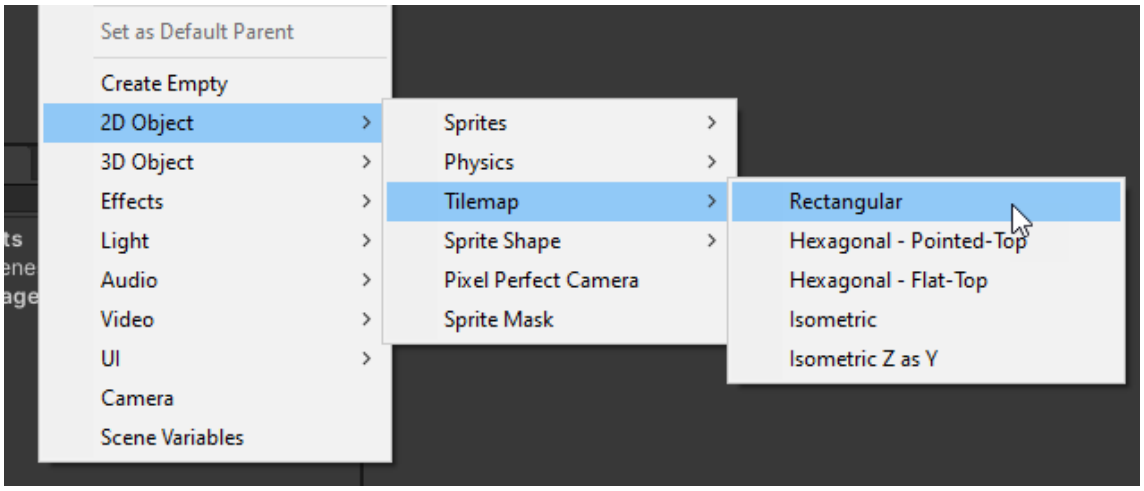
https://www.youtube.com/watch?v=2MImqNvgrNg

DAY 9: TILE MAP

INSTALL VIA PACKAGE MANAGER

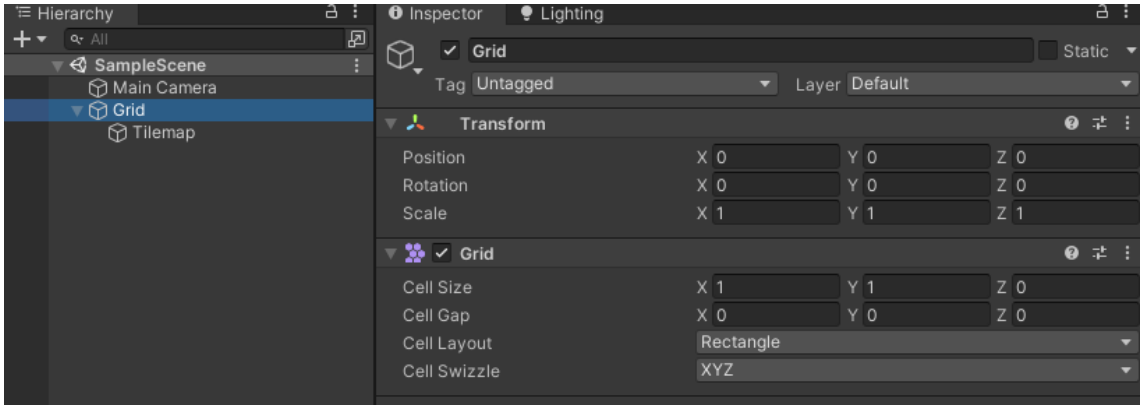


CREATE A TILE MAP



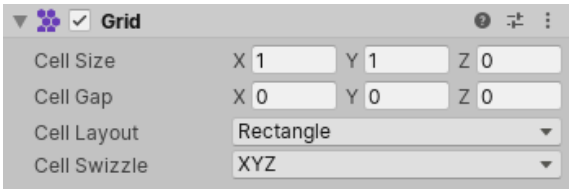
TILEMAP SYSTEM

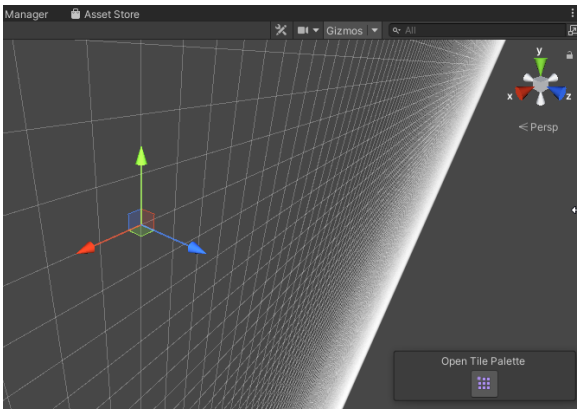
- Tilemap consists of two things: a grid and the actual Tilemap. There's only one grid in the Scene but there can be several Tilemaps.
- The grid creates the layout for the Scene that the Tilemaps rely on.



GRID COMPONENT

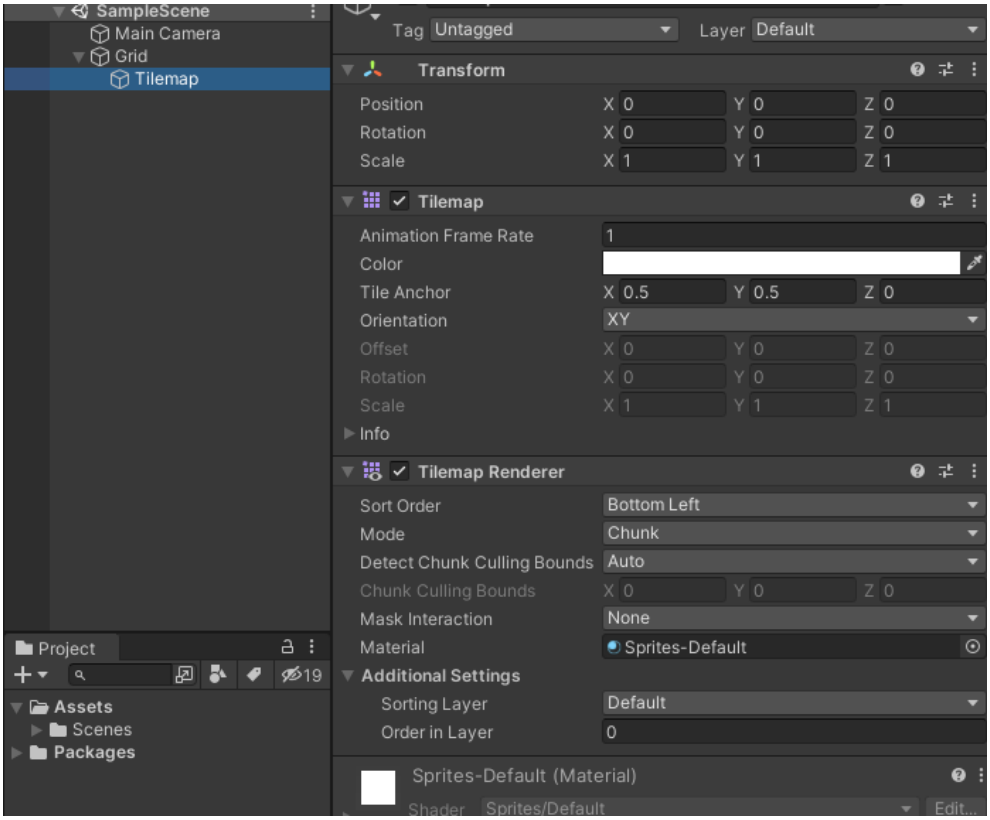
- Cell Size: size of each square in the grid
- Cell Gap: space between each square on the grid.
- Cell Layout: layout of the tiles on the grid (Rectangle, Hexagon, Isometric…)
- Cell Swizzle: which direction the grid is facing (XYZ, XZY, YXZ, ZXY and ZYX)





TILEMAP GAME OBJECT

- Select the Tilemap underneath the grid, it will display two components:
- Tilemap
- Tilemap Renderer



TILEMAP COMPONENT

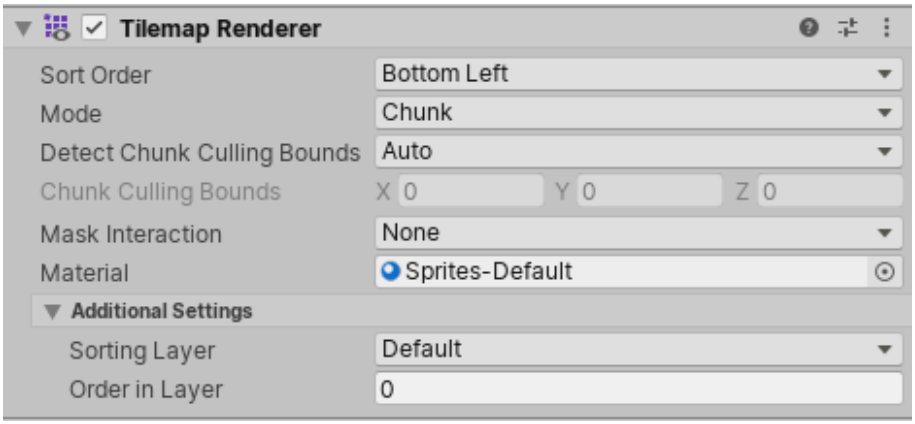
- The Tilemap component is a system which stores and handles Tile Assets for creating 2D levels. It transfers the required information from the Tiles placed on it to other related components such as the Tilemap Renderer and the Tilemap Collider 2D
- When you create a Tilemap, the Grid component is automatically parented to the Tilemap and acts as a guide when you lay out Tiles onto the Tilemap.
- To create, modify, and pick the Tiles for painting onto a Tilemap, use the Tile Palette (menu: Window > 2D > Tile Palette) and its tools.

TILEMAP PROPERTIES

- Animation Frame Rate: affects the speed of any animated tiles in Tilemap
- Color: color of the tiles
- Tile Anchor: where each tile will be anchored in the grid (default: center).
- Orientation: control the direction the tiles will be facing (in 3D only).

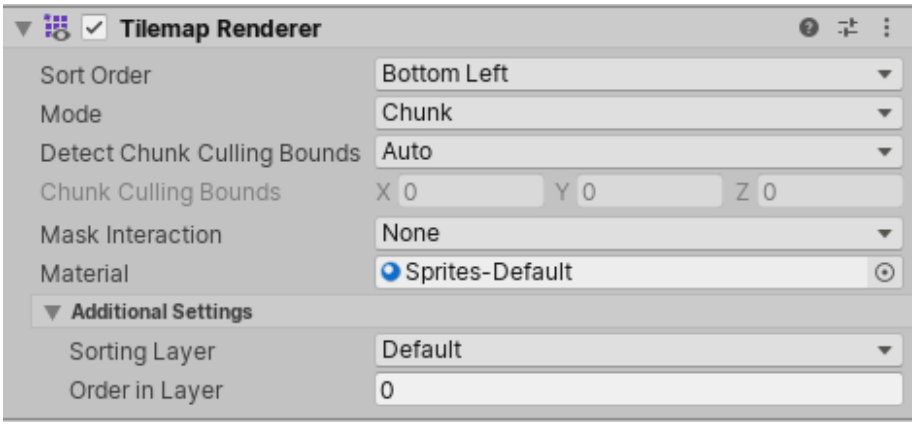
TILEMAP RENDERER

- Sort Order: Which tiles are rendered first
- Mode: Render mode of the tiles
    - Chunk Mode: sprites on a Tilemap are rendered in batches (for performance).
    - Individual Mode: sprites are sorted based on their position in the Tilemap and the Sort Order, and rendered in a way that allows for interweaving of Sprites. This allow character to go behind other Sprites, such as between trees in a forest.



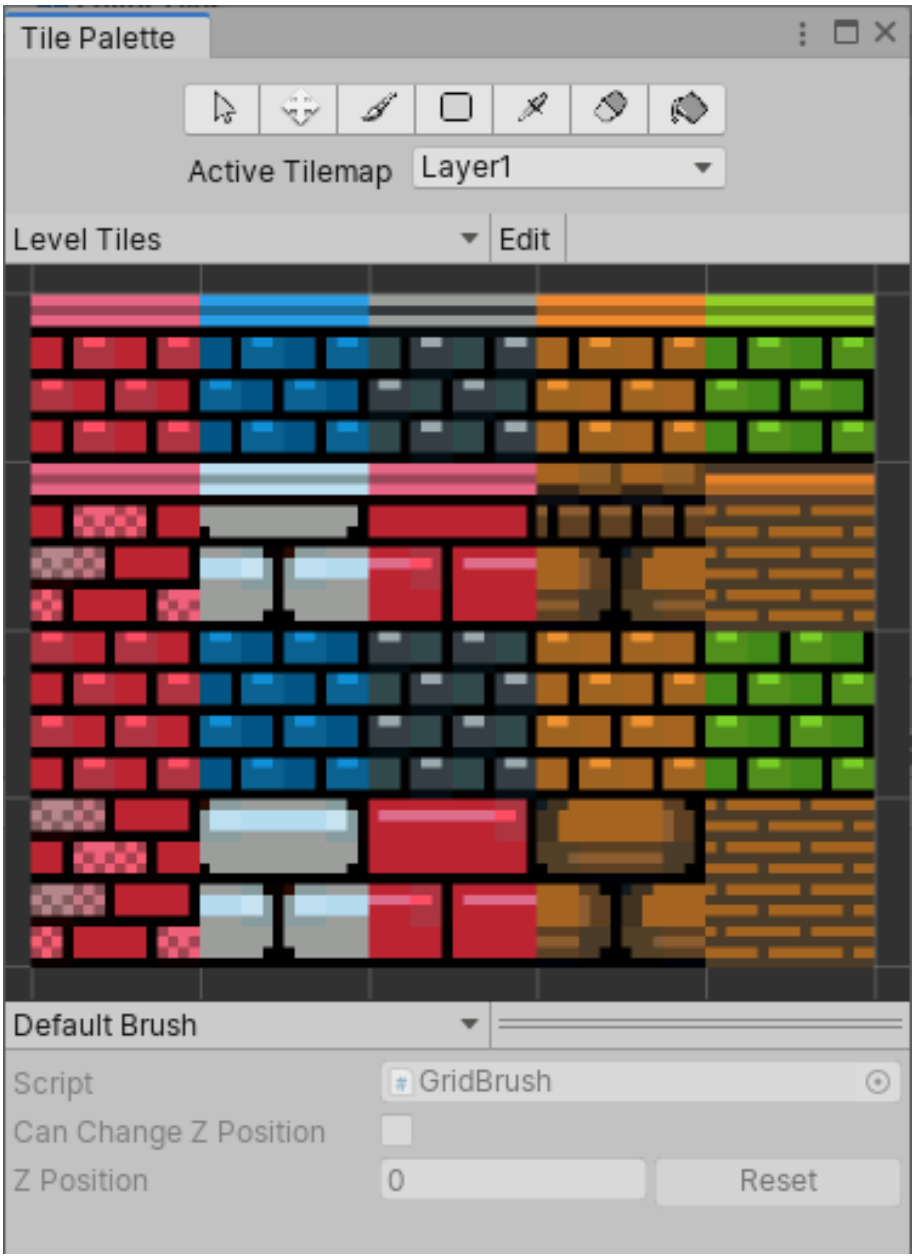
TILEMAP RENDERER

- Detect Chunk Culling: can detect bounds automatically, or bounds can be manually set in the next options.
- Chunk Cull Bounds: allows for extension of bounds for culling in Chunk Mode
- Mask Interaction: optionally be visible only inside or outside a Sprite Mask



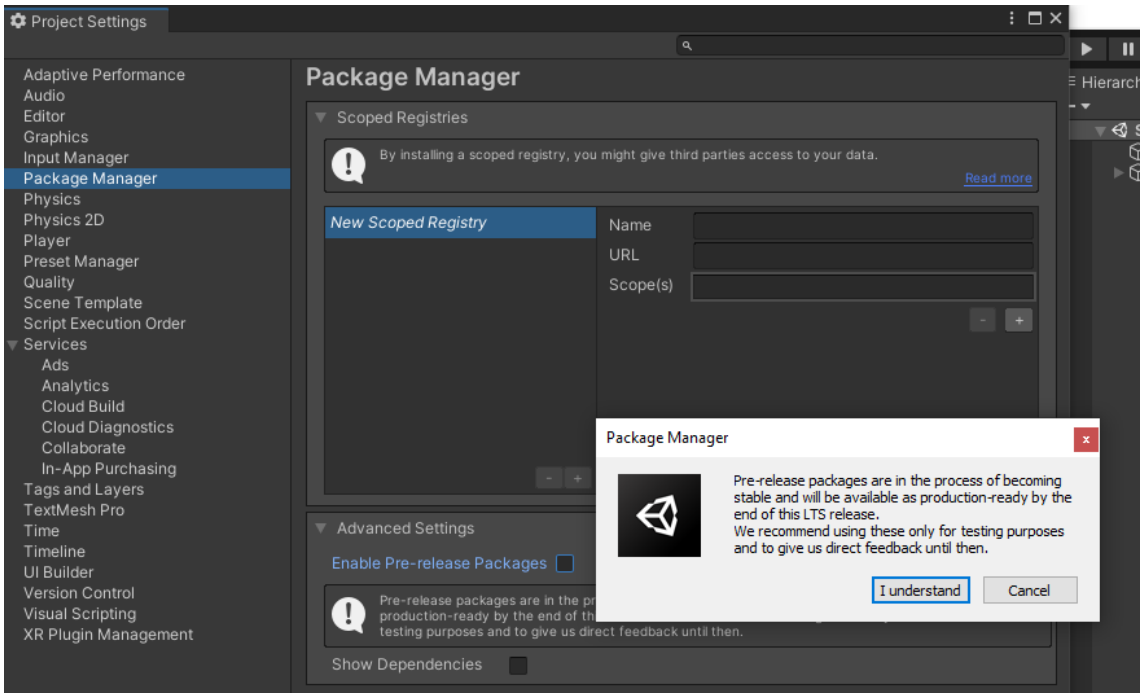
TILE PALLETE

- Tile Palette tools assists creating, modifying or editing Tilemaps
- Open the Tile Palette window by going to **Window > Tile Palette** in your project.

