Intro to Unity!

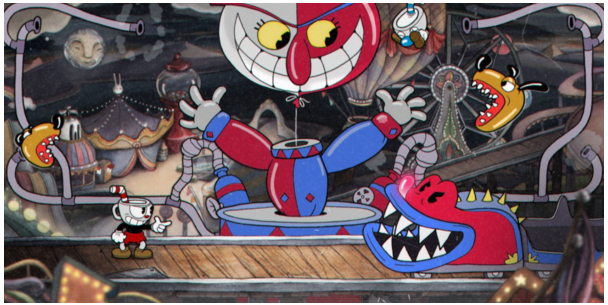
Orbital Mission Control #2
26 May 2018

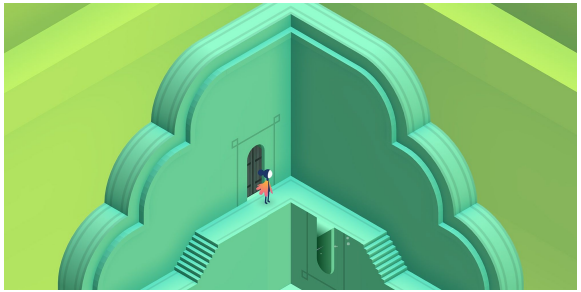
What is Unity

Unity is a cross-platform game engine developed by Unity Technologies, which is primarily used to develop both three-dimensional and two-dimensional video games and simulations for computers, consoles, and mobile devices.

Why Unity?

- Easy to learn
- Great documentation
- Mostly free
- Wide platform support

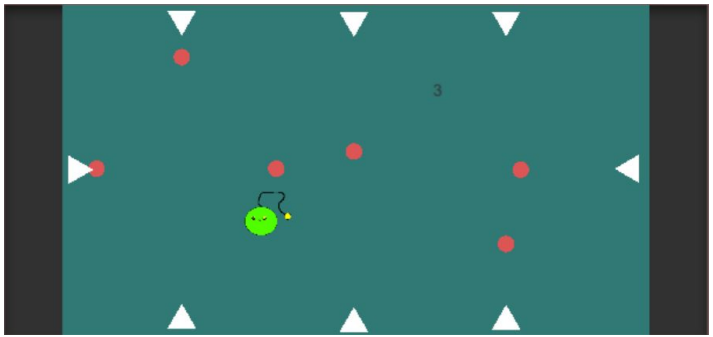




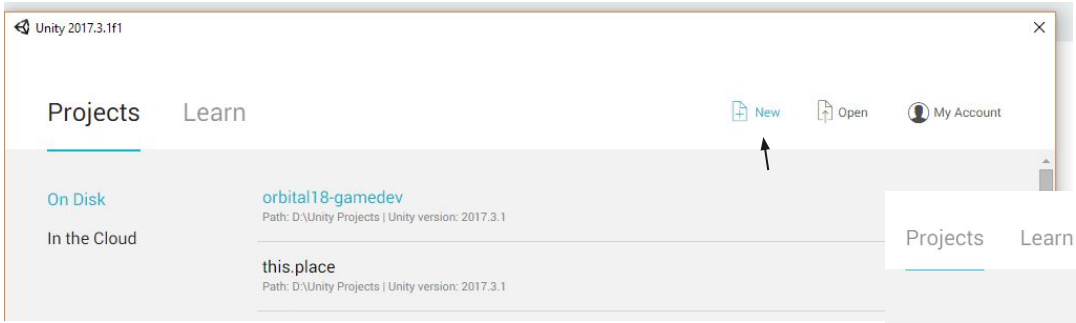
https://unity3d.com/games-made-with-unity

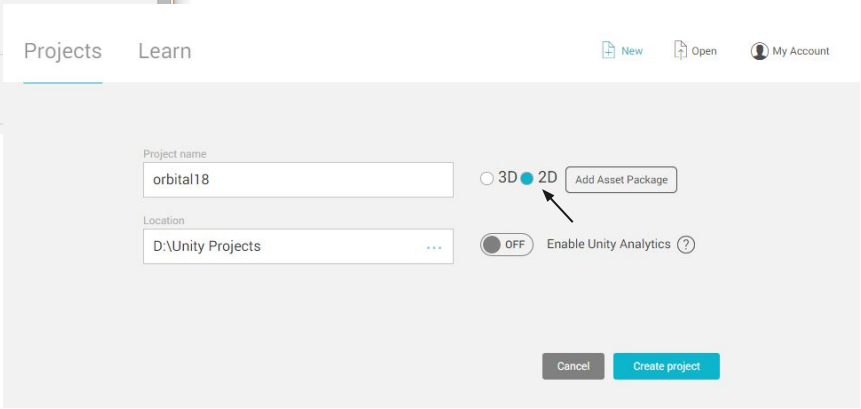
How - Lets Begin!

- Make a simple game
- Introduce the basics of making a game
- Some additional unity features if we have time



Creating a project



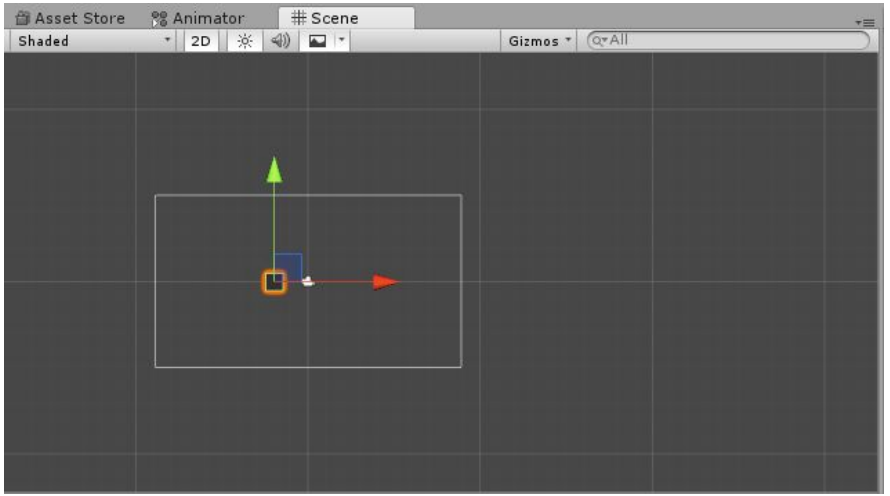


GameObject

- Every object in your game is a GameObject
- Add behaviour to a GameObject by attaching components like
  - Visual - sprites, effects
  - Physics - gravity, collision
  - Audio - sounds
  - Additional behaviour through scripting
- A GameObject can contain another GameObject (parent-child relationship)

