PROJECT – ROCKET JUMP

Game Design: -

- **Player experience: -**

  The player should be Skillful.

- **Core Mechanics: -**

  Skillfully fly a spaceship and avoid environmental hazards.

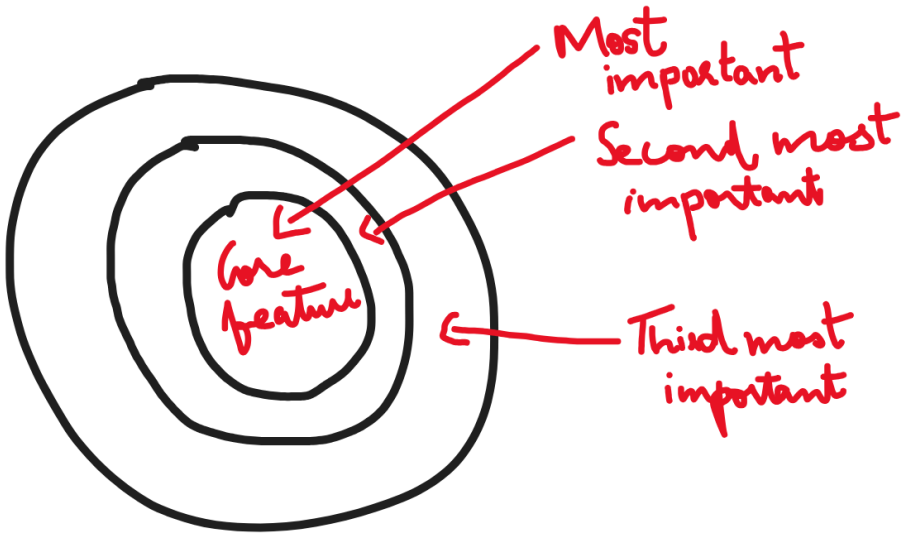
- **Core Game Loop: -**

  Go from A to B to complete the level, then progress to the next level.

- **Game Theme: -**

  ➔ Environmental early-generation spacecraft.
  ➔ On an unknown planet, trying to escape.

- **Onion Design: -**

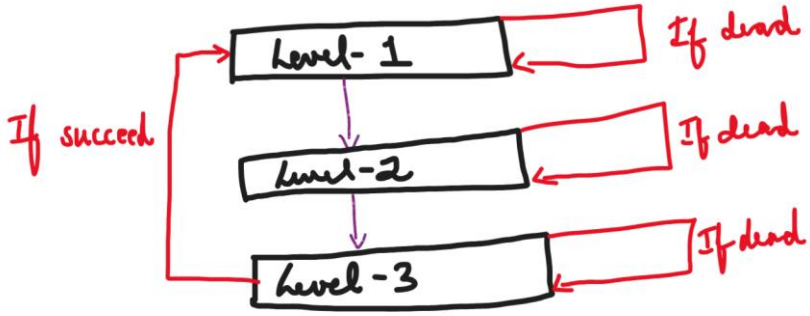


Note: Do the most important feature first.