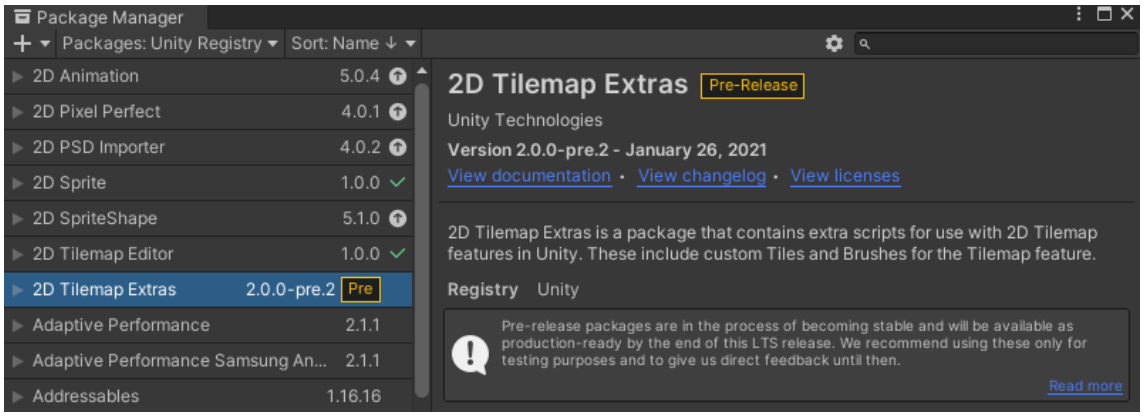

Tile Palette
Active Tilemap
Layer1
Level Tiles
Edit
Default Brush
Script
GridBrush
Can Change Z Position
Z Position
0
Reset

DAY 10: ADVANCED TILE MAP

ENABLE PREVIEW PACKAGE

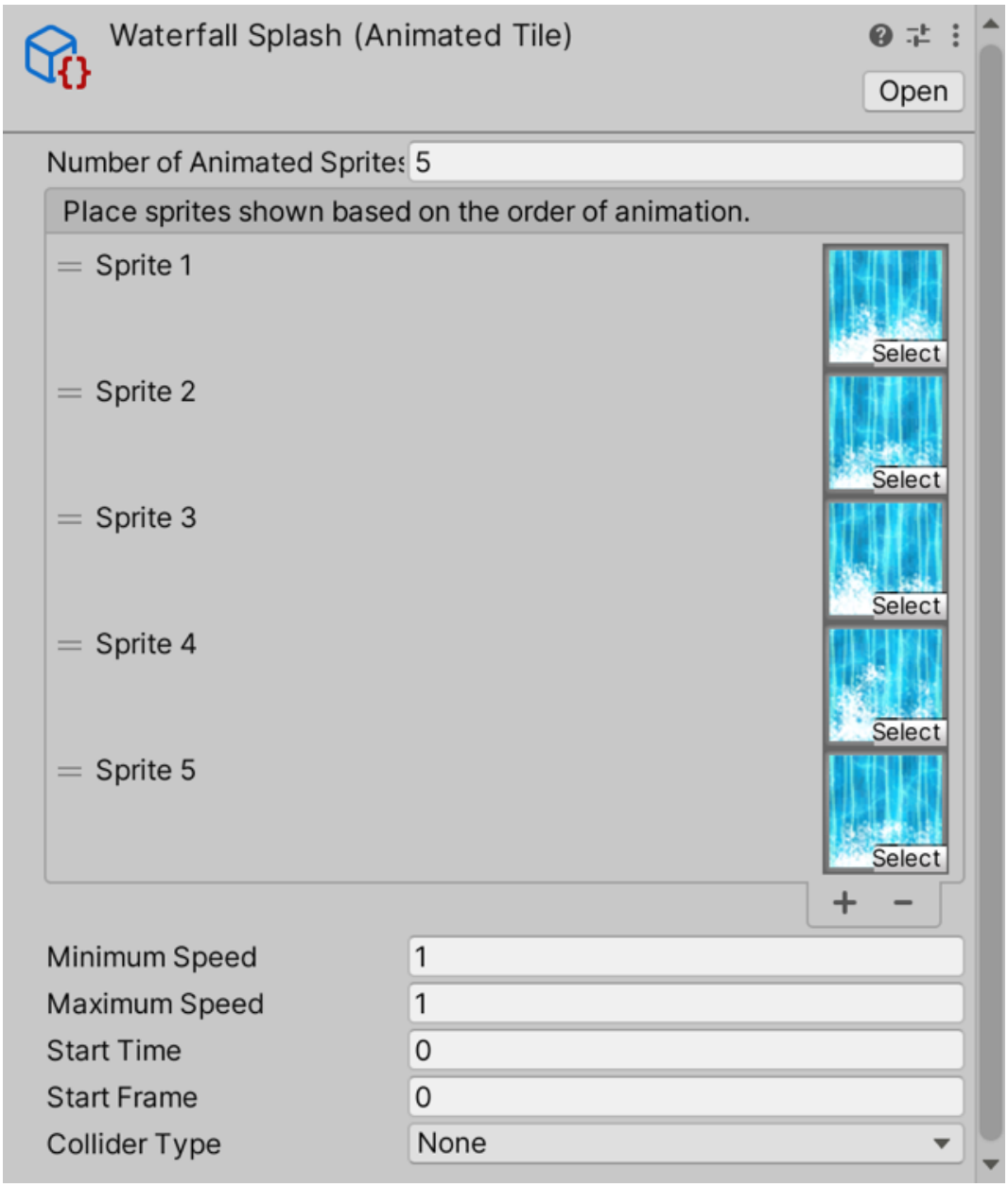


INSTALL 2D TILEMAP EXTRAS



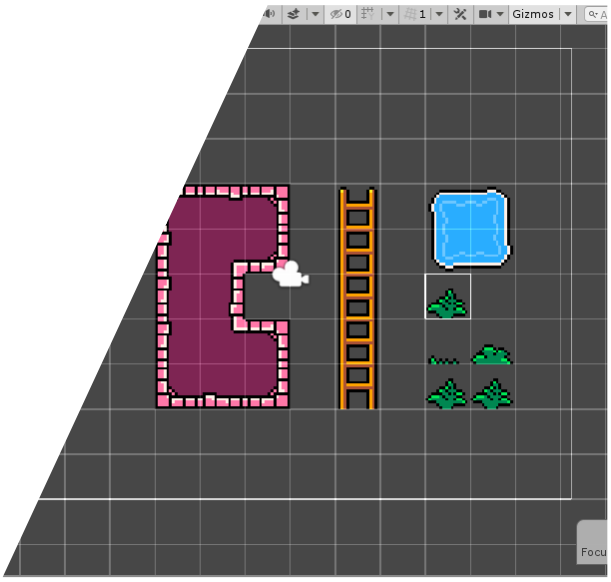
ANIMATED TILE

- An Animated Tile runs through and displays a list of Sprites in sequence to create a frame-by-frame animation.

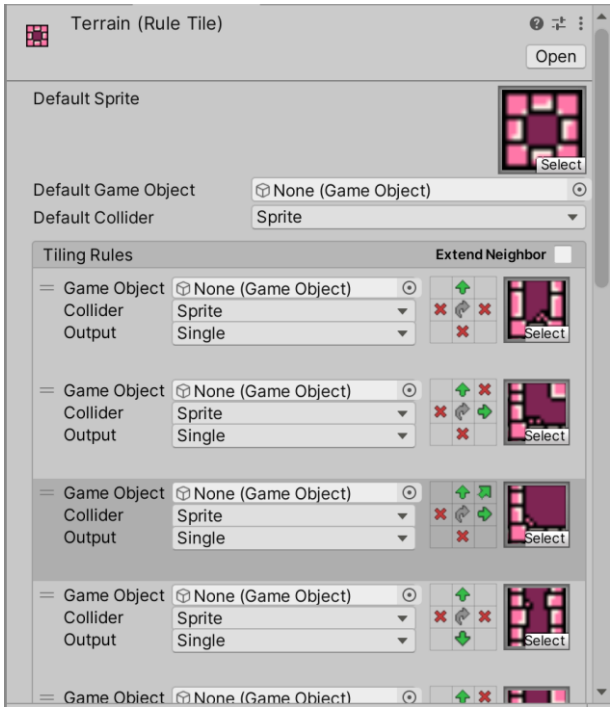


RULE TILE

- This is a generic visual Tile that other Tiles such as the Terrain Tiles, Pipeline Tile, Random Tile or Animated Tiles are based on.
- There are specific types of Rule Tiles for each of the Tilemap grid types.
- The default Rule Tile is for the default Rectangle Grid type;
- The Hexagonal Rule Tile is for the Hexagonal Grid type;
- And the Isometric Rule Tile is for the Isometric Grid types. The different types of Rule Tiles all possess the same properties.



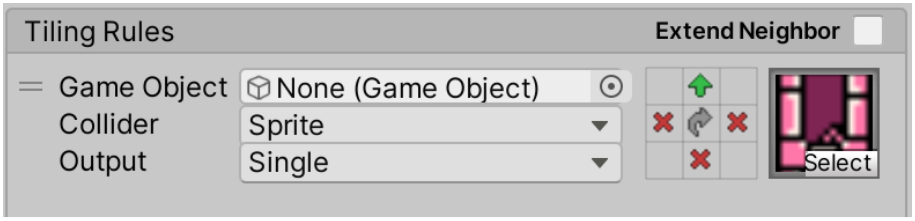
RULE TILE PROPERTIES



TILING RULES

- The 3x3 box represents the neighbors a Tile can have, where the center represents the Tile itself, and the eight bordering cells are its neighboring Tiles in their relative positions to the Tile.
- Each of the neighboring cells can be set with one of three options:
- Don't Care: ignores the contents in this cell

- This: checks if the contents of this cell is an instance of this Rule Tile. If it is an instance, the rule passes. If it is not an instance, the rule fails.
- Not This: The Rule Tile checks if the contents of this cell is not an instance of this Rule Tile. If it is not an instance, the rule passes. If it is an instance, the rule fails.

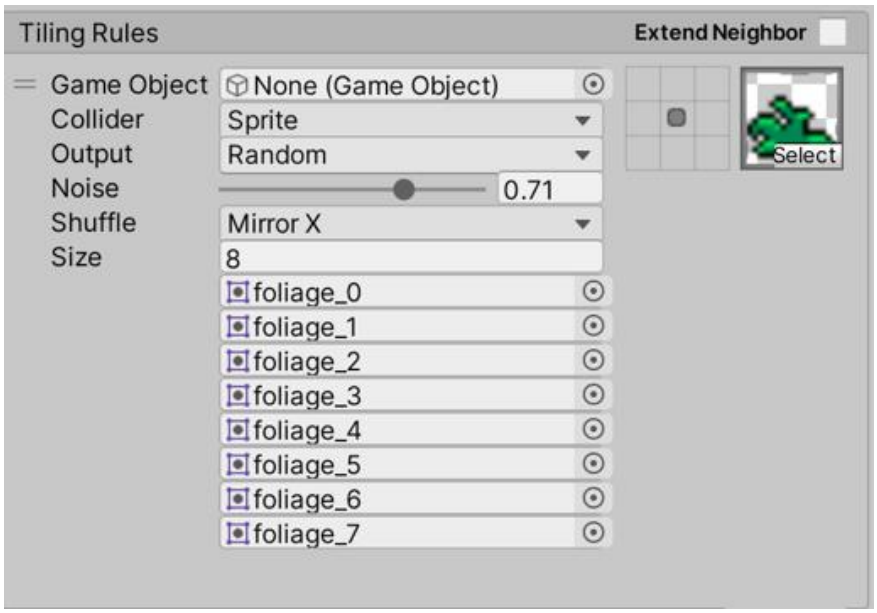


TILING RULES CENTER

- When the Rule is set to Fixed, the Rule will only match exactly the conditions set for its neighbors.
- When the Rule is set to 'Rotated', the 3x3 box will be rotated 90 degrees each time the Rule fails to match and it will try to match again with this rotated 3x3 box.
- When the Rule is set to Mirror X, Mirror Y or Mirror XY, the 3x3 box will be mirrored in that axis each time the Rule fails to match and it will try to match again with this mirrored 3x3 box.

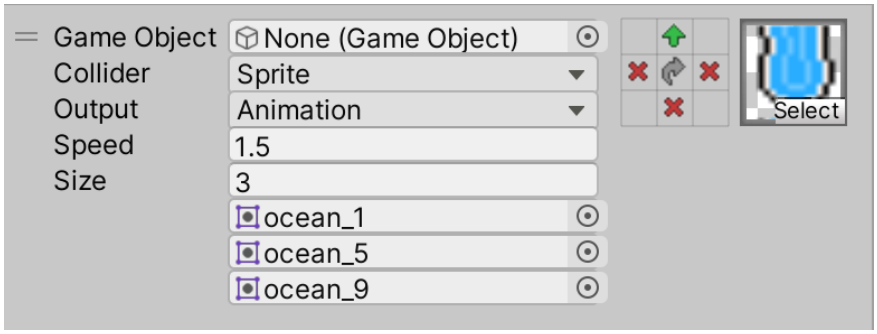
RANDOM OUTPUT

- If you want the Rule Tile to have a Random output, you can set the Output to Random.
- This will allow you to specify a number of input Sprites to randomize from. The rotation of the Sprites can be randomized as well by changing the Shuffle property.



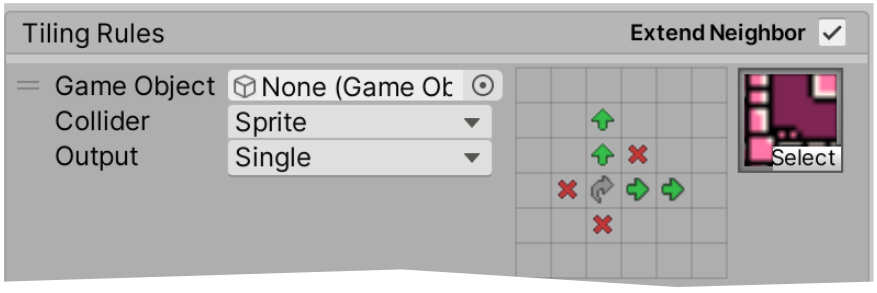
ANIMATION OUTPUT

- If you want the Rule Tile to output a Sprite Animation, you can set the Output to Animation.
- This will allow you to specify a number of Sprites to animate sequentially.
- The speed of the Animation can be randomized as well by changing the Speed property.