Scene

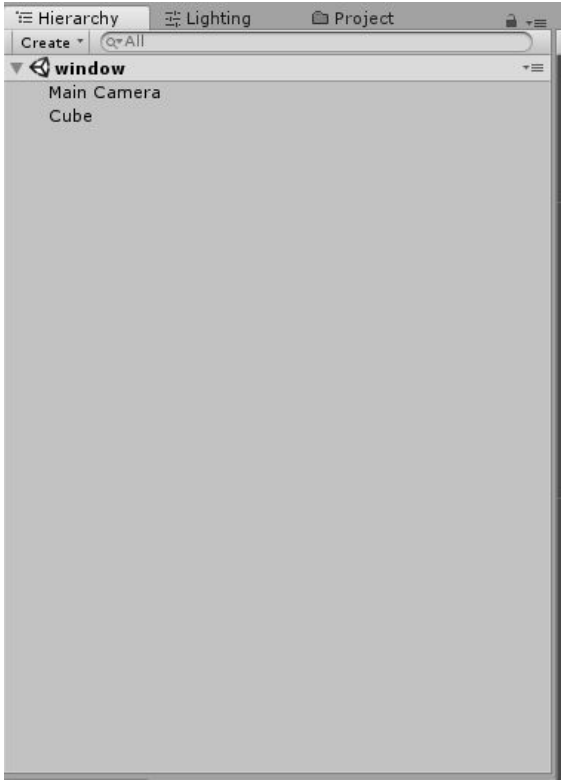
- Collection of GameObjects
- Anything in the game should be in the scene



Scene view

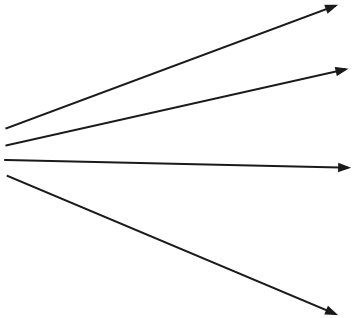
Hierarchy

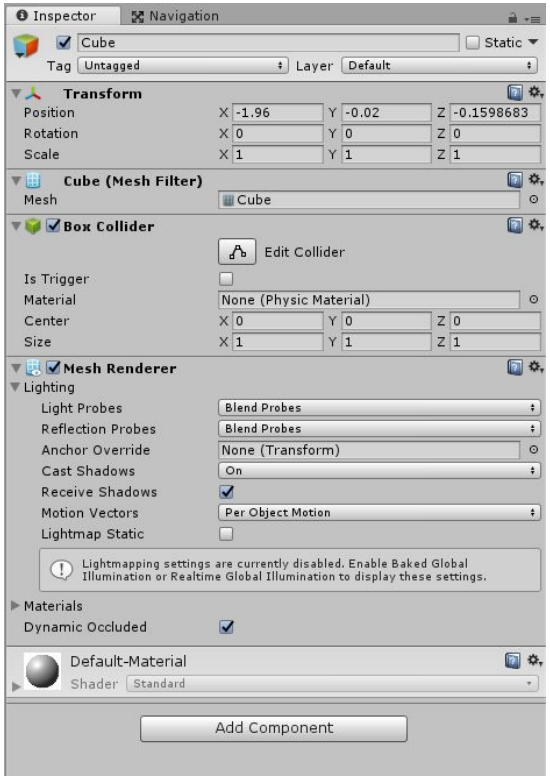
- Contains all GameObjects in a Scene



Inspector

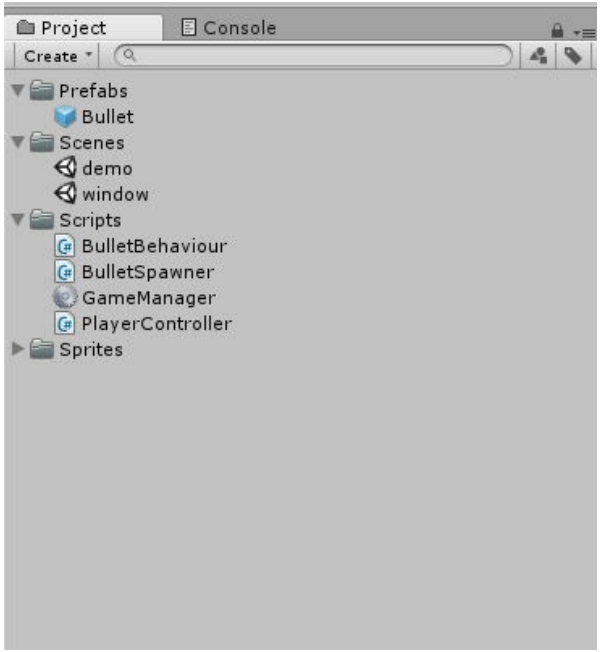
- Information of selected GameObject
- View components attached to GameObject



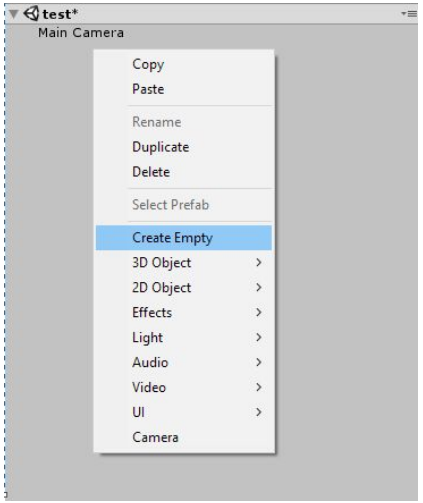


Project

- Contains all files in the project
- Does not display .meta files
  - However, these files are important - don't delete!
  - Contains meta info like object references etc.