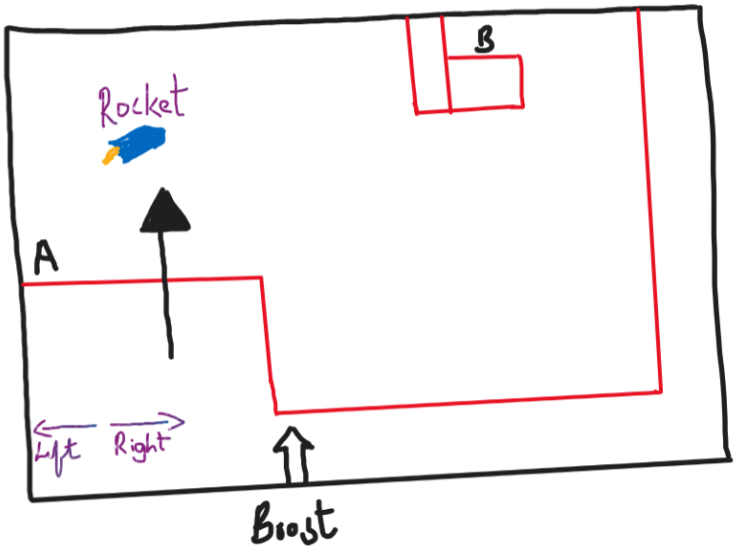
Important features in the game: -

- ➔ **Movement/Flying**
- ➔ **World Collision**
- ➔ **Level Progression**

- **Game Flow: -**

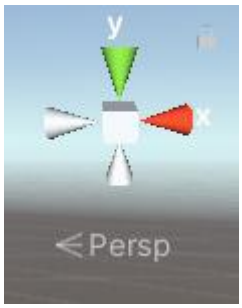


- **Sample Game Design Plan: -**

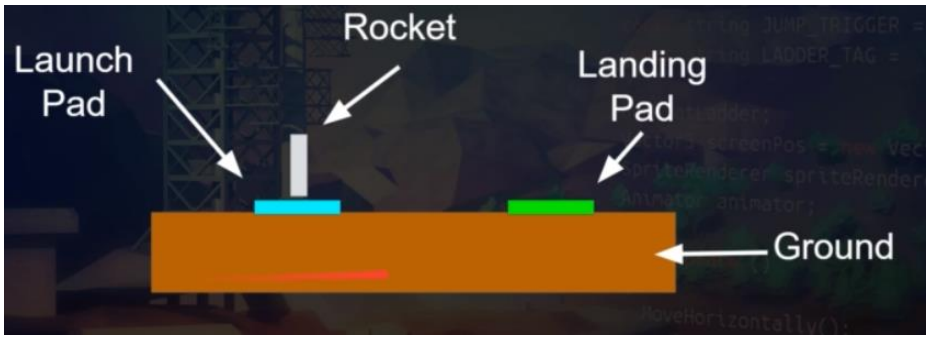


➔ **Unity Units: -**

Make sure that you have this one.



- **Building the starting piece.**



Remember these things: -

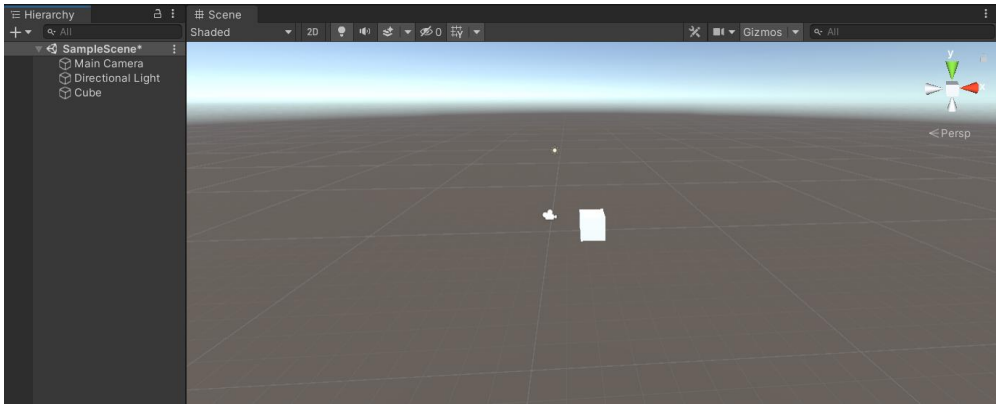
+x = right

+y = up

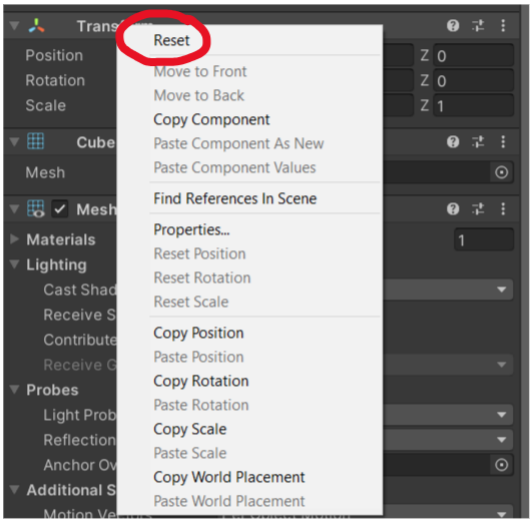
+z = left.