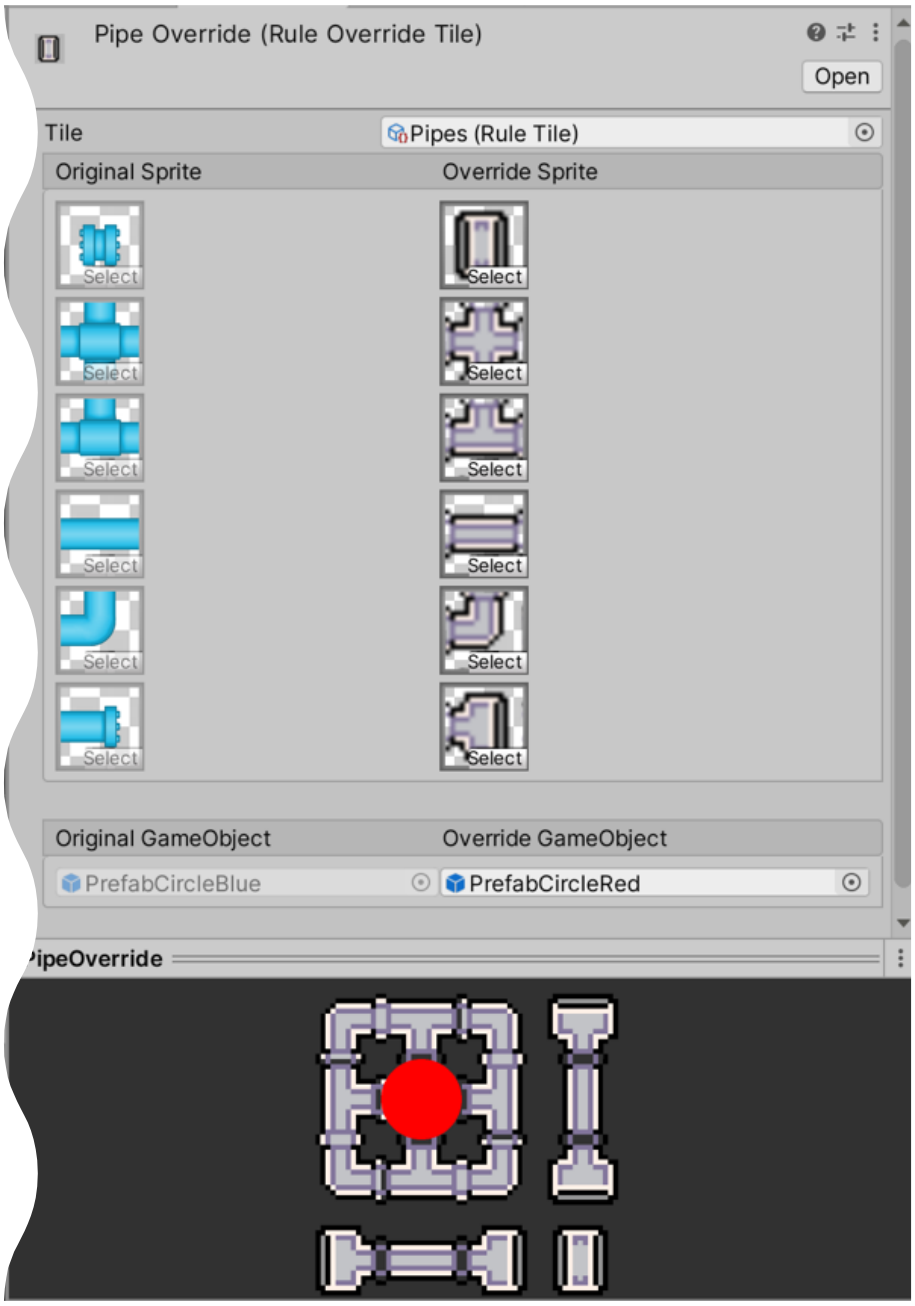
EXTEND NEIGHBORS

- When Extend Neighbors is enabled, the 3x3 box can be extended to allow for more specific neighbor matching.



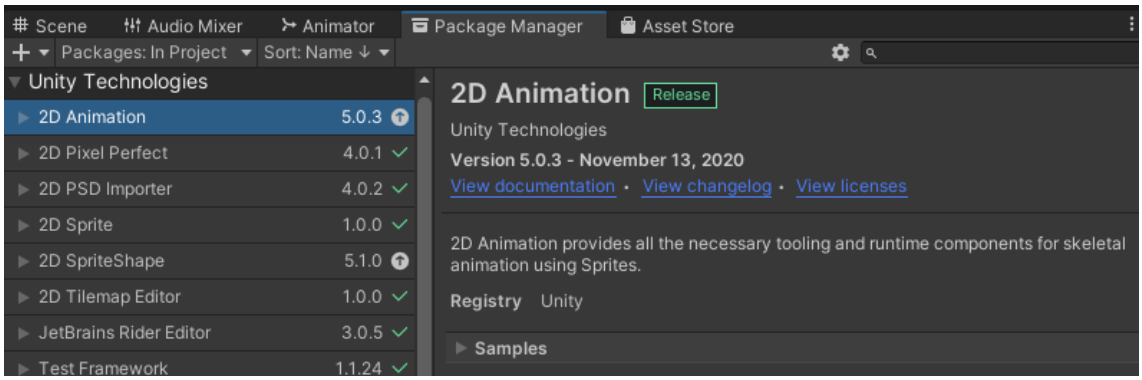
RULE OVERRIDE TILE

- Rule Override Tiles are Tiles which can override the Sprites and GameObjects for a given Rule Tile while maintaining the Rule set of the Rule Tile. This allows you to create Tiles that provide variations of a Rule Tile without setting new Rules.
- Depending on the Rule Tile that is overridden, there may be further properties which you can override here. Any public property in the Rule Tile that does not have a RuleTile.DontOverride attribute will be shown here and can be overridden.

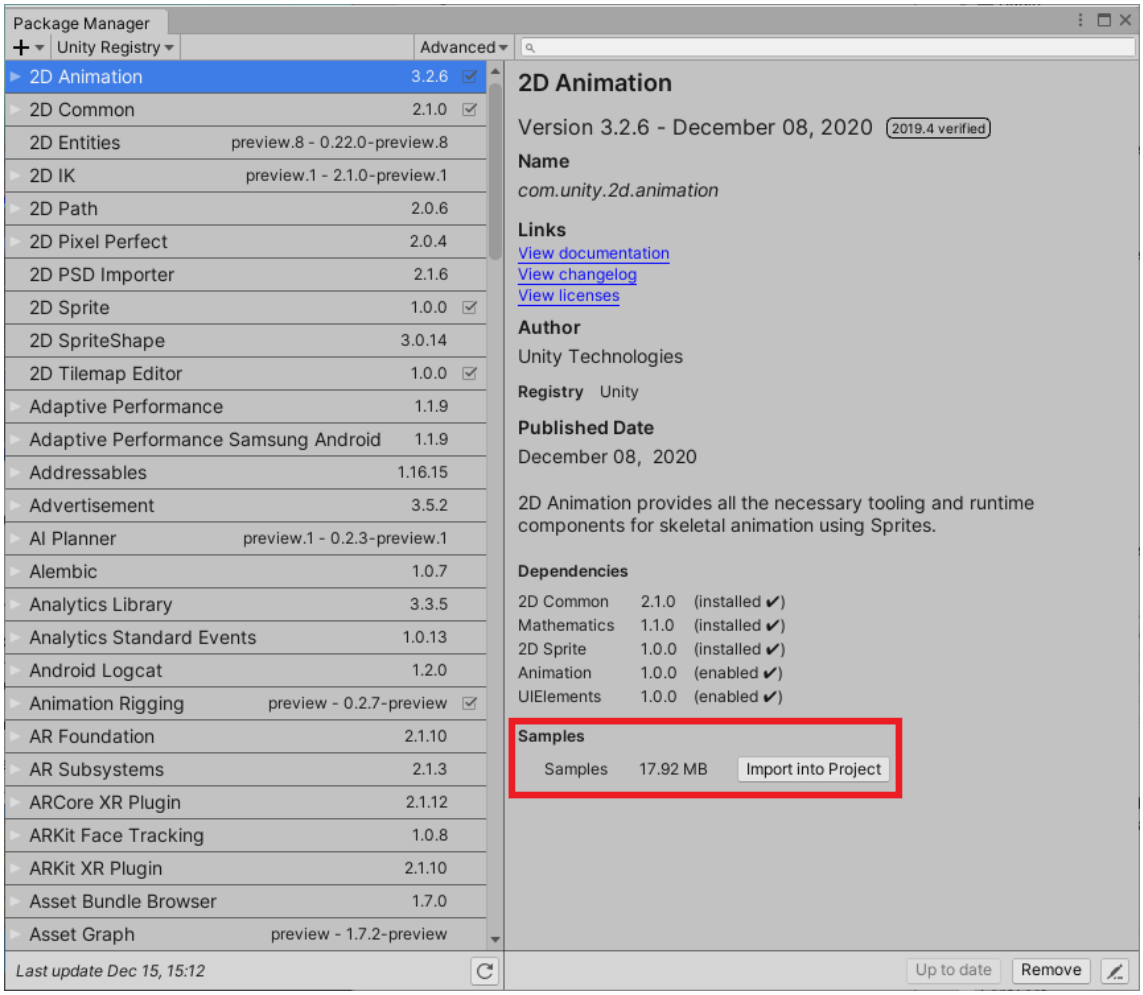


DAY 11: 2D ANIMATION

INSTALLATION



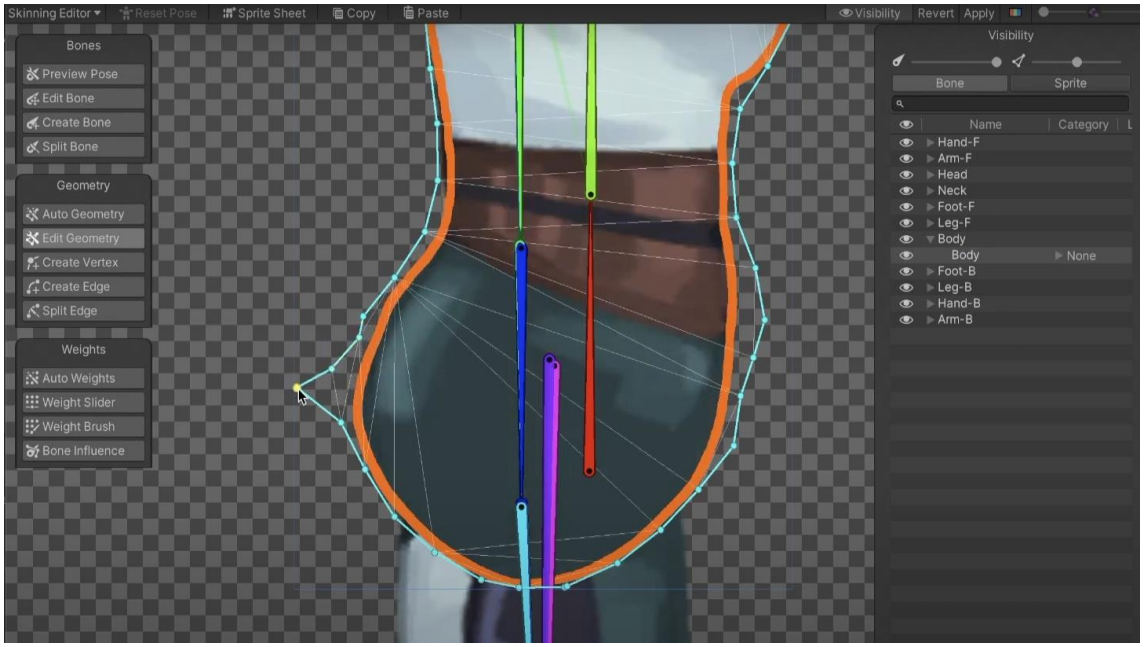
IMPORT SAMPLE



SKINNING EDITOR

- The Skinning Editor is used to create a skeleton in the character rigging process.
- The skeleton, or ‘bones’, (also known as 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 consists of three group of controls: Bones, Geometry and Weight.



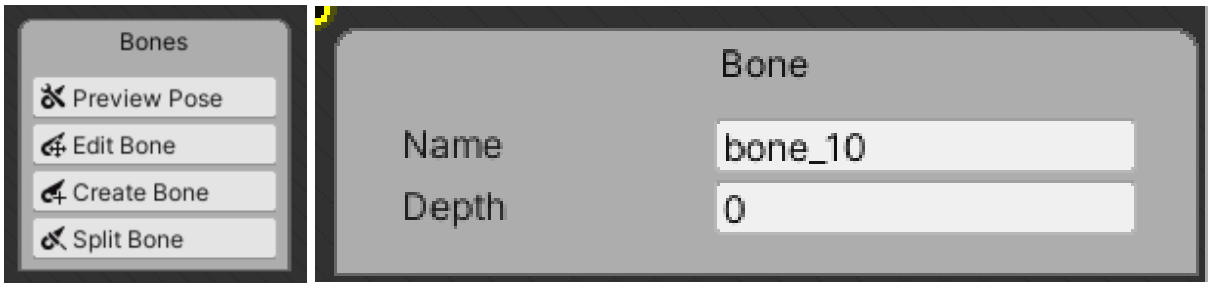
BONES

- **Preview Pose**: 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 change are not permanent and are only active during the preview.
- **Edit Joints**: move, reposition, disconnect, and reconnect bones.
    - To move a bone without disconnect its joints, click-and-drag either the root or the tip of the bone.
    - To disconnect a joint, click-and-drag a bone until it separates.
    - To reconnect the joint, click-and-drag its root to the unconnected tip of another bone, or its tip to the unconnected root of another bone.

BONES

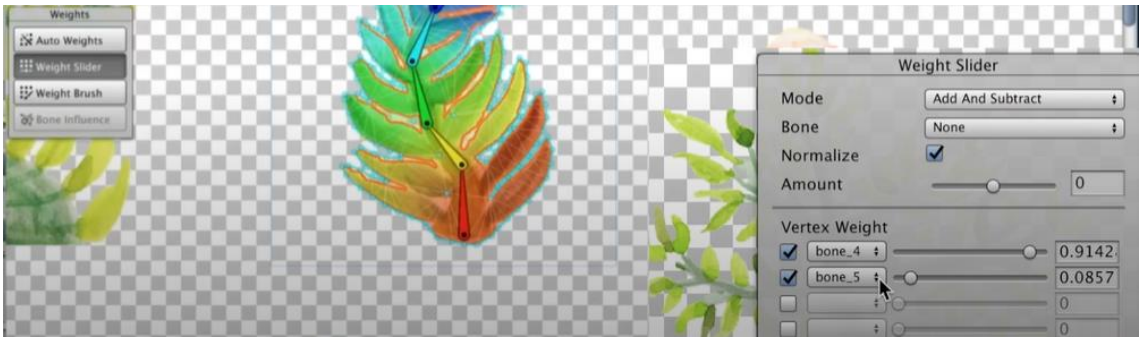
- Create Bone: allows you to create a bone.

- If you are switching to this tool from another, the bone will be default be a child of the root of previously selected bone chain, starting a new chain in Hierarchy.
- Alternatively, you can select the tip of an existing bone to make it the parent of the new bone.
- If you like to make a new bone without a parent, right-click or press ESC to break any connection.
- Left-click to place the bone's root and select once more to place the tip.