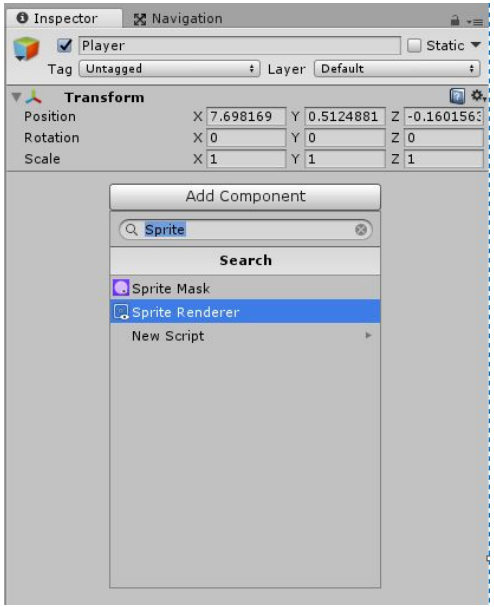


Creating a Player GameObject

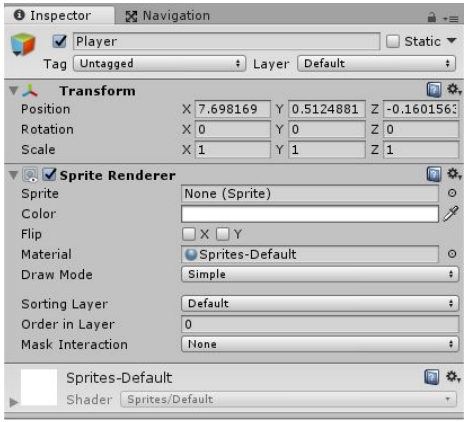


Create empty GameObject in the Hierarchy

Rename to Player

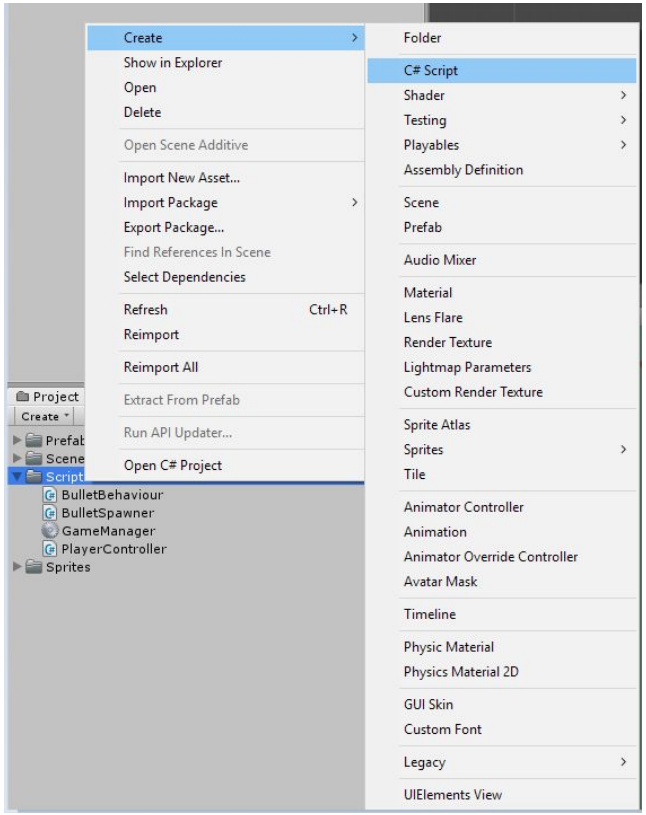


Add Sprite Renderer Component



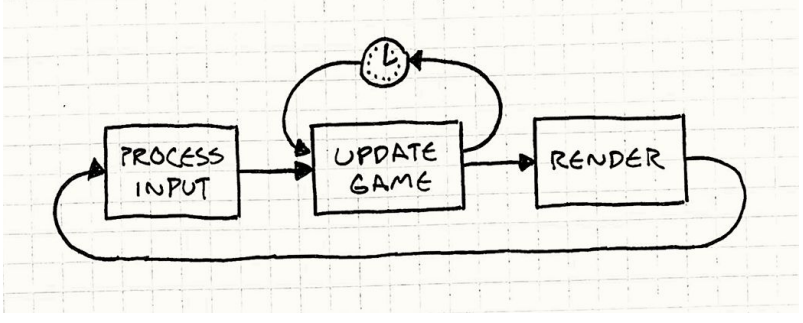
PlayerController

- Create c# script
- This script will handle
  - Reading of Input
  - Translating the Player



Scripting in Unity

- Can be attached to GameObjects as components
- Scripts inherit from MonoBehaviour
  - Specific functions declared in script will be called by the game engine
  - **Update: Update** is called once per frame
  - **Start:** Start is called before the first frame update
  - Read more: https://docs.unity3d.com/Manual/ExecutionOrder.html



PlayerController

```
public class PlayerController : MonoBehaviour {
    float speed = 1f;

    void Update () {
        Vector2 dir = Vector2.zero;

        if (Input.GetKey(KeyCode.W)) {
            dir += Vector2.up; // same as new Vector2(0, 1);
        }

        if (Input.GetKey(KeyCode.S)) {
            dir += Vector2.down; // same as new Vector2(0, -1);
        }

        if (Input.GetKey(KeyCode.A)) {
            dir += Vector2.left; // same as new Vector2(-1, 0);
        }

        if (Input.GetKey(KeyCode.D)) {
            dir += Vector2.right; // same as new Vector2(1, 0);
        }

        dir = dir.normalized; // ensure direction is a normal vector
        Vector2 dist = dir * speed * Time.deltaTime;
        transform.Translate(dist);
    }
}
```

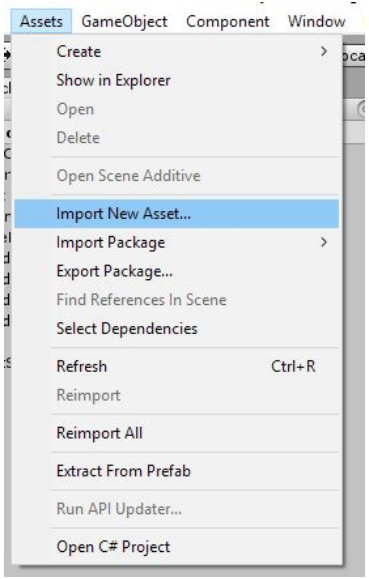
- Each frame we check for inputs to calculate the desired direction of motion (dir)
- Next we calculate the translate amount (dist)
- Time.deltaTime
    - The time in seconds it took to complete the last frame
- transform
    - transform component found in the GameObject this script is attached to
    - transform contains information about position, rotation and scale of the GameObject

Using directives omitted (don't delete them)

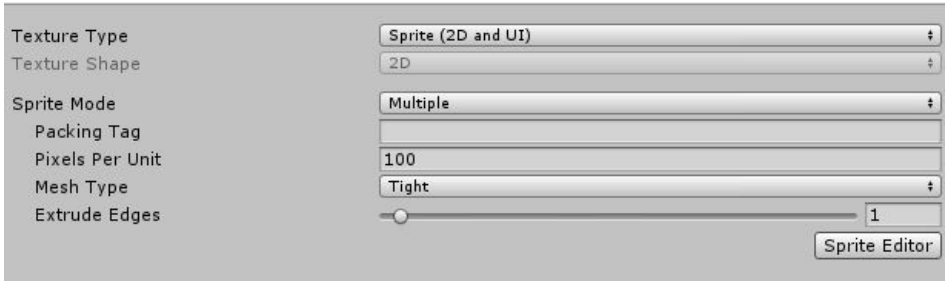
Attach your script

- Attach the script the same way you attach your sprite renderer!