Unity unit Scale is 1 which is generic.

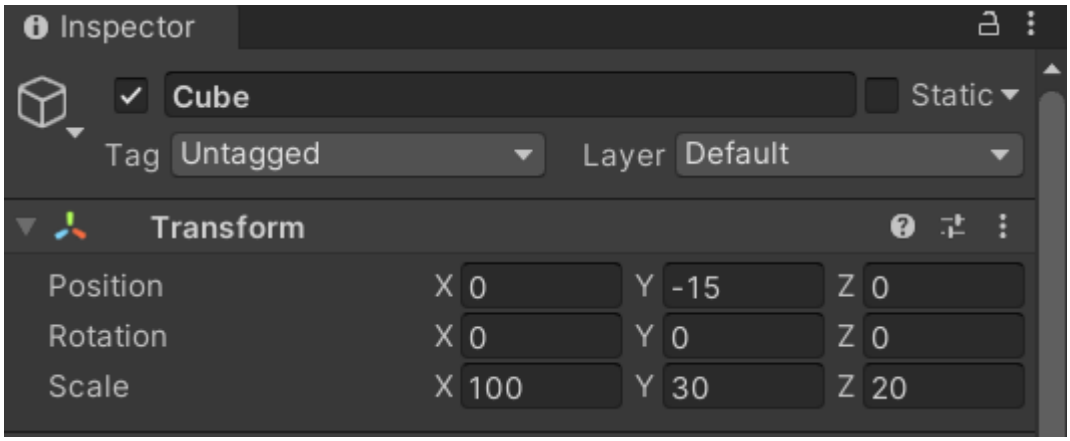
Step 1: Create a 3D Object Cube.



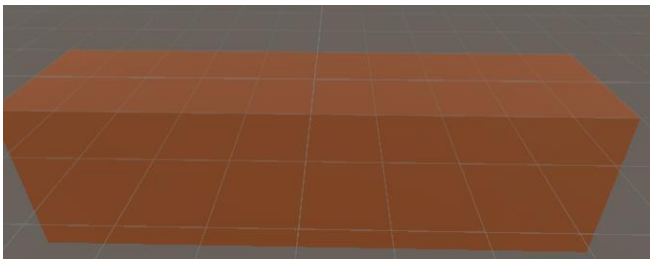
Note: To keep it in the center of the world, we must Reset the position in Transform from the Inspector.



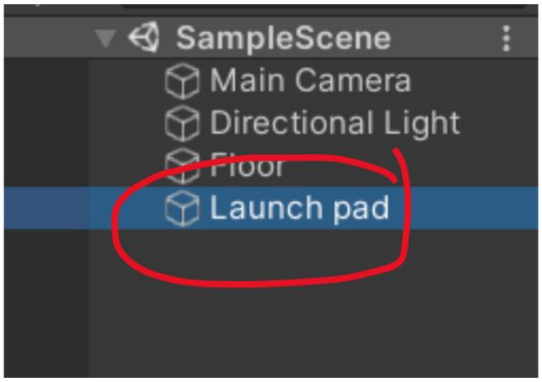
Keep the cube size at these values for now.