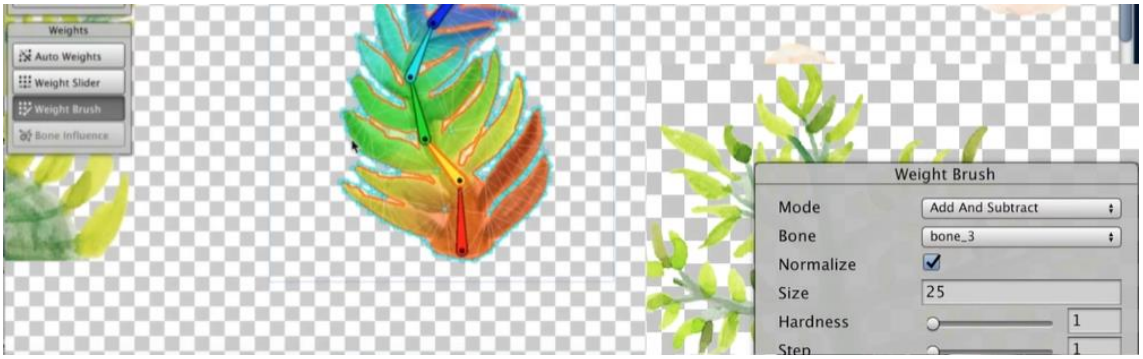


BONES FOR TREES

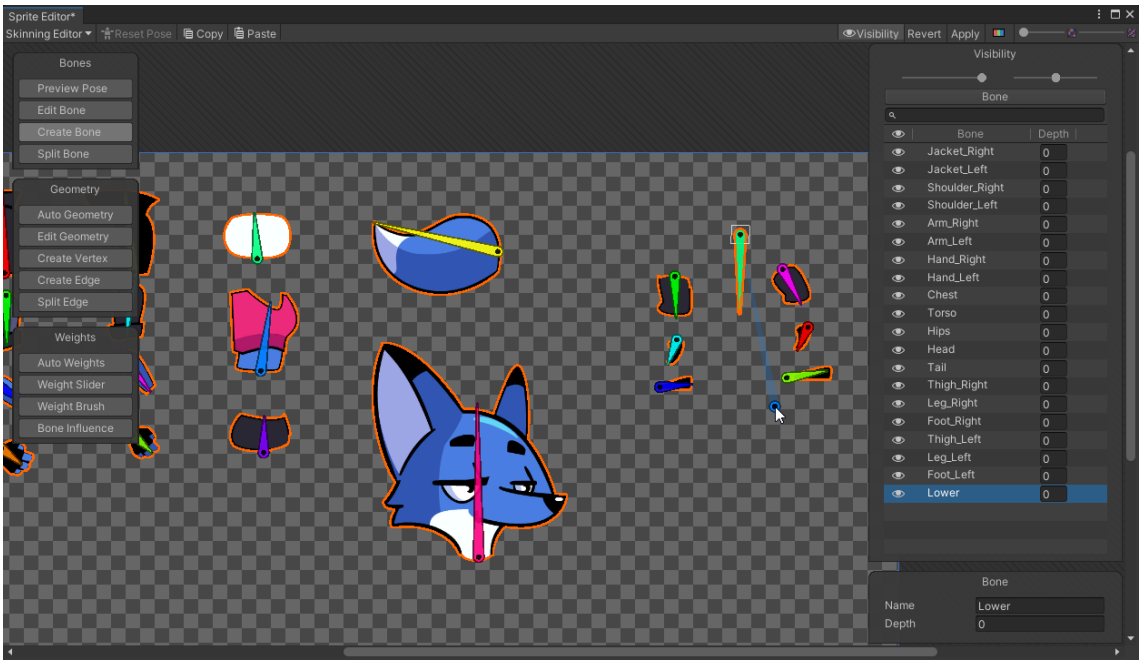
WEIGHT SLIDER FOR VERTEX



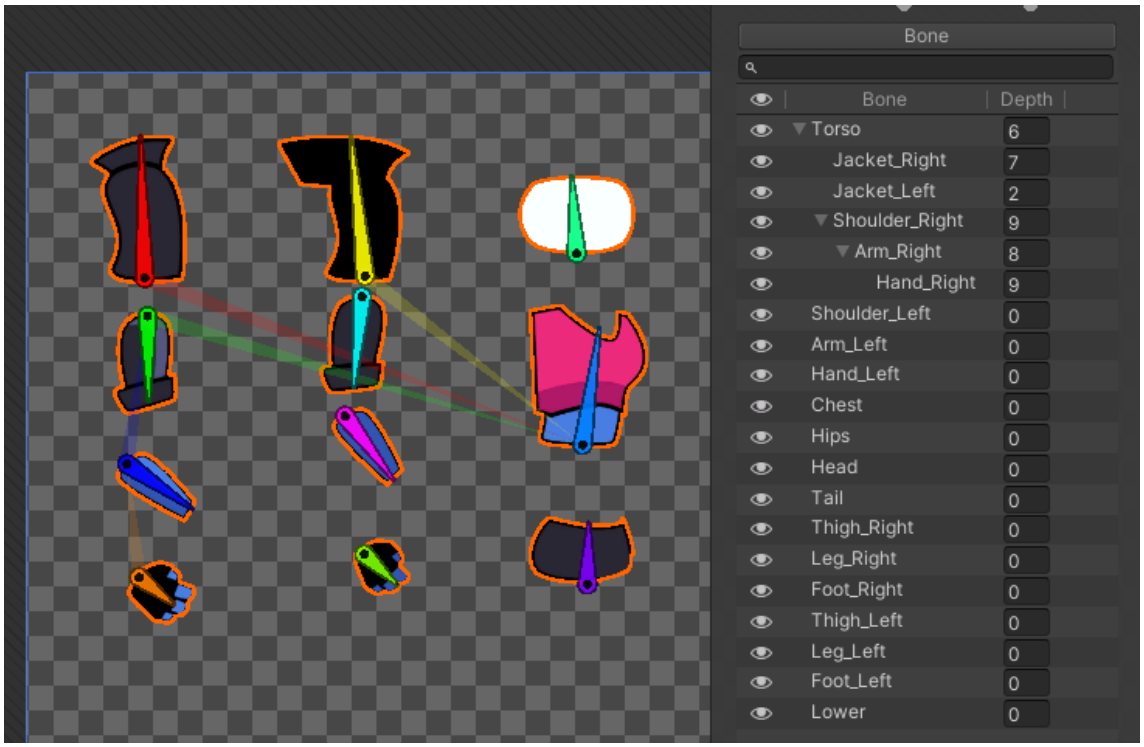
WEIGHT BRUSH (PAINT BONE ON VERTICES)