Adding your sprite sheet!

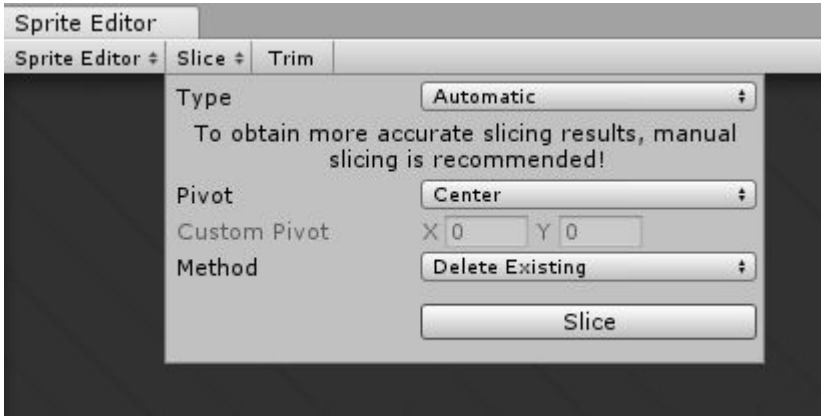


Import your sprite sheet

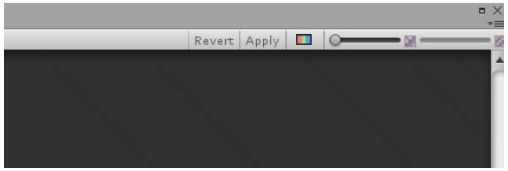


Sprite mode multiple, then click on sprite editor to start slicing

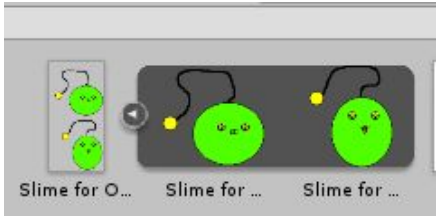
Slicing your sprite sheet



Click slice
IMPORTANT: Make sure the background of the spritesheet is transparent

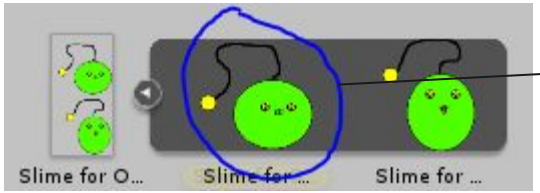


Click apply

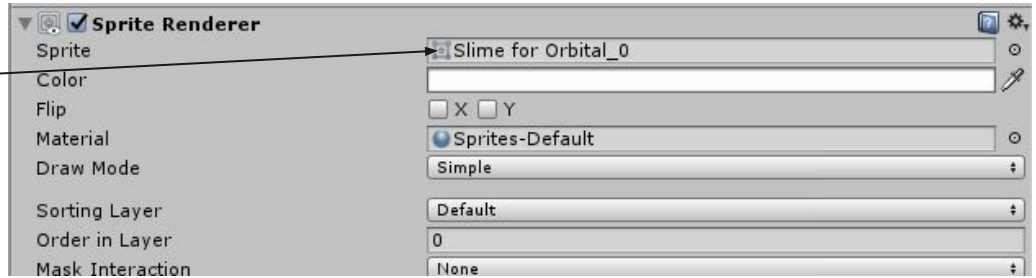


Sprite sheet has been sliced into sprites!

Attach sprite to sprite renderer



Drag one sprite into sprite renderer in your player



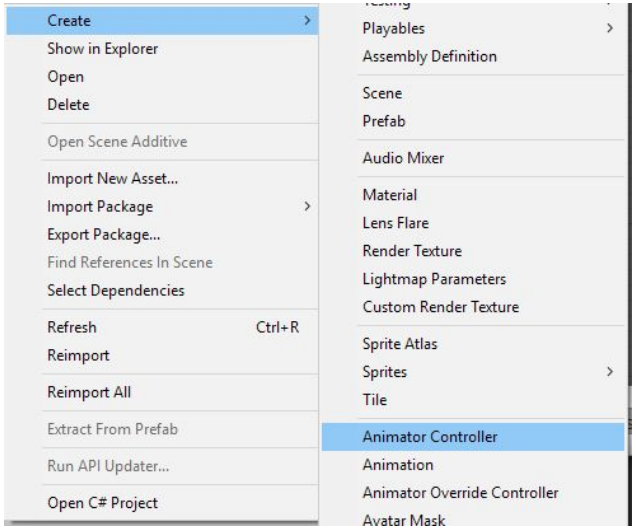
Drag it into the "Sprite field"

Flipping the sprite

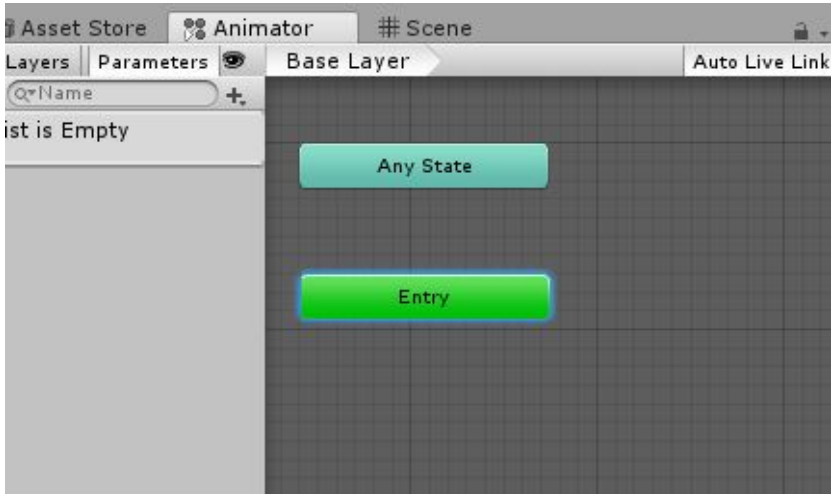
```
public class PlayerController : MonoBehaviour {
    public float speed = 1f;
    private SpriteRenderer sprite;

    void Start () {
        sprite = GetComponent<SpriteRenderer>();
    }

    void Update () {
        //...
        if (Input.GetKey(KeyCode.A)) {
            dir += Vector2.left; // same as new Vector2(-1, 0);
            sprite.flipX = true;
        }

        if (Input.GetKey(KeyCode.D)) {
            dir += Vector2.right; // same as new Vector2(1, 0);
            sprite.flipX = false;
        }

       //...
    }
}
```