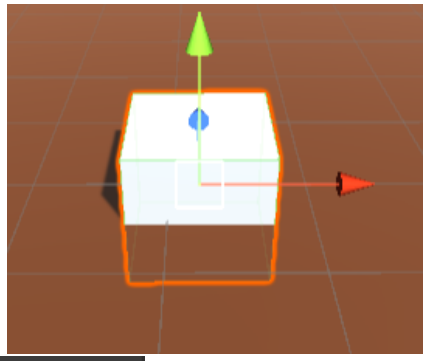


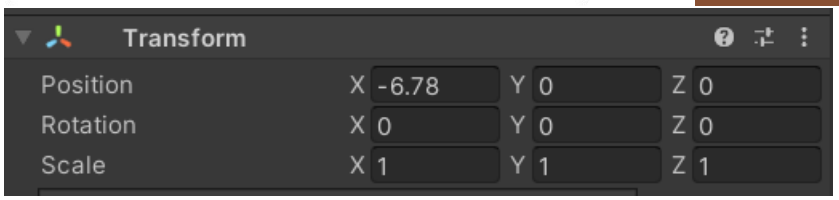
Step 2: Create a Material and attach it to the cube ground object.



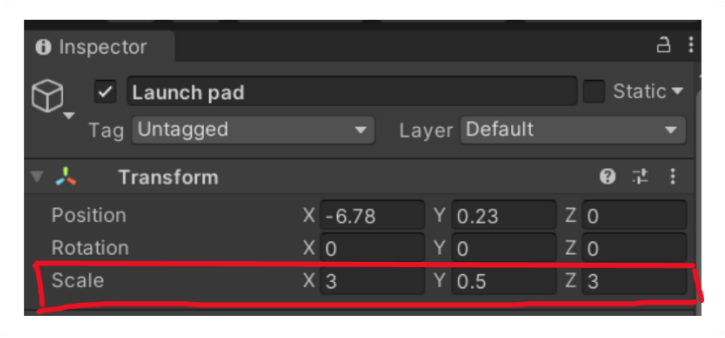
Step 3: Create a Launch pad.