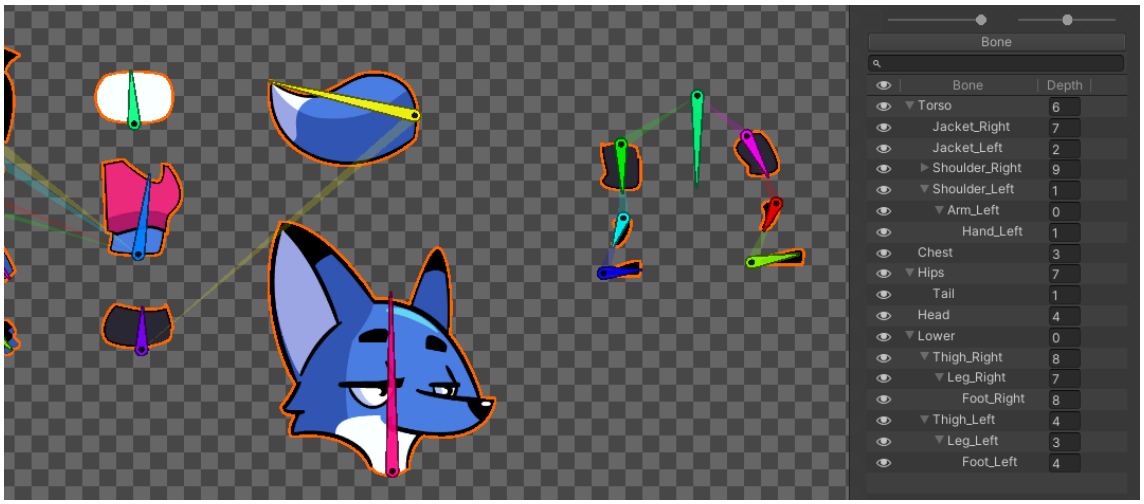
CREATE AND NAMING BONES



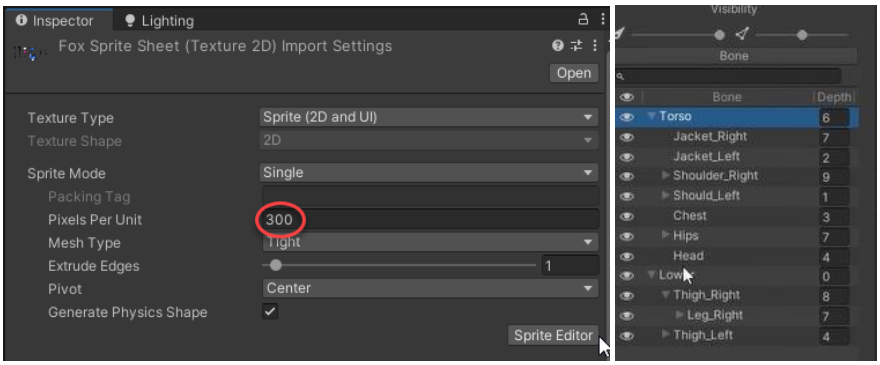
LINK RIGHT ARM



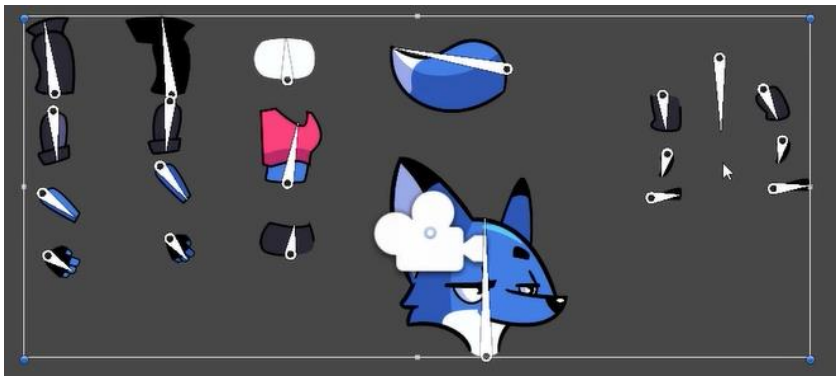
LINK LEGS

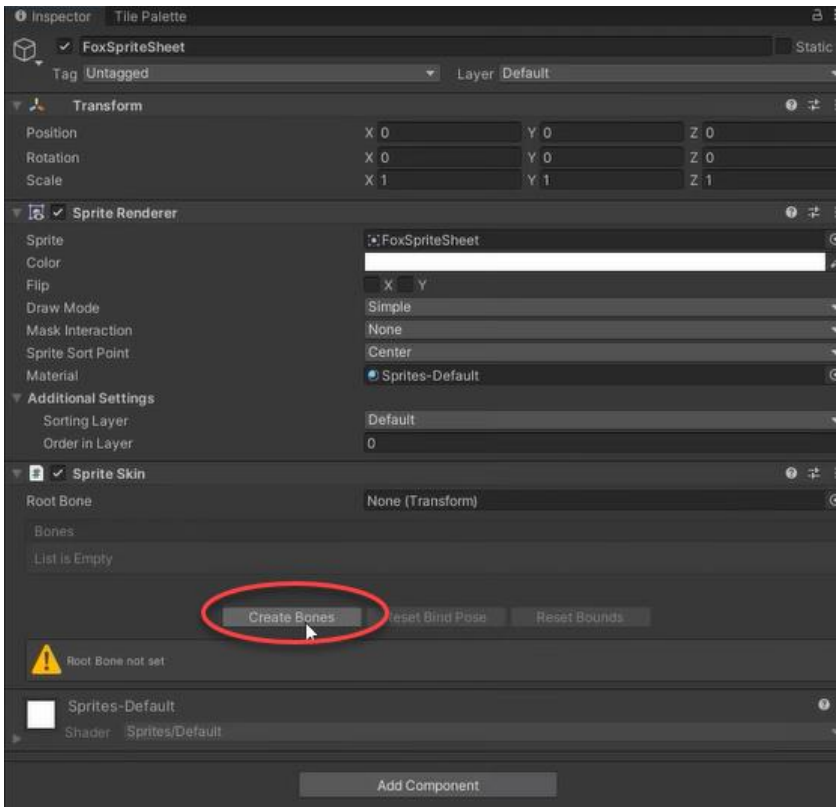


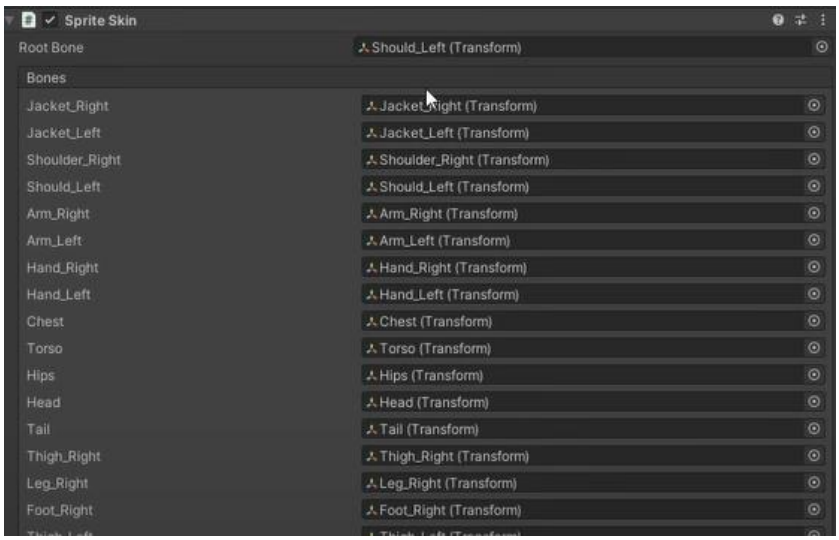
ADJUST SPRITE SIZE & BONE HIERARCHY



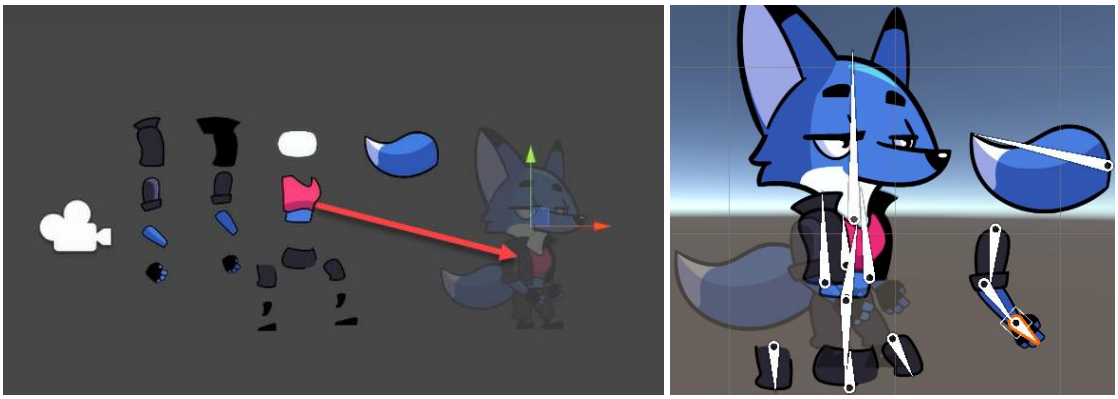
ADD SPRITE SKIN & BONES





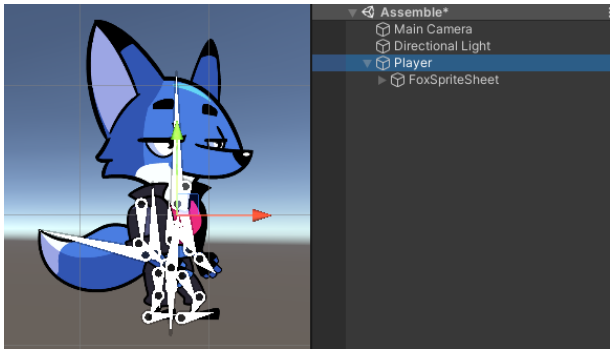


START ASSEMBLING THE FOX

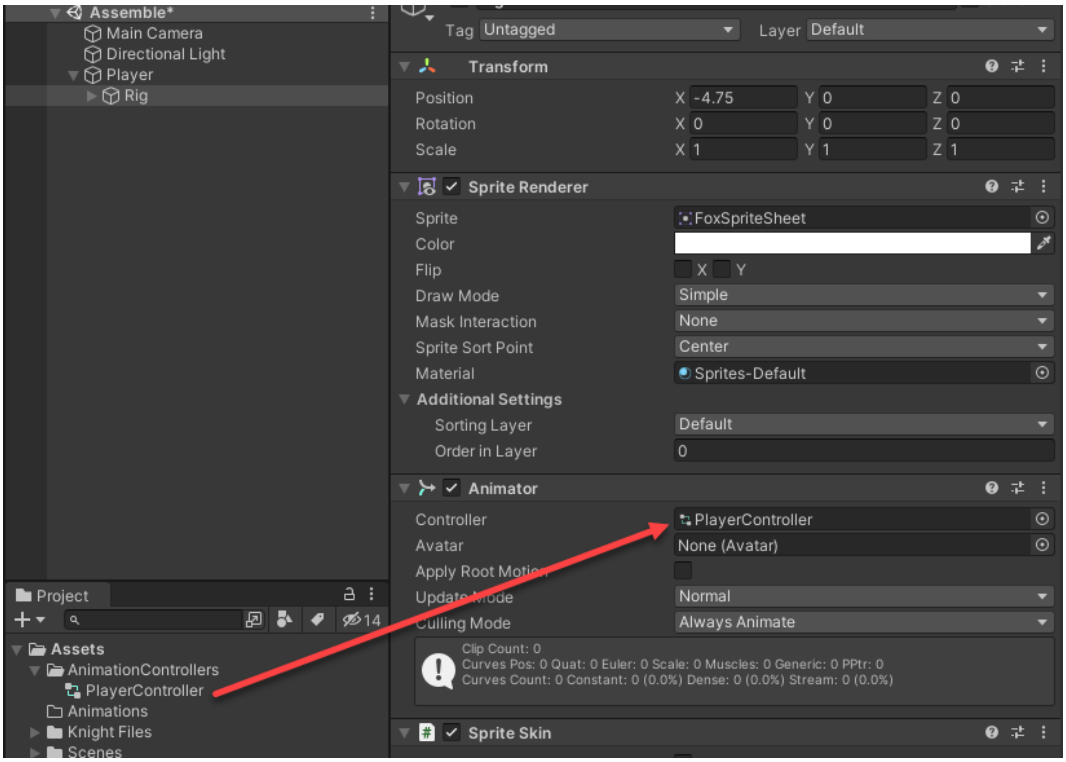


FINISH ASSEMBLING THE FOX

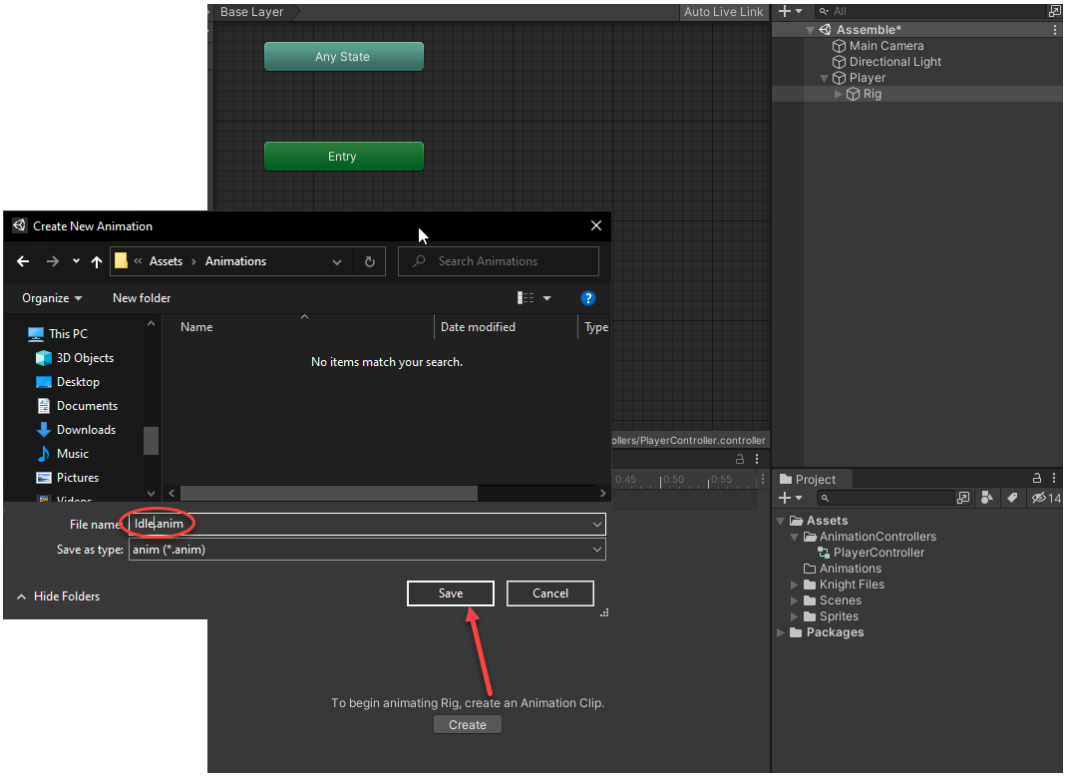
TRANSFORM PLAYER PIVOT



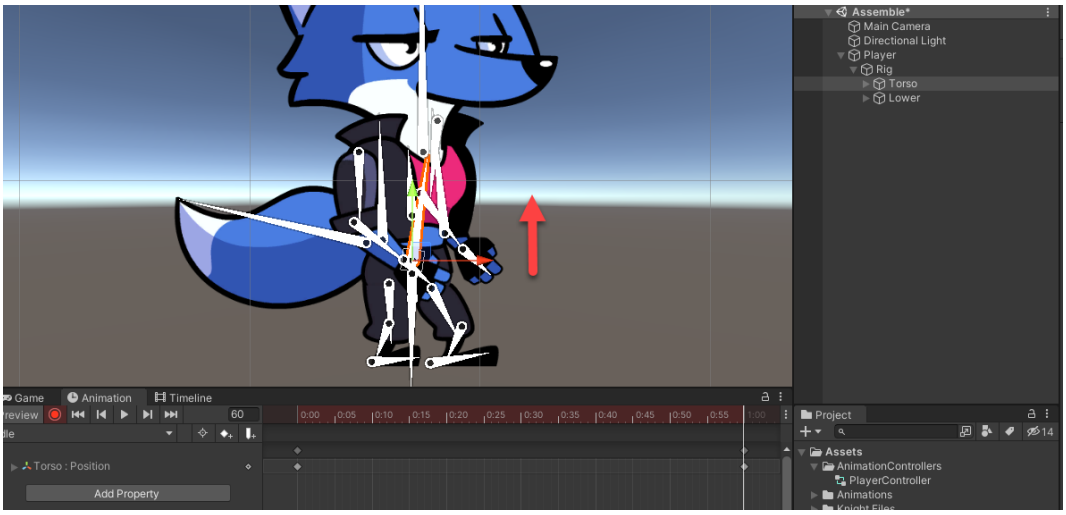
ADD ANIMATOR



CREATE IDLE ANIMATION



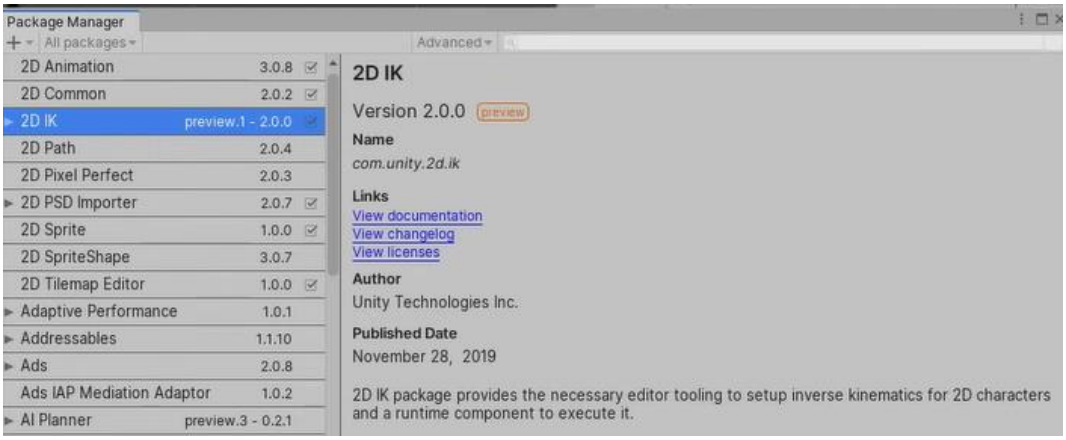
MOVE TORSO UP IN RECORDING MODE



DUPLICATE FIRST FRAME TO BE LAST FRAME

IK ANIMATION

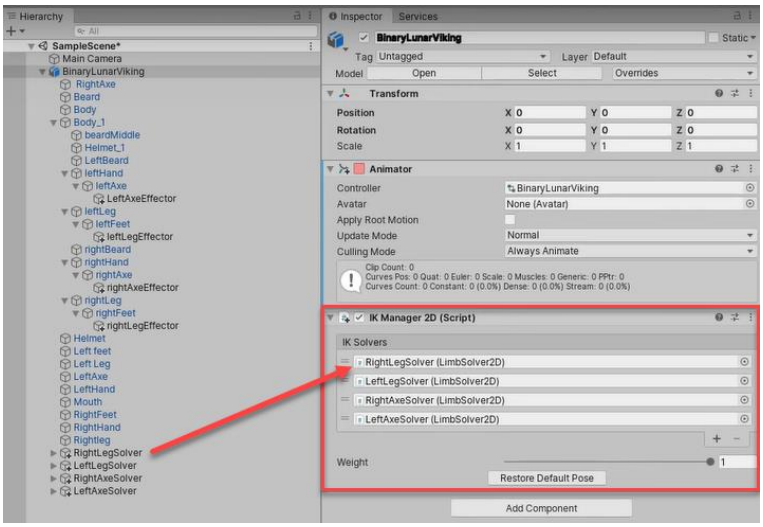
2D IK PREVIEW PACKAGE



WORKFLOW

- Refer to the hierarchy of bones created with the Bone Editor.
- Add the IK Manager 2D component to the GameObject at the top of the hierarchy. This is usually the main root bone of the entire character skeleton.
- Add to the IK Solvers list by selecting which type of IK Solver to use. The IK Solvers are also added as additional GameObjects in the hierarchy.
- With an IK Solver selected, select its Effector bone/Transform . Create or set a Target for the IK Solver.
- Position bones by moving the Target. Transforms to move the chain of bones with IK applied.

MANAGER & SOLVER

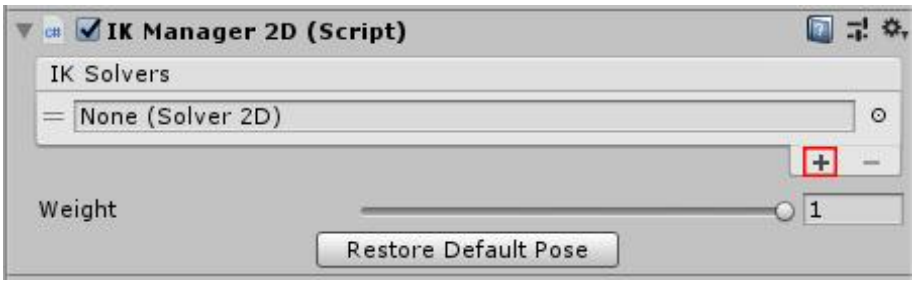


SOLVER TYPES

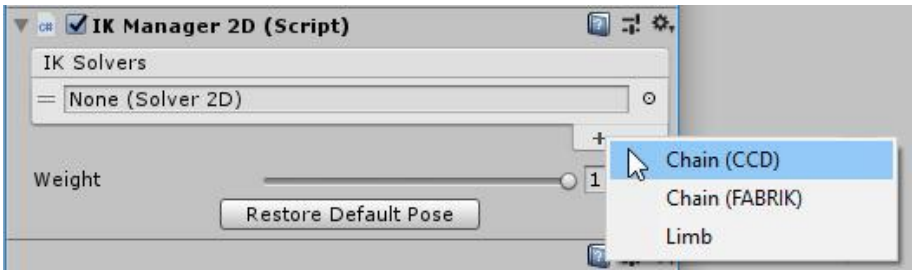
- Limb: a standard two bone Solver that is ideal for posing joints such as arms and legs. This Solver's chain length is fixed to three bones - starting from the Effector bone/Transform and including up to two additional bones in its chain.
- Chain (CCD): uses the Cyclic Coordinate Descent algorithm, which gradually becomes more accurate the more times the algorithm is run. The Solver stops running once the set tolerance or number of iterations is reached.
- Chain (FABRIK):
    - Uses the Forward And Backward Reaching Inverse Kinematics (FABRIK) algorithm. It is similar to Chain (CCD) as its solution becomes more accurate the more times its algorithm is run. The Solver stops running once the set tolerance or number of iterations is reached.
    - The Chain (FABRIK) Solver generally takes less iterations to reach the Target's destination compared to Chain (CCD), but is slower per iteration if rotation limits are applied to the chain. This Solver can adapt quickly to if the bones are manipulated in real-time to different positions.

SOLVER 2D

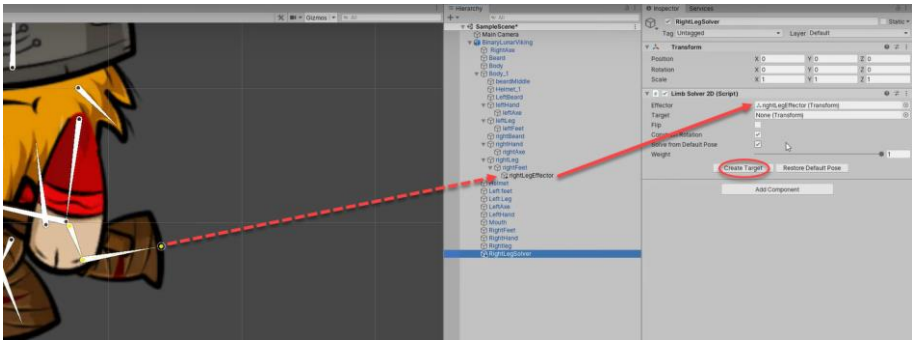
- To add an IK Solver, click the + symbol at the bottom right of the IK Solvers list



- A drop-down menu then appears with three options - Chain (CCD), Chain (FABRIK), and Limb. Each type of IK Solver uses a different algorithm to solve for the position of Effectors.



EFFECTOR & TARGET



REFERENCES

- How to make a 2D platformer:

https://www.youtube.com/watch?v=Cr5Hs0CJ1Sg

- Unity 2D Animation 2020 – Bones & Rig:

https://www.youtube.com/watch?v=k4LkNtp9_wU

- Rigging a Sprite with the 2D Animation Package:

https://learn.unity.com/tutorial/rigging-a-sprite-with-the-2d-animation-package

- 2D Animation & IK Tutorial:

https://www.youtube.com/watch?v=Ed8CoET9b6U

- 2D Inverse Kinematics (IK):