- GetComponent grabs the SpriteRenderer component in the GameObject (Player)
    - IMPORTANT: do GetComponents only in Start as it is an expensive method
    - There is more than one way to grab the reference of the SpriteRenderer
- Since our sprite is facing right, when we move left, we flip it horizontally

Adding animations

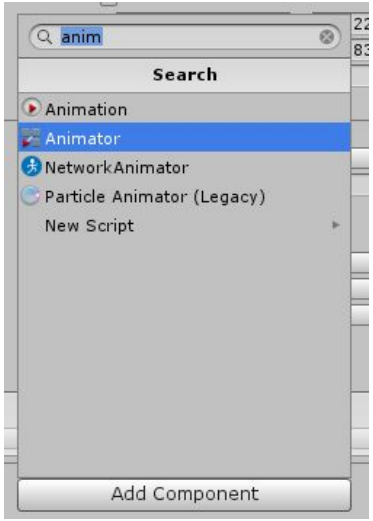


Create in project view a new Animator Controller and call it Player

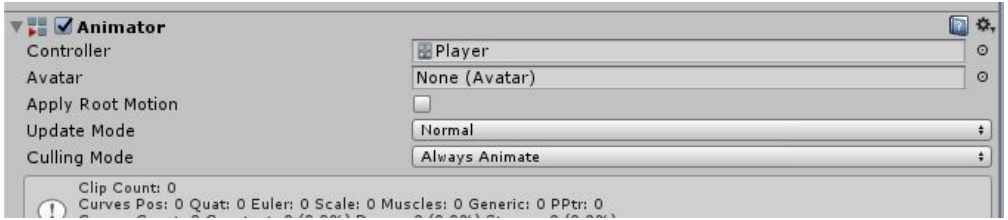


Double click it and you will see a state machine

Attach animation controller to Player

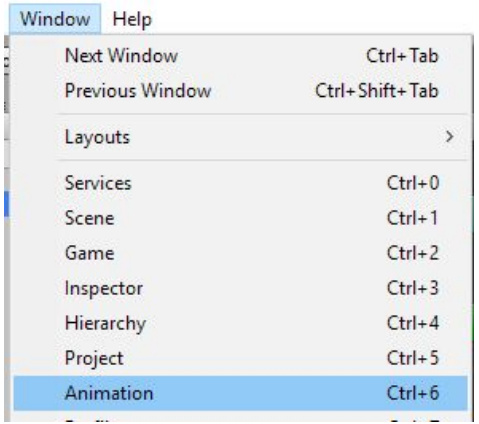


Add animator component onto Player

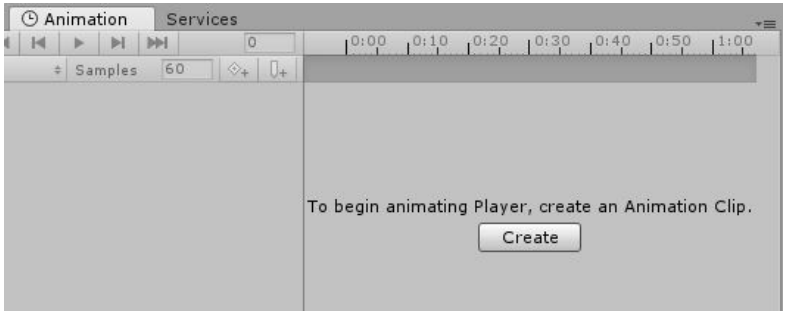


Drag your animator controller into 'Controller'

Create a new animation

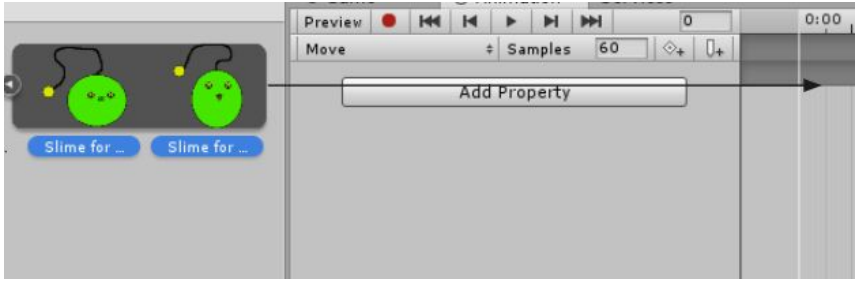


Open the animation window



Select Player in the Hierachy - important
And create a new clip called Move

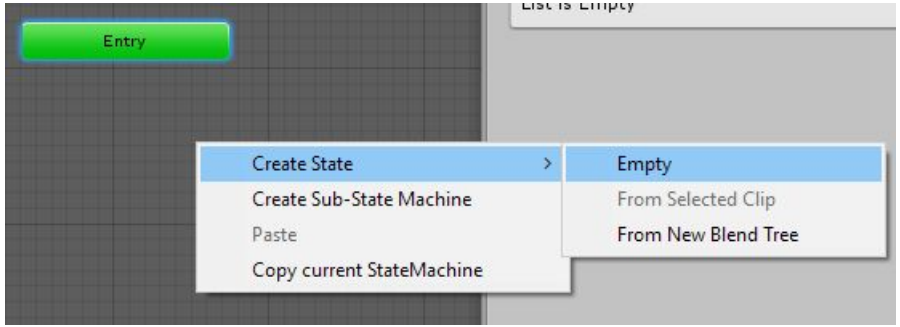
Create a new animation



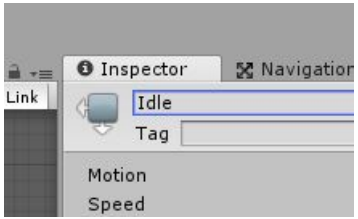
- This creates a new animation
- Click play and view the animation in the Scene view
- Adjust the number next to Samples to adjust the speed of animation
- If you have only two frames, use a lower sample like 10

Drag your sprites into the animation
IMPORTANT: Make sure Player is still selected in the Hierarchy