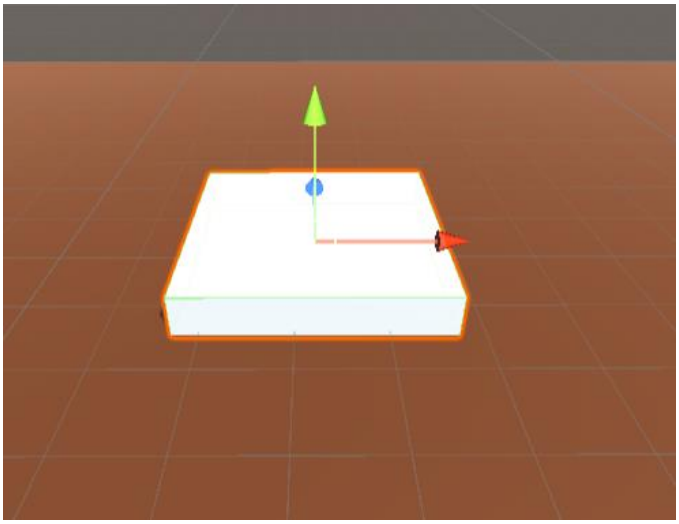




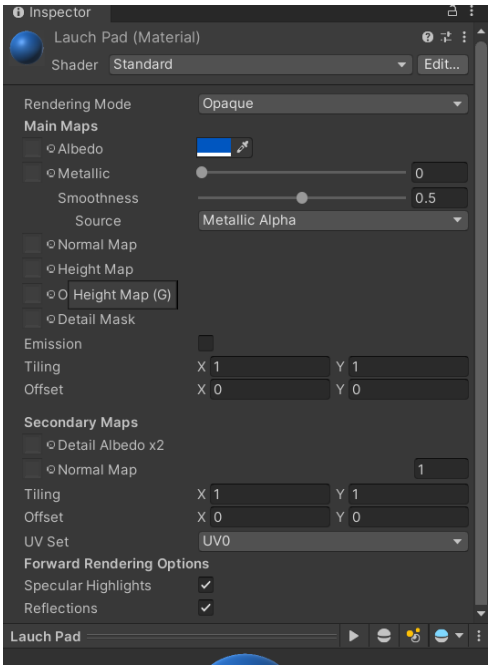


Step 4: Scale to the size: -



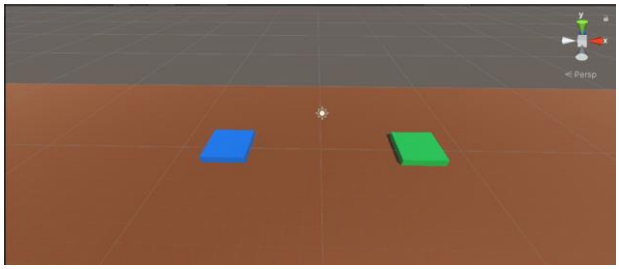


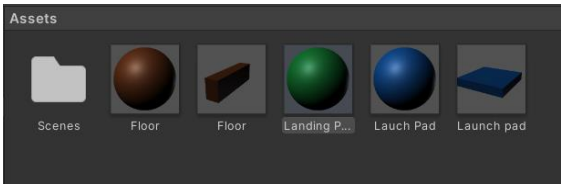
Step 5: Create material for the launch pad.



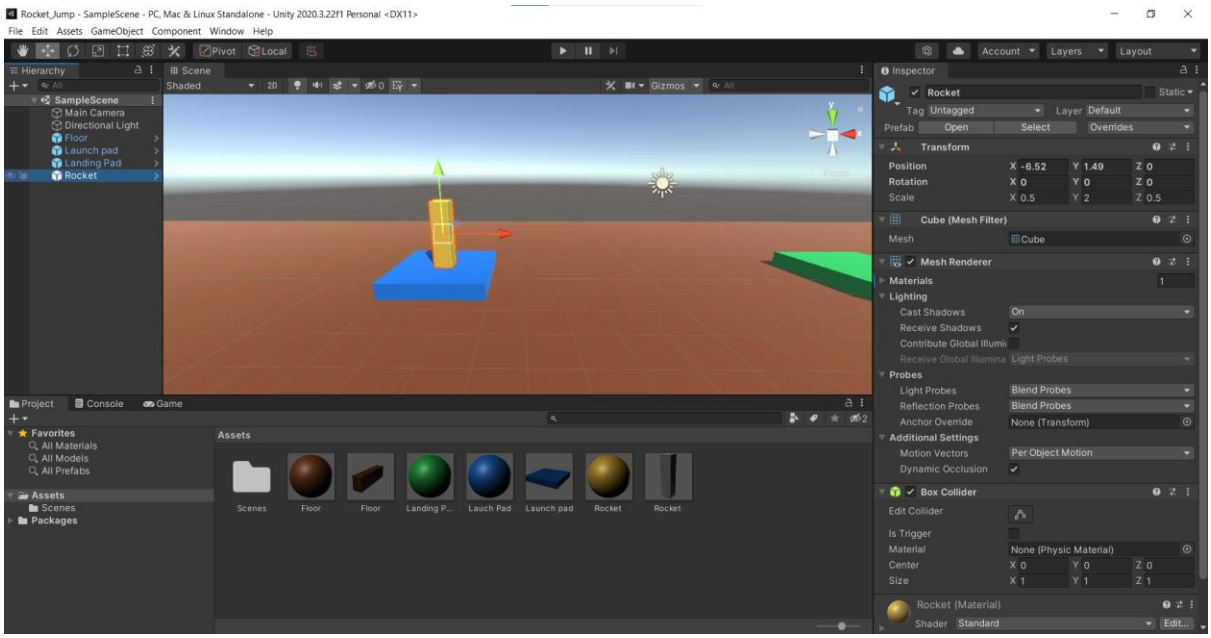
Step 6: Make Floor and Launch Pad as prefabs by dragging them into the Asset Folder.

Step 7: Duplicate the Launch Pad and rename it as Landing Pad, at the same time duplicate the material as well.





Step 8: Create a Rocket.



Note: In case you want to rename a scene, go to the Scenes Folder and change the scene name to Sandbox.

CLASSES:-

```
public class Movement : MonoBehaviour
```

Movement is derived from MonoBehaviour. This concept is called inheritance.