https://docs.unity3d.com/Packages/com.unity.2d.ik@1.1/manual/index.html

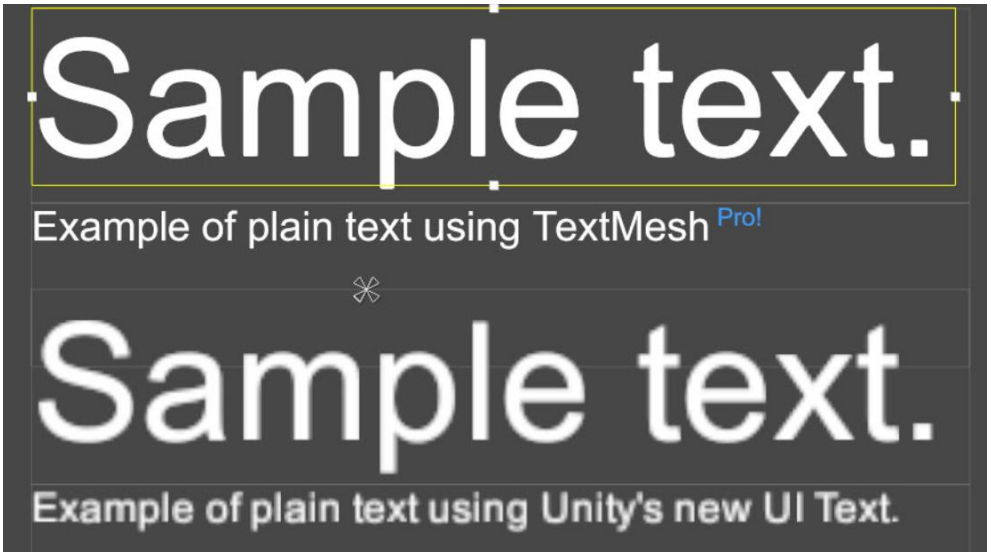
DAY 12: ADVANCED UI FEATURES

TextMesh Pro

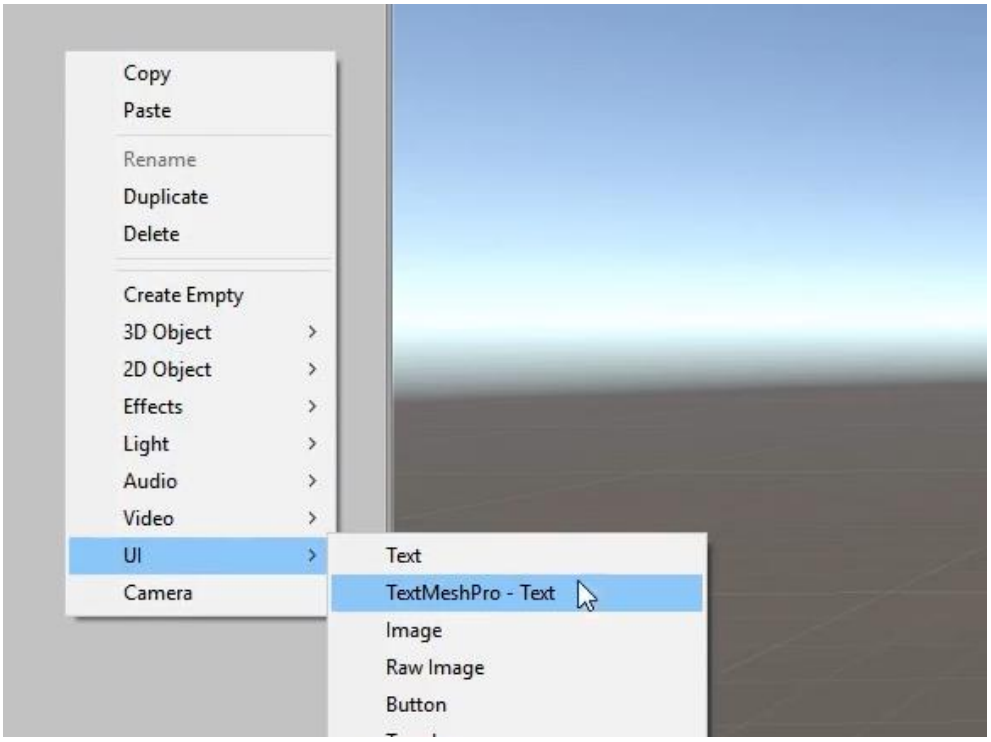
UI Drag & Drop

TEXTMESH PRO

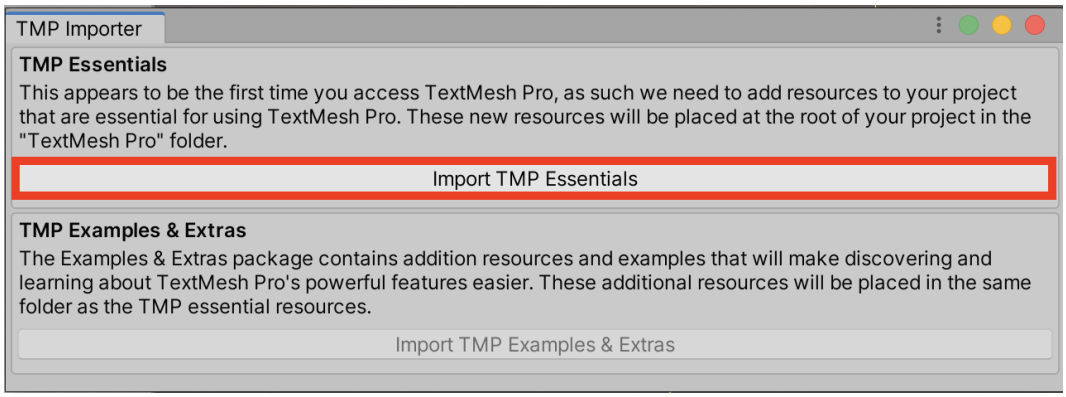
TEXT MESH PRO VS UI TEXT



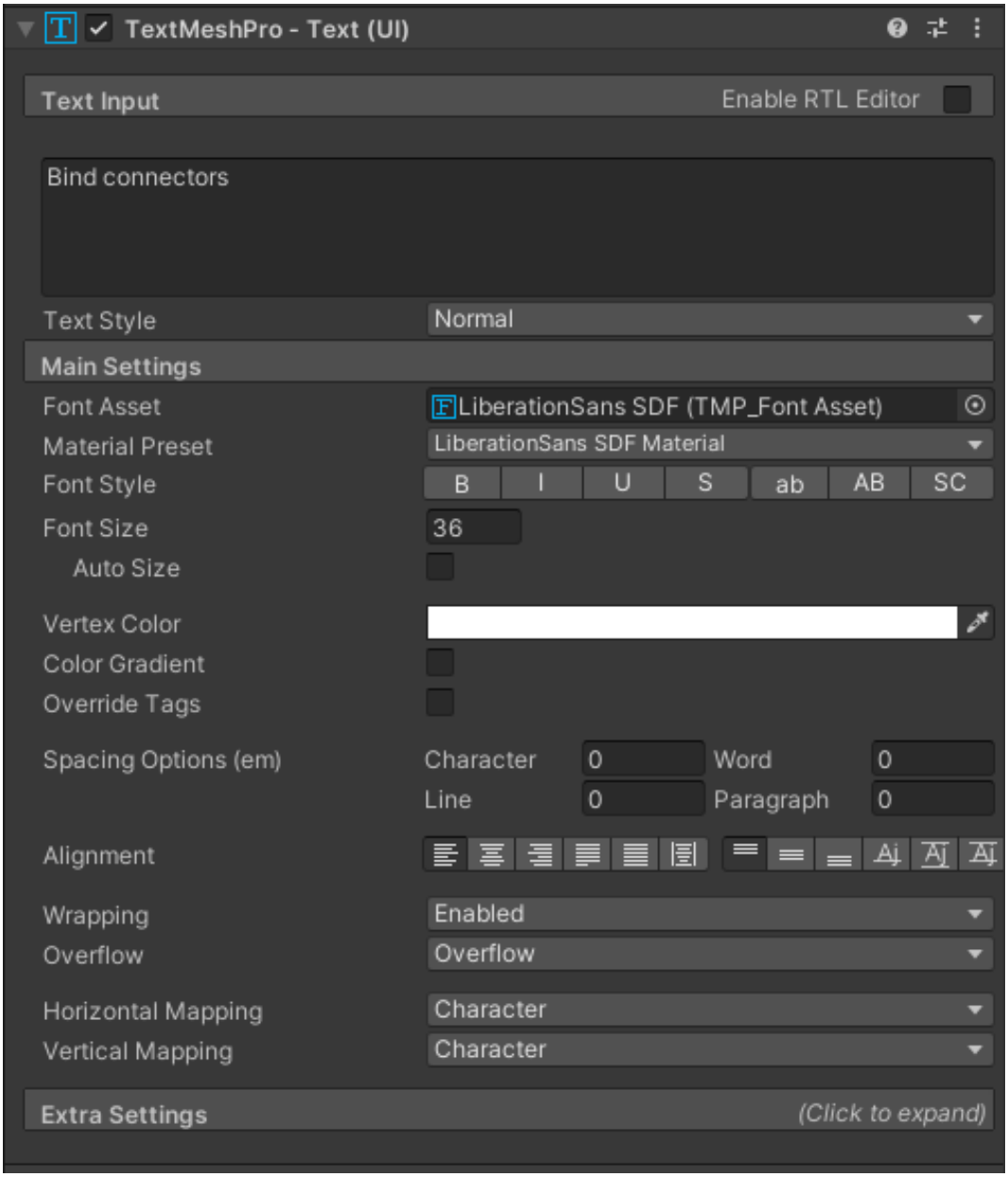
CREATE TEXT MESH PRO TEXT



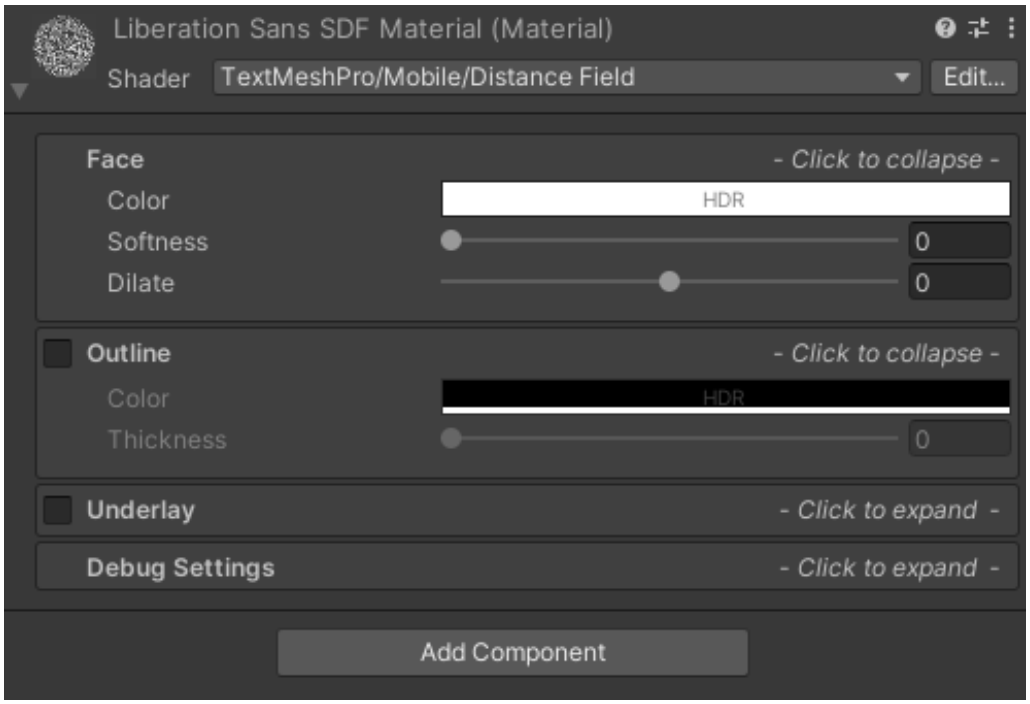
IMPORT TMP ESSENTIALS



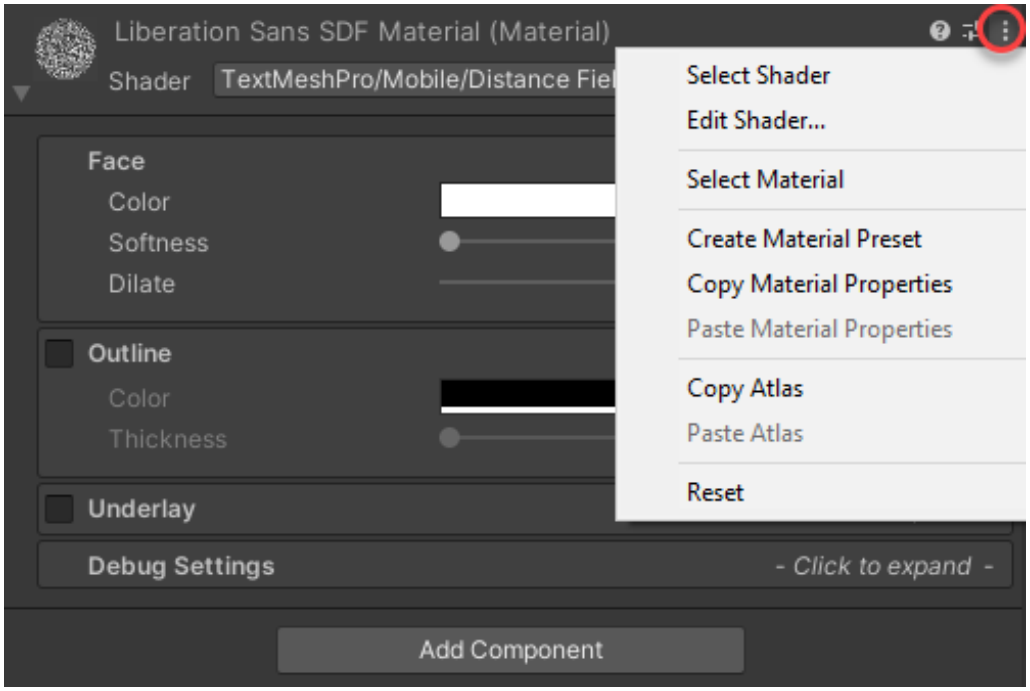
TMP TEXT COMPONENT



TMP MATERIAL



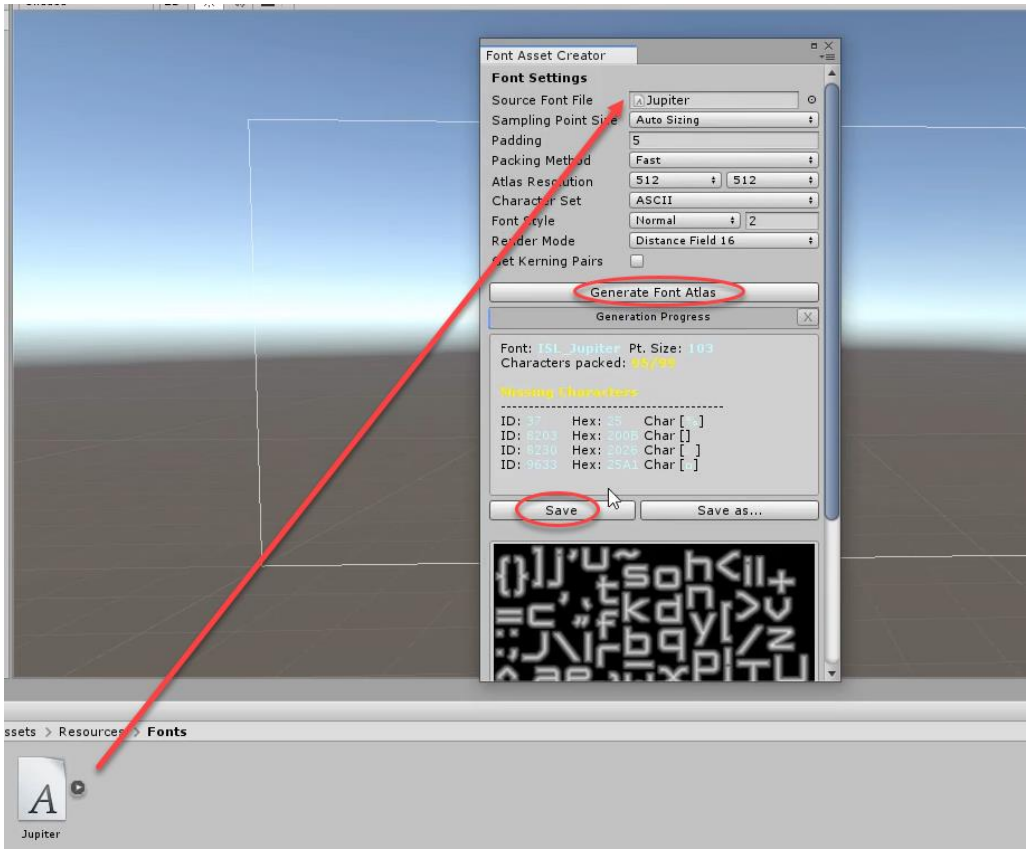
WORKING WITH MATERIAL PRESETS



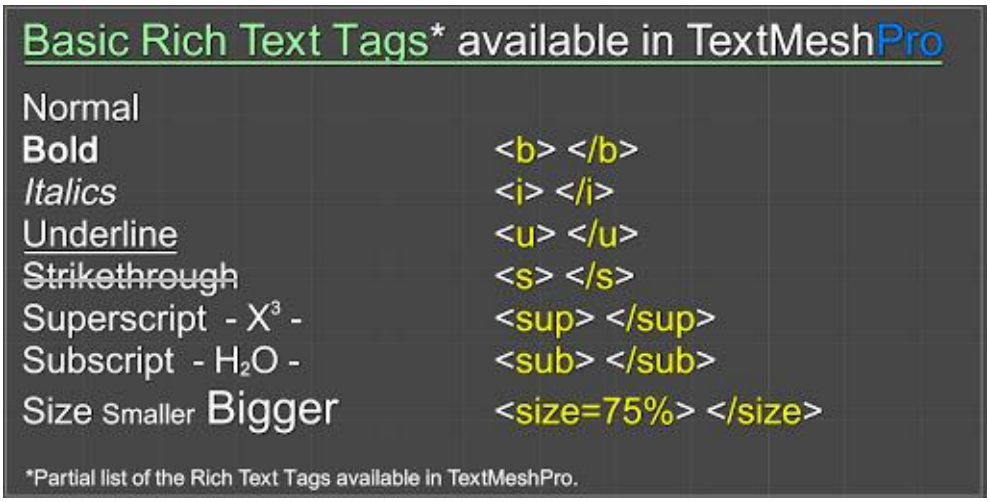
ACCESS TMP TEXT

```
using TMPro;
using UnityEngine;
using UnityEngine.UI;

Unity Script | 1 reference
public class BindingConnectorDialog : MonoBehaviour
{
    public TMP_InputField inputDistance;
    public TMP_InputField inputRotation;
    public PartSelector partSelector;
    public RotationByMouse floorRotation;

    Unity Message | 0 references
    public void OnEnable()
    {
        inputDistance.text = "0";
        inputRotation.text = "0";
```

CREATE FONT ASSET

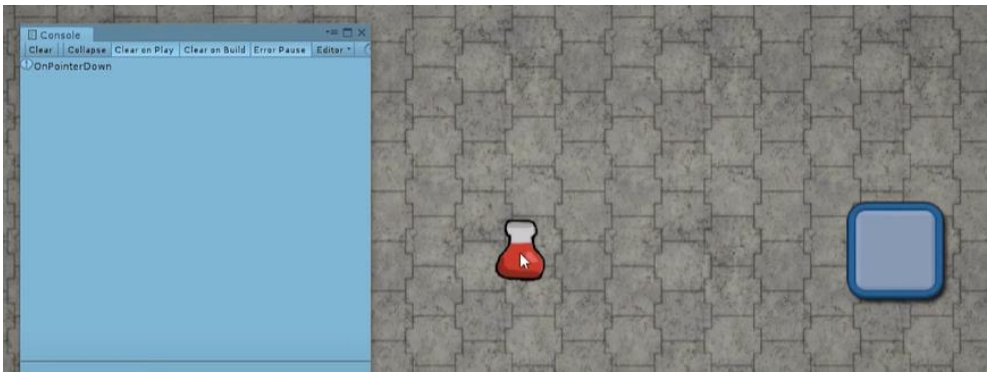


RICH TEXT TAGS



UI DRAG & DROP

POINTER DOWN HANDLER




```
using UnityEngine;
using UnityEngine.EventSystems;

public class DragDrop : MonoBehaviour, IPointerDownHandler {

    public void OnPointerDown(PointerEventData eventData) {
        Debug.Log("OnPointerDown");
    }

}
```