Adding an Idle state

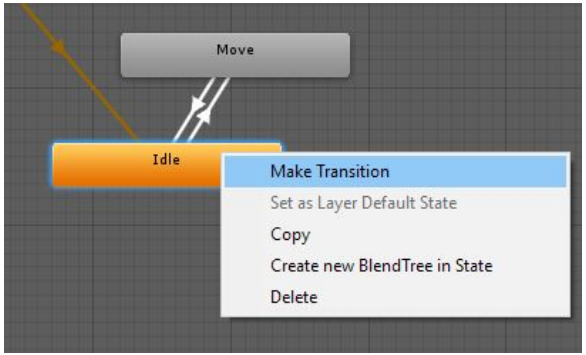


Create a new Empty state

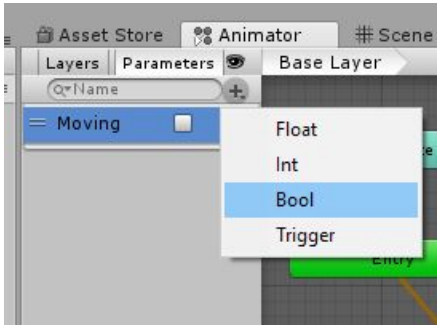


Name it Idle

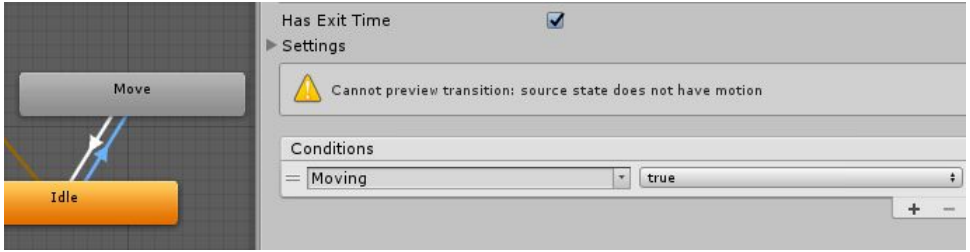
Set up state transition



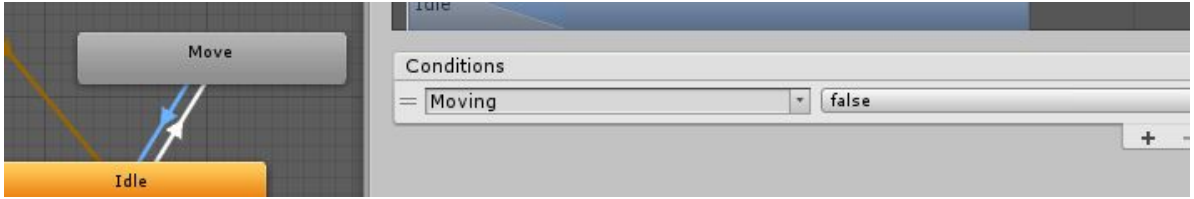
Select your Player Animator Controller make transitions between Move and Idle



Create a bool parameter, call it Moving

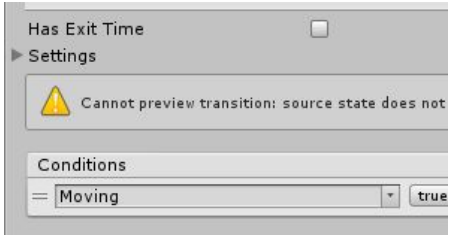


For the state going to Move, set a condition for Moving to be True



For the state going to Idle, set a condition for Moving to be False

- If we set this “Moving” condition true, we will play the “Moving” animation
- Now all we need to do is set this programmatically!
- Also, set ‘Has Exit Time’ to false for both transitions



PlayerController

```
public class PlayerController : MonoBehaviour {
    // ...
    private Animator anim;
    void Start () {
        // ...
        anim = GetComponent<Animator>();
    }
    void Update () {
        // input logic
        dir = dir.normalized; // ensure direction is a normal vector

        if (dir.magnitude > 0) {
            anim.SetBool("Moving", true);
        } else {
            anim.SetBool("Moving", false);
        }

        Vector2 dist = dir * speed * Time.deltaTime;
        transform.Translate(dist);
    }
}
```

- Go to the Move animation if our direction vector has length greater than 0
- Note also that if we are in the Move State and we “Moving” to true our animation won’t restart
    - The outgoing transition only responds when “Moving” is false
    - This is a behaviour of state machine

Player Movement v2 - Unity Physics

- Box Collider 2D
- Rigidbody 2D
- Adjust velocity instead of translating
- https://docs.unity3d.com/Manual/ExecutionOrder.html

Bullets

- Bullets move in one direction only
- The plan:
    a. Create a Bullet GameObject in the hierarchy
    b. Create a script BulletBehaviour
    c. Attach the script to the GameObject
    d. Write code to make it move
        - For now we can just move it downwards every frame (Vector2.down)
- Try to implement this!

BulletBehaviour

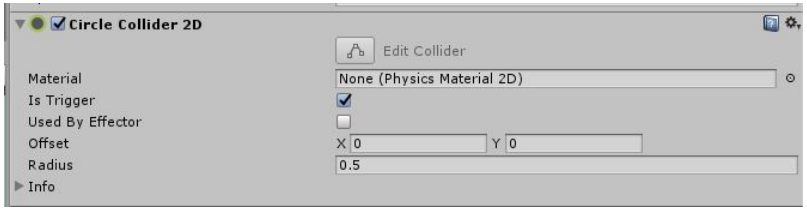
```
public class BulletBehaviour : MonoBehaviour {

    public float speed = 4;

    void Update()
    {
        Vector2 dist = Vector2.down * speed * Time.deltaTime;
        transform.Translate(dist);
    }
}
```

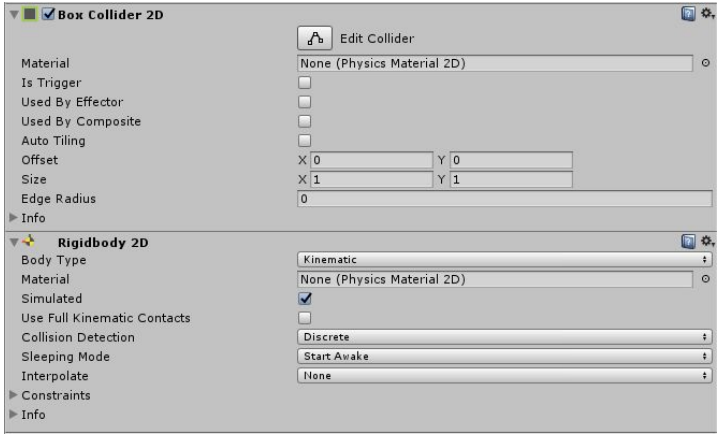
Having a public variable in the script exposes the variable in the inspector!

- This allows you to tweak any variables without touching any code!
- Change speed in PlayerController to public to make gameplay testing easier!

Adding collision

- For objects to collide, we need to use colliders
- For bullets - add CircleCollider 2D
  - Set isTrigger in CircleCollider to true
  - Trigger allows and detects object which pass into the collider
- For the player - add BoxCollider 2D and Rigidbody2D
  - Rigidbody lets us use trigger events like OnTrigger2D (next slide)





Read:
https://docs.unity3d.com/ScriptReference/Rigidbody2D.html

OnTriggerEnter2D

```
public class BulletBehaviour : MonoBehaviour {

    public float speed = 4;


    void Update()
    {
        Vector2 dist = Vector2.down * speed * Time.deltaTime;
        transform.Translate(dist);
    }

    void OnTriggerEnter2D(Collider2D collider) {
        Destroy(gameObject);
        Destroy(collider.gameObject);
    }
}
```

gameObject is an inherited member of MonoBehaviour (which inherits from Behaviour which inherits from Component)

https://docs.unity3d.com/ScriptReference/Component-gameObject.html

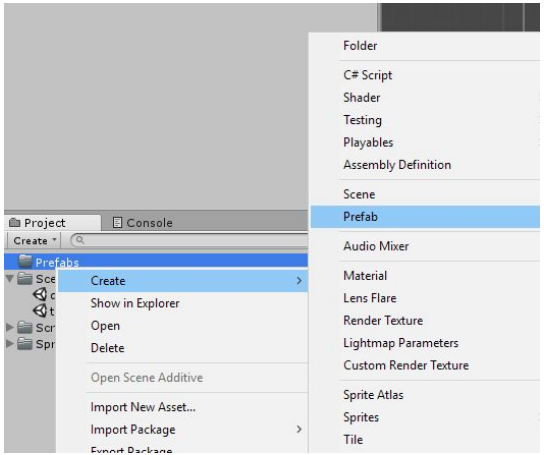
- OnTriggerEnter2D is a function that is called when another collider enters trigger
- In this case it will be the player's collider
- Here if the player enters a bullet's collider, we destroy the bullet GameObject as well as the player
- Common mistake: Destroy(this)
    - Only destroys the component and not the GameObject

Spawner

- Every set amount of time we spawn a bullet a spawn location
- We also want to specify a list of locations and choose a random one
- Create a Spawner GameObject in the scene
- Create a BulletSpawner script
- Questions:
    - How do we spawn a copy of an object?

Prefabs

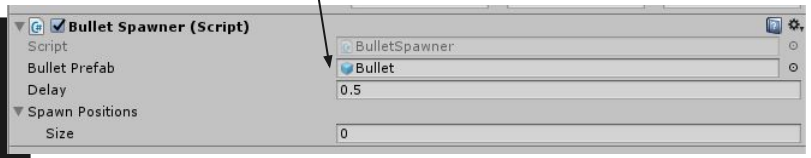
- **Prefab** asset type that allows you to store a GameObject object complete with components and properties
- Basically a template GameObject
- Create a prefab
- Drag the bullet game object from the hierarchy into the prefab
    - This copies the entire GameObject into the prefab
- We can reference this prefab and Instantiate it!



Spawn logic

```
public class BulletSpawner : MonoBehaviour {

    public GameObject BulletPrefab;
    public float Delay = 0.5f;
    public List<Transform> spawnPositions = new List<Transform>();
    float timeStamp;

    // Update is called once per frame
    void Update () {
        if (Time.time < timeStamp) {
            return;
        }

        int randomIndex = Random.Range(0, spawnPositions.Count);
        Transform spawnPosition = spawnPositions[randomIndex];
        Instantiate(BulletPrefab, spawnPosition.position, spawnPosition.rotation);
        timeStamp = Time.time + Delay;
    }
}
```

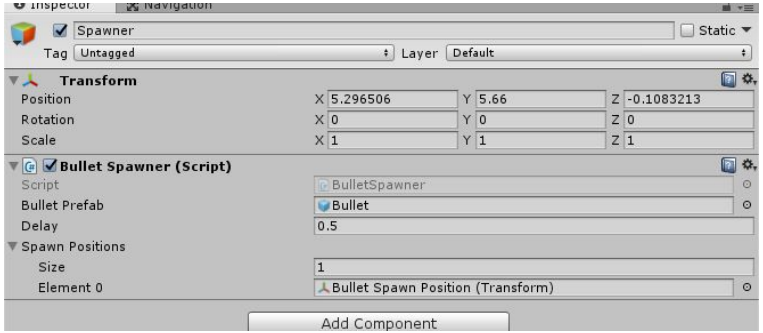
Bullet Prefab found in Project Window



- timeStamp determines when is the next time to fire a bullet
- From list of transforms, we pick a random one
- spawn the bullet at its position with its rotation

Creating spawn locations

- Create an empty GameObject and call it Bullet Spawn Position
- Drag the GameObject into List in the BulletSpawner script in the Spawner GameObject
- Make more Spawn Position!
- Try changing the rotation of a Spawn Position GameObject and observe how the bullet moves!
    - Do you know why it behaves that way?



Even though you are dragging a GameObject into a List<Transform>, Unity is smart enough to pass in the Transform component. This also works for scripts!

Some Collision problems

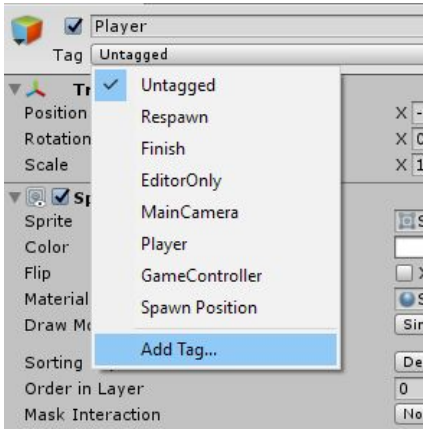
- Bullets don’t despawn if it leaves the screen

Despawn Bullets

- Create 4 colliders around the screen to destroy any bullets
  - Make sure these bounding colliders have a Rigidbody2D component
- Do you see any problems?
  - The first bullet will destroy the bounding box
  - We need a way to differentiate player from other entities!