BEGIN & END DRAG

```
using UnityEngine;
using UnityEngine.EventSystems;

public class DragDrop : MonoBehaviour, IPointerDownHandler, IBeginDragHandler, IEndDragHandler {

    public void OnBeginDrag(PointerEventData eventData) {
        Debug.Log("OnBeginDrag");
    }

    public void OnEndDrag(PointerEventData eventData) {
        Debug.Log("OnEndDrag");
    }

    public void OnPointerDown(PointerEventData eventData) {
        Debug.Log("OnPointerDown");
    }

}
```

MOVE ITEM ON DRAG

```
public class DragDrop : MonoBehaviour, IPointerDownHandler, IBeginDragHandler, IEndDragHandler, IDragHandler {

    private RectTransform rectTransform;

    private void Awake() {
        rectTransform = GetComponent<RectTransform>();
    }

    public void OnBeginDrag(PointerEventData eventData) {
        Debug.Log("OnBeginDrag");
    }

    public void OnDrag(PointerEventData eventData) {
        Debug.Log("OnDrag");
        rectTransform.anchoredPosition += eventData.delta;
    }
```

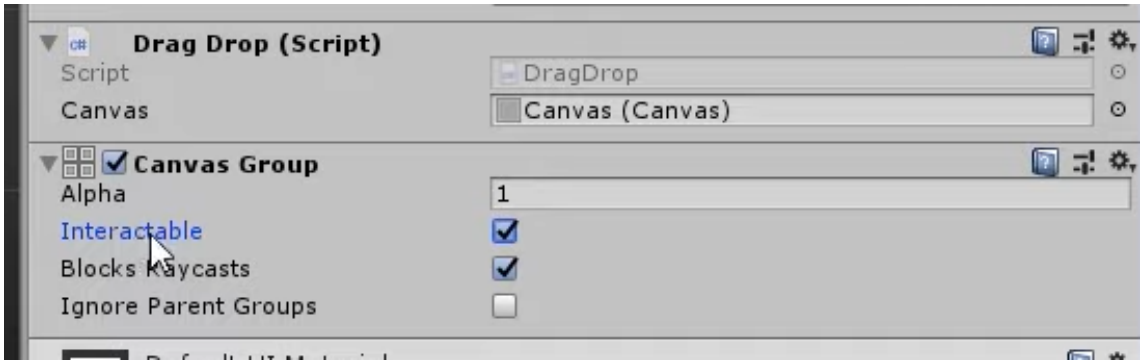
ADJUST MOVEMENT BY SCALE FACTOR

```
public class DragDrop : MonoBehaviour, IPointerDownHandler, IBeginDragHandler, IEndD

    [SerializeField] private Canvas canvas;

    private RectTransform rectTransform;

    private void Awake() {
        rectTransform = GetComponent<RectTransform>();
    }

    public void OnBeginDrag(PointerEventData eventData) {
        Debug.Log("OnBeginDrag");
    }

    public void OnDrag(PointerEventData eventData) {
        Debug.Log("OnDrag");
        rectTransform.anchoredPosition += eventData.delta / canvas.scaleFactor;
    }
```

DROP HANDLER ON ITEM SLOT

```
using UnityEngine;
using UnityEngine.EventSystems;

public class ItemSlot : MonoBehaviour, IDropHandler {

    public void OnDrop(PointerEventData eventData) {
        Debug.Log("OnDrop");
    }
}
```

DISABLE INTERACTION WHILE DRAGGING




```
private RectTransform rectTransform;
private CanvasGroup canvasGroup;

private void Awake() {
    rectTransform = GetComponent<RectTransform>();
    canvasGroup = GetComponent<CanvasGroup>();
}

public void OnBeginDrag(PointerEventData eventData) {
    Debug.Log("OnBeginDrag");
    canvasGroup.alpha = .6f;
    canvasGroup.blocksRaycasts = false;
}

public void OnDrag(PointerEventData eventData) {
    Debug.Log("OnDrag");
    rectTransform.anchoredPosition += eventData.delta / canvas.scaleFactor;
}

public void OnEndDrag(PointerEventData eventData) {
    Debug.Log("OnEndDrag");
    canvasGroup.alpha = 1f;
    canvasGroup.blocksRaycasts = true;
}
```

ITEM SNAPPING

```
public class ItemSlot : MonoBehaviour, IDropHandler {

    public void OnDrop(PointerEventData eventData) {
        Debug.Log("OnDrop");
        if (eventData.pointerDrag != null) {
            eventData.pointerDrag.GetComponent<RectTransform>().anchoredPosition = GetComponent<RectTransform>().anchoredPosition
        }
    }

}
```

REFERENCES

- Quick Start to TextMesh Pro:

https://learn.unity.com/tutorial/working-with-textmesh-pro

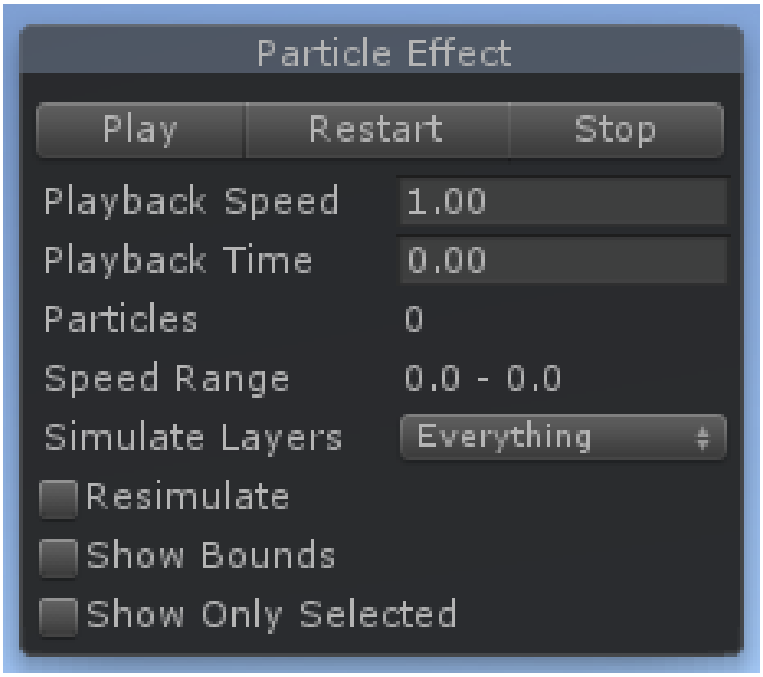
- Simple Drag & Drop:

https://www.youtube.com/watch?v=BGr-7GZJNXg

DAY 13: PARTICLE SYSTEM

PARTICLE EFFECT PANEL

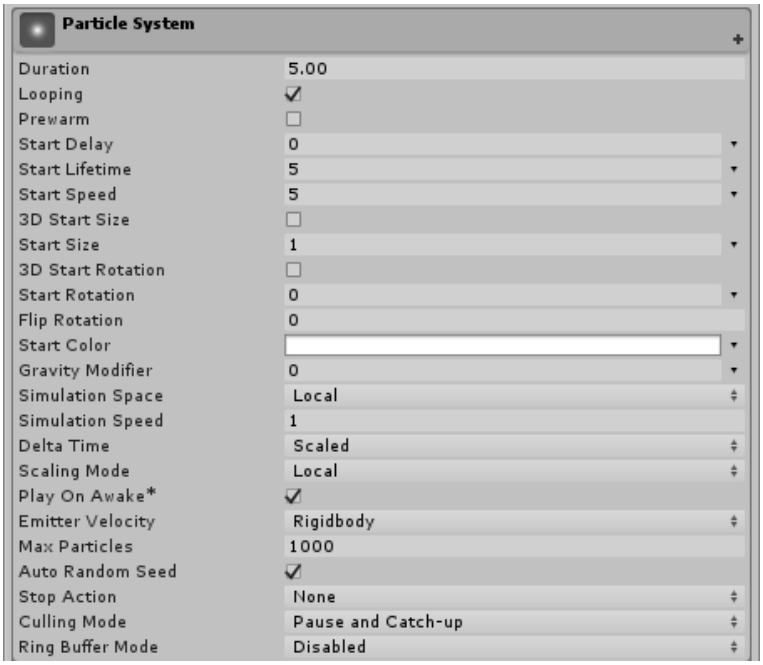
- Simulate Layers: Allows you to preview Particle Systems that are not selected. By default, only selected Particle Systems play in the Scene View.
- Resimulate: the Particle System immediately applies property changes to particles it has already generated.
- Show Bounds: displays the bounding volume around the selected Particle Systems. These bounds determine whether a Particle System is currently on screen or not.
- Show Only Selected: hides all non-selected Particle Systems, allowing you to focus on producing a single effect.



MAIN MODULE

- The system emits particles for a specific duration, and can be set to emit continuously using the Looped property.
- The Start properties (lifetime, speed, size, rotation and color) specify the state of a particle on emission. You can specify a particle's width, height and depth independently, using the 3D Start Size property.

- All Particle Systems use the same gravity vector specified in the Physics settings. The Gravity Multiplier value can be used to scale the gravity, or switch it off if set to zero.



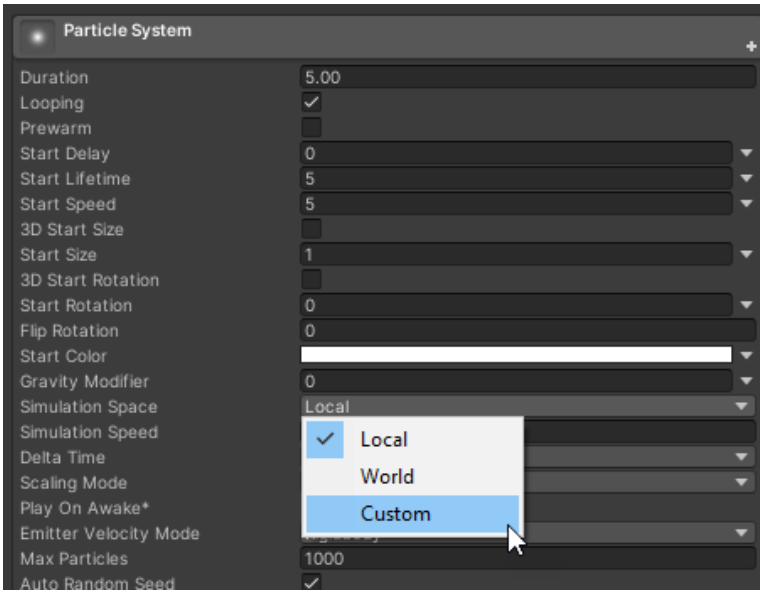
NON-UNIFORM PARTICLE SCALING

- The 3D Start Size property allows you to specify a particle's width, height and depth independently. In the Particle System Main module, check the 3D Start Size checkbox, and enter the values for the initial x (width), y (height) and z (depth) of the particle. Note that z (depth) only applies to 3D Mesh particles. You can also set randomized values for these properties, in a range between two constants or curves.
- You can set the particle's initial size in the Particle System Main module, and its size over the particle's lifetime using the Separate Axes option in the Size over Lifetime module. You can also set the particle's size in relation to its speed using the Separate Axes option in the Size by Speed module.

SIMULATION SPACE

- The Simulation Space property determines whether the particles move with the Particle System parent object, a custom object, or independently in the game world.

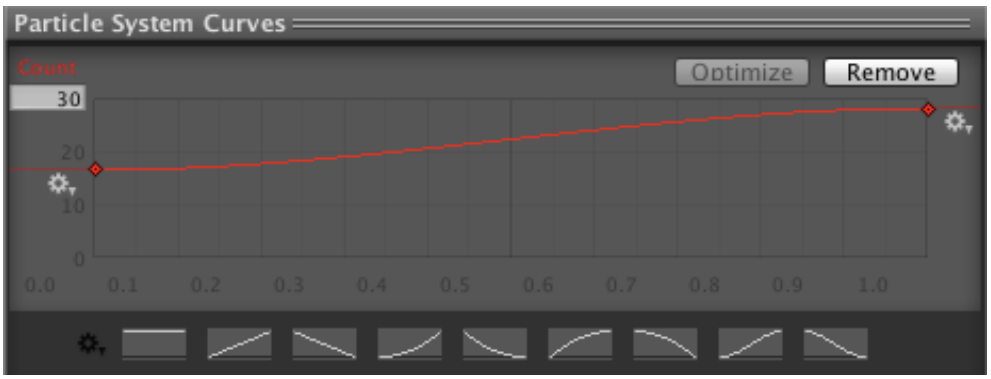
- When set to Custom, particles no longer move relative to their own Transform component. Instead, they all move relative to the movement of the specified Transform component.

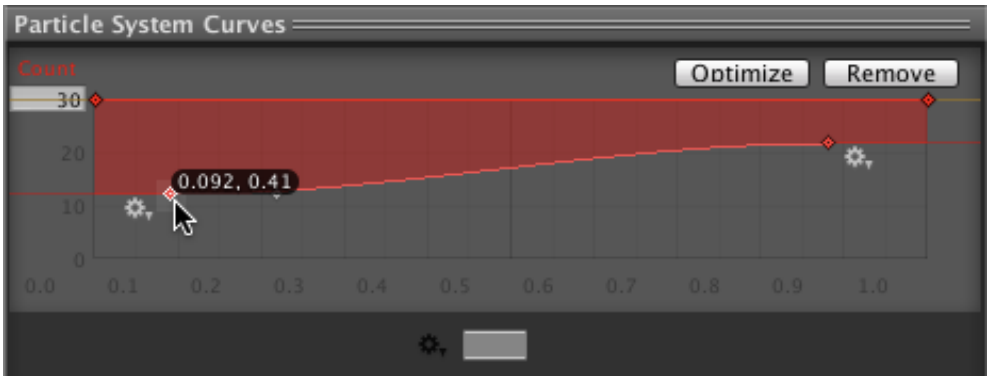


VARYING PROPERTIES OVER TIME

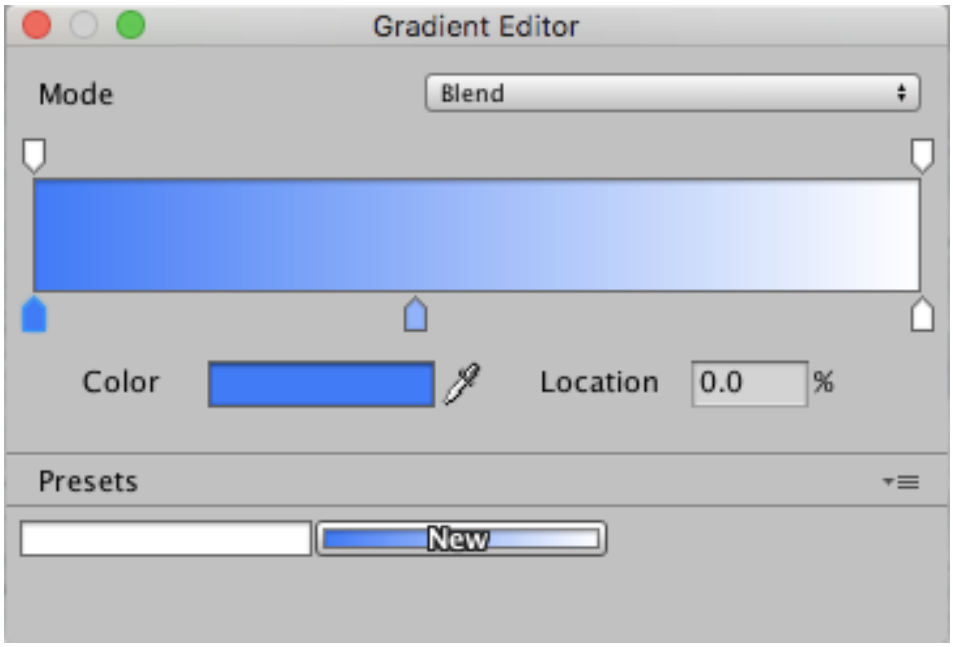
- Constant: The property's value is fixed throughout its lifetime.
- Curve: The value is specified by a curve/graph.
- Random Between Two Constants: Two constant values define the upper and lower bounds for the value; the actual value varies randomly over time between those bounds.
- Random Between Two Curves: Two curves define the upper and lower bounds of the value at a given point in its lifetime; the current value varies randomly between those bounds.

CURVES

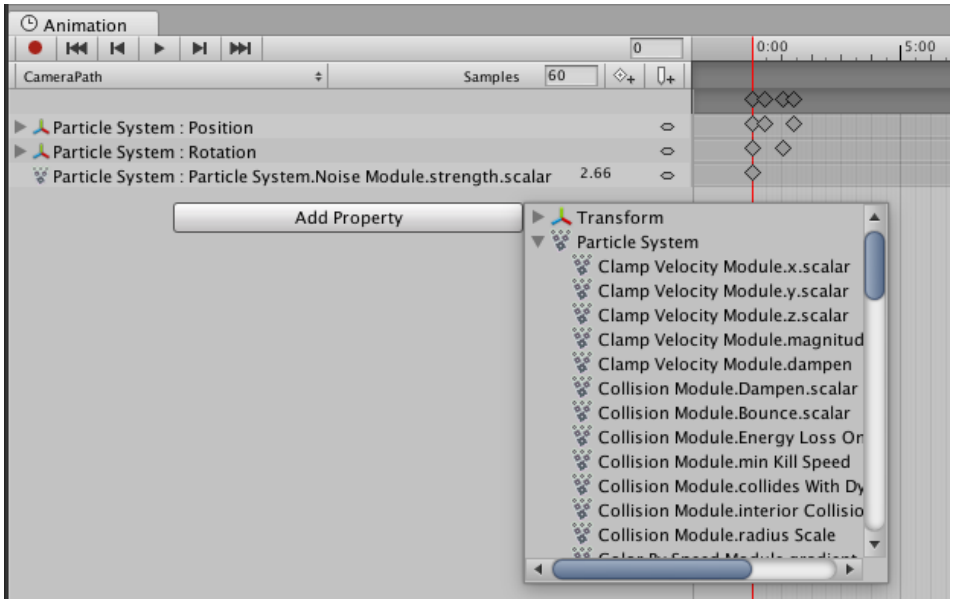




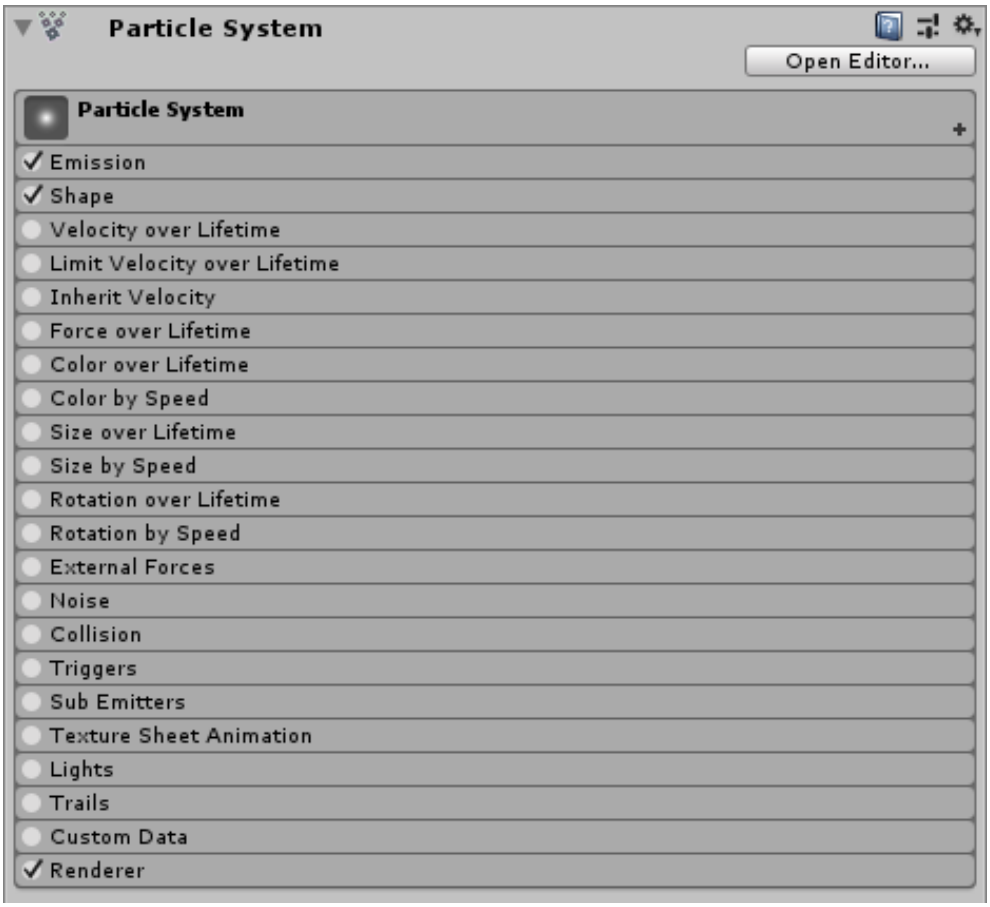
GRADIENT



ANIMATION BINDING

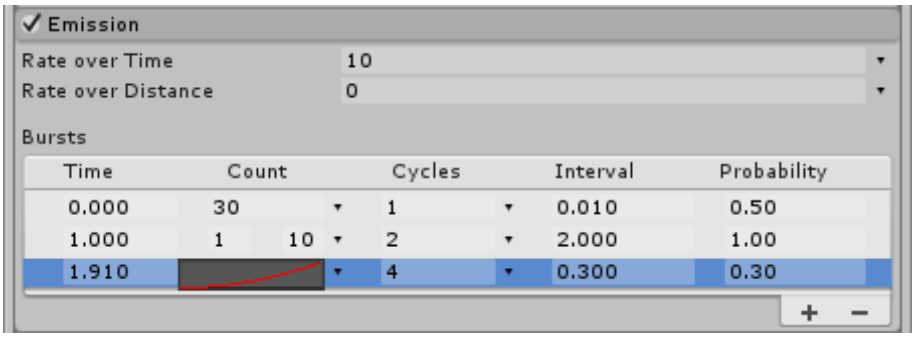


MODULES

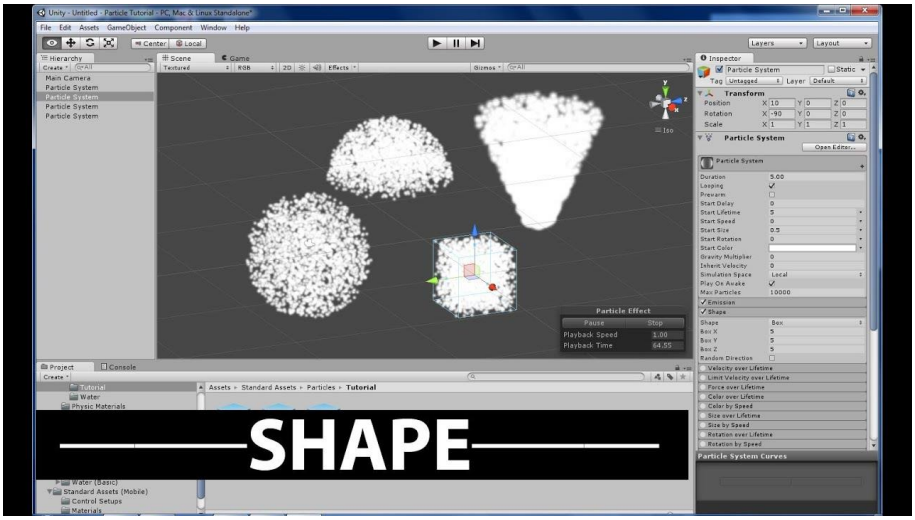


EMISSION MODULE

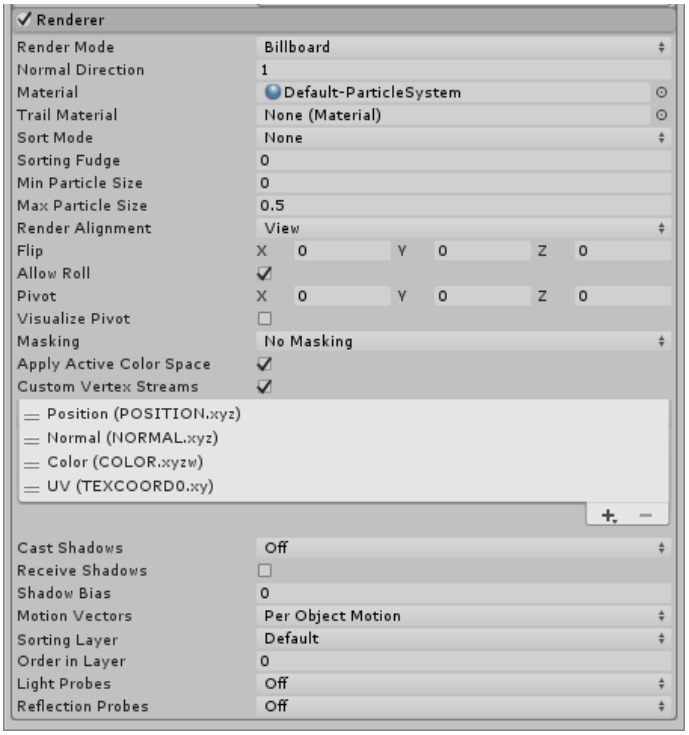
- Rate over Time: The number of particles emitted per unit of time.
- Rate over Distance: The number of particles emitted per unit of distance moved.
- Bursts: A burst is an event which spawns particles. These settings allow particles to be emitted at specified times.
    - Time: Set the time (in seconds, after the Particle System begins playing) at which to emit the burst.
    - Count: Set a value for the number of particles that may be emitted.
    - Cycles: Set a value for how many times to play the burst.
    - Interval: Set a value for the time (in seconds) between when each cycle of the burst is triggered.
    - Probability: Controls how likely it is that each burst event spawns particles. A higher value makes the system produce more particles, and a value of 1 guarantees that the system produces particle



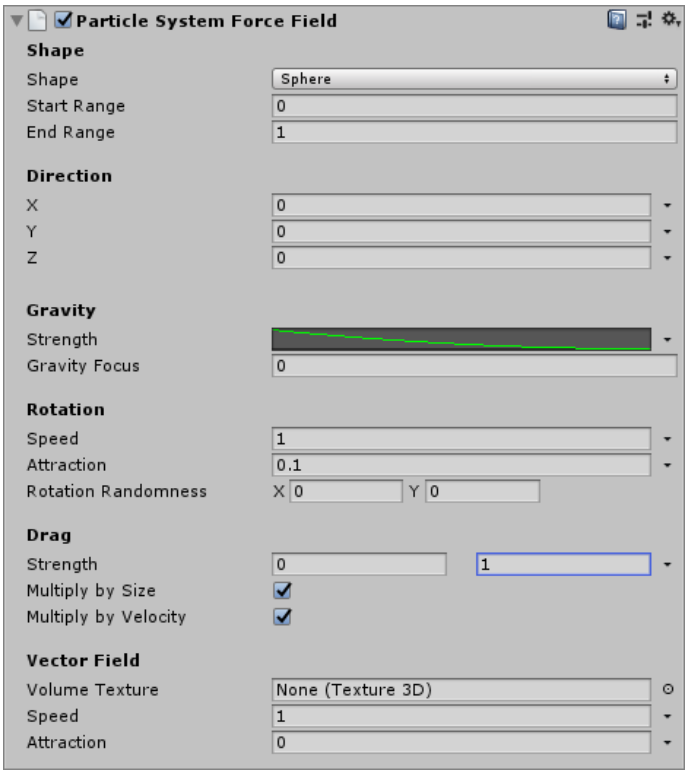
SHAPE MODULE



RENDERER MODULE



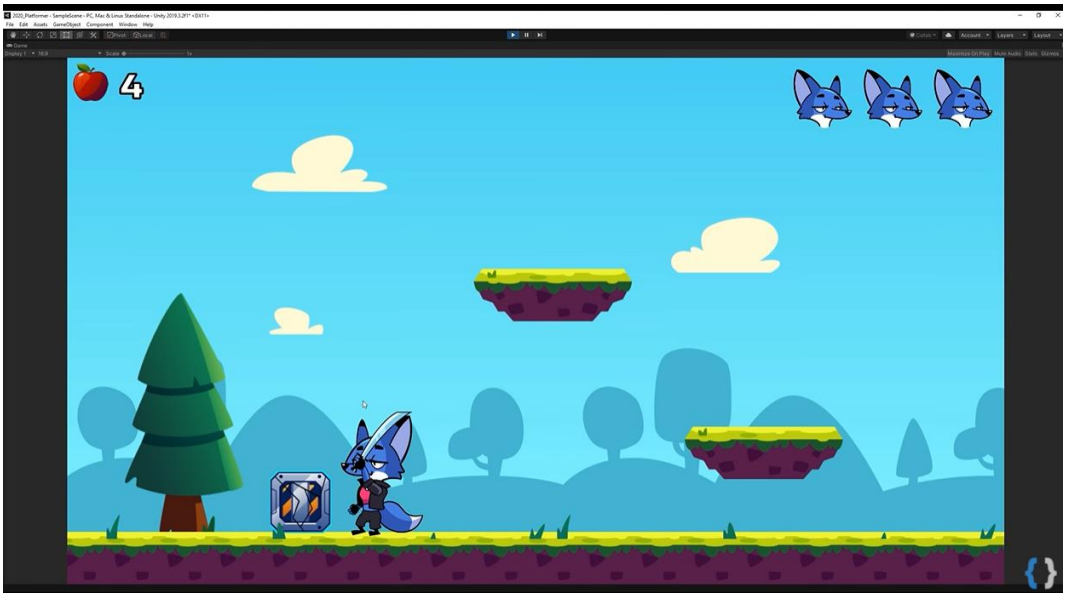
FORCE FIELD


Particle System Force Field
Shape
Shape
Sphere
Start Range
0
End Range
1
Direction
X
0
Y
0
Z
0
Gravity
Strength
Gravity Focus
0
Rotation
Speed
1
Attraction
0.1
Rotation Randomness
X 0
Y 0
Drag
Strength
0
1
Multiply by Size
Multiply by Velocity
Vector Field
Volume Texture
None (Texture 3D)
Speed
1
Attraction
0

DAY 14: SPECIAL TECHNIQUES

PLATFORMER 2D

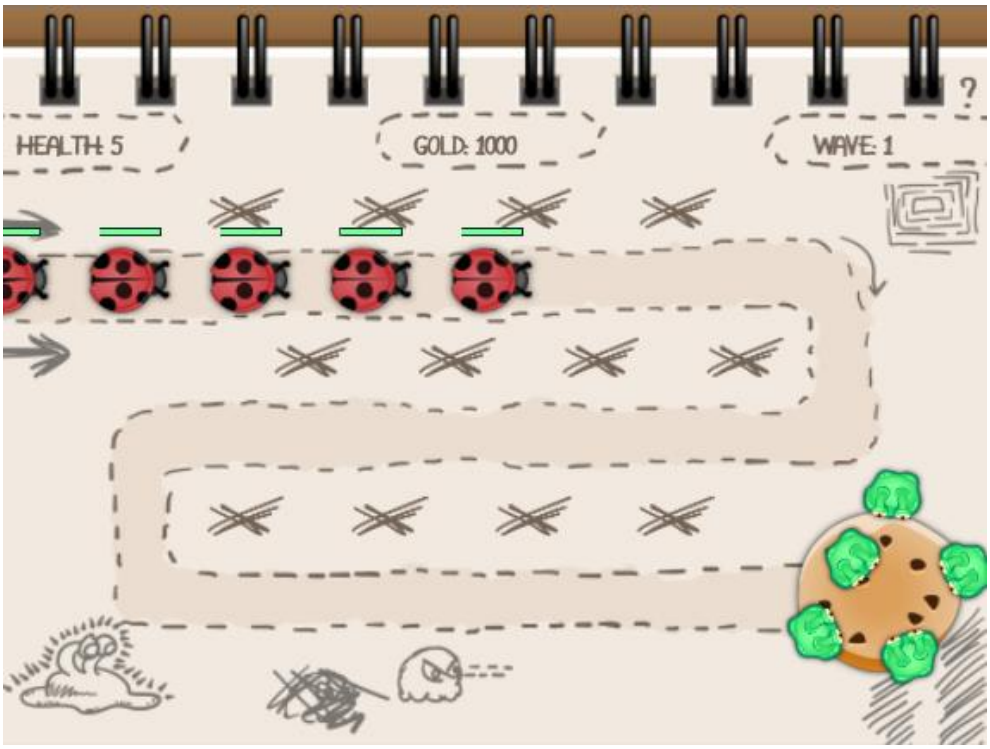
- https://www.youtube.com/watch?v=NC1qMcsJsG8



- Melee Combat System
    - https://laptrinhx.com/how-to-make-2d-melee-combat-in-unity-practical-tutorials-4231143771/
- Isometric Tile Map
    - https://www.gamedev.net/tutorials/programming/general-and-gameplay-programming/unity-isometric-tilemap-tutorials-r5509/

TOWER DEFENSE GAME

- https://www.raywenderlich.com/269-how-to-create-a-tower-defense-game-in-unity-part-1



MATCH 3 GAME

- https://www.youtube.com/watch?v=cqJ5b5aFo5U