Tags

- GameObjects can be given a tag
- It lets you identify an object for scripting purposes
- Give Player the “Player” tag
- You can also add your own tags



BulletBehaviour

```
public class BulletBehaviour : MonoBehaviour {

    public float speed = 4;


    void Update()
    {
        Vector2 dist = Vector2.down * speed * Time.deltaTime;
        transform.Translate(dist);
    }

    void OnTriggerEnter2D(Collider2D collider) {
        Destroy(gameObject);

        if (collider.CompareTag("Player")) {
            Destroy(collider.gameObject);
        }
    }
}
```

- Only destroy the other collider if its the Player
- Note: we can also do collider.tag == “Player”
    - but it’s better to use to CompareTag as it is more optimized
    - Read: https://answers.unity.com/questions/200820/is-comparetag-better-than-gameobjecttag-performanc.html

Game Manager

- Manages game related logic
  - Example: increasing difficulty of game
- The plan:
  - Increase level periodically
  - The greater the level
    - Faster respawn
    - Faster bullet speed

GameManager

```
public class GameManager : MonoBehaviour {
    public int Level = 1;
    public float LevelTime = 5f;
    private float timeStamp

    // Update is called once per frame
    void Update () {
       if (Time.time < timeStamp) {
         return;
       }
       timeStamp = Time.time + LevelTime;
       Level++;
    }
}
```

- Every 5s, increase game level
- Question:
  - Where to adjust bullet speed and respawn time
  - How to get level information?

Singleton Pattern

```
public class GameManager : MonoBehaviour {
    // other logic...
    private static GameManager instance;
    void Start () {
        if (instance == null) {
            instance = this;
            return;
        }
        Destroy(this);
    }

    public static GameManager GetInstance() {
        return instance;
    }

}
```

- Common way to handle managers
- First instance of GameManager will be the **only** instance which can be accessed via getter
- GetInstance() will always call the only instance of GameManager

Spawn Logic

```
public class BulletSpawner : MonoBehaviour {

    public GameObject BulletPrefab;
    public float Delay = 0.5f;
    public List<Transform> spawnPositions = new List<Transform>();
    float timeStamp;

    // Update is called once per frame
    void Update () {
        if (Time.time < timeStamp) {
            return;
        }

        int randomIndex = Random.Range(0, spawnPositions.Count);
        Transform spawnPosition = spawnPositions[randomIndex];
        Instantiate(BulletPrefab, spawnPosition.position, spawnPosition.rotation);
        timeStamp = Time.time + Delay / GameManager.GetInstance().Level;

    }
}
```

- GameManager instance can be accessed from spawner
- Use level to adjust delay logic

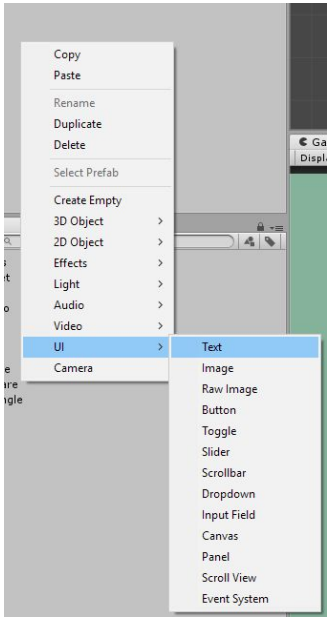
BulletBehaviour

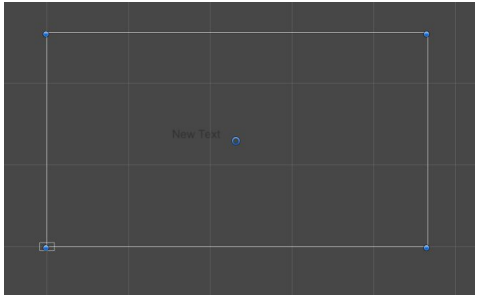
```
public class BulletBehaviour : MonoBehaviour {

    public float speed = 4;

    void Start()
    {
        speed += 2 * (GameManager.GetInstance().Level - 1);
    }

    //... other

}
```

- Every new bullet spawns with a speed specific to Game Level

UI

- Display text on screen for level information
- Create a UI>Text GameObject
- Name it Text
- Zoom out so that you can see the entire canvas
    - Treat the canvas as the screen
    - Position of the UI will be mapped accordingly
    - Look at ‘Game View’ if in doubt

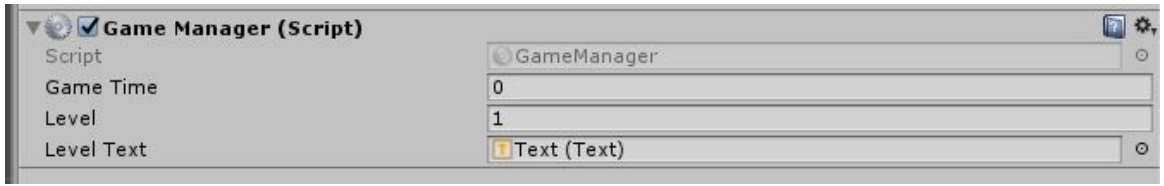




GameManager

```
using UnityEngine.UI;
public class GameManager : MonoBehaviour {
    private static GameManager instance;
    public Text LevelText;

    void Start () {
        if (instance == null) {
            instance = this;
            LevelText.text = Level.ToString();
            return;
        }
        Destroy(this);
    }

    // Update is called once per frame
    void Update () {
        GameTime += Time.deltaTime;
        if (GameTime > nextLevelTime) {
            Level++;
            LevelText.text = Level.ToString();
            nextLevelTime += 5f;
        }
    }
}
```

- At the start of the game, set the text to current level
- Update text when level changes
- Important:
    - Import UnityEngine.UI
- Now, drag the text GameObject into the LevelText field in the GameManager component



Other areas to explore

- Particle systems/emitters
- Audio
- Post processing
- Scriptable Objects
- Multiplayer
- Profiling

Useful resources

- Unity
  - Sebastian Lague https://www.youtube.com/user/Cercopithecan
  - Brackeys https://www.youtube.com/user/Brackeys
- General
  - GDC https://www.youtube.com/channel/UC0JB7TSe49lg56u6qH8y_MQ
- Free Assets
  - Itch io https://itch.io/game-assets
  - Kenney http://kenney.nl/

Tips to survive Orbital

- Set realistic goals
  - MMOs, HalfLife 3, Overwatch 2 are not made in 3 months by a 2 man team
- Focus on a few core mechanics
  - Eg: core mechanic of Mario is platforming
- Learn how to use git
  - Set up for unity: https://robots.thoughtbot.com/how-to-git-with-unity
- Have fun!

NUS GDG

- Game development workshops during the Semesters
  - every Thursday 6pm - 830pm
- GameCraft 2018
  - 24 hour Game Jam
  - Prizes to be won
  - December 2018