Monobehaviour itself is a class that has a lot of information inside it.

- **Classes are used to organize code.**
- **Classes are "containers" for variables and methods that allow us to group different things together.**
- **Usually, we aim for a class to do one main thing and not multiple things.**
  - ➔ **Easier to read our code.**
  - ➔ **Easier to fix issues.**
  - ➔ **Easier to have multiple people work on a project.**
  - ➔ **We tend to create a new class each time we create a new script and have that script responsible for one thing. Examples include.**

    Movement, CollisionHandler, Shooting, Score and EnemyAI.

  Note: Too much of one class is a mess.

  To avoid messy code, we use the concept of Encapsulation.

  Wrapping data into a single unit is called Encapsulation.

- **In unity Classes are already Built for us.**
  - **I) Unity has many Classes already created for us.**
  - **II) By accessing the class, we access its contents.**
  - **III) In our code we write the class name and then use the dot operator to access things in the class.**

    Eg is given in the photo.

```
ClassName.MethodName()
```

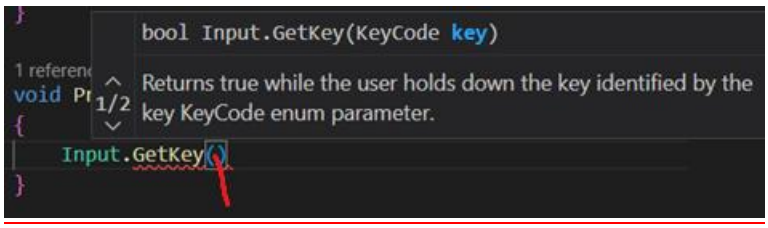
BASIC INPUT BINDING: -

The most important thing to do is always to organize code and materials in a folder.

GetKey	Returns true while the user holds down the key identified by name.
GetKeyDown	Returns true during the frame the user starts pressing down the key identified by name.

Do refer to this once for more information in the documentation.

https://docs.unity3d.com/2020.1/Documentation/ScriptReference/Input



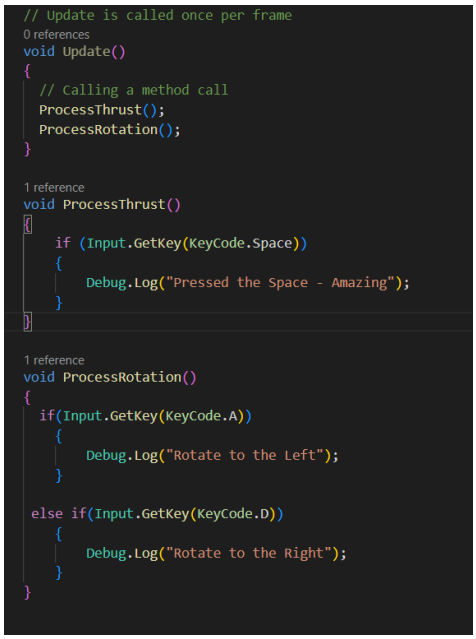
Inside the braces press ctrl + shift + space which shows the signature options.

```
void ProcessInput()
{
    Input.GetKey("Space");
}
```

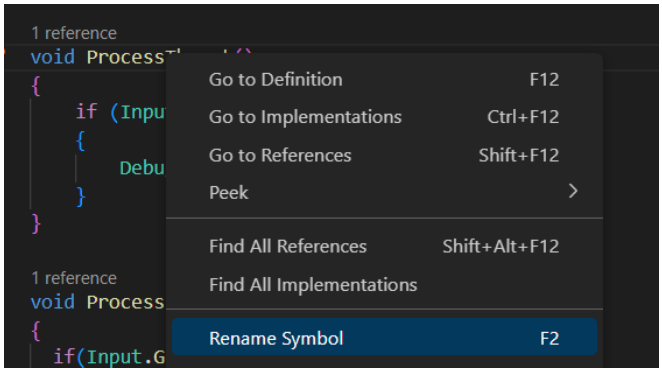
Maximum avoid string references as much as possible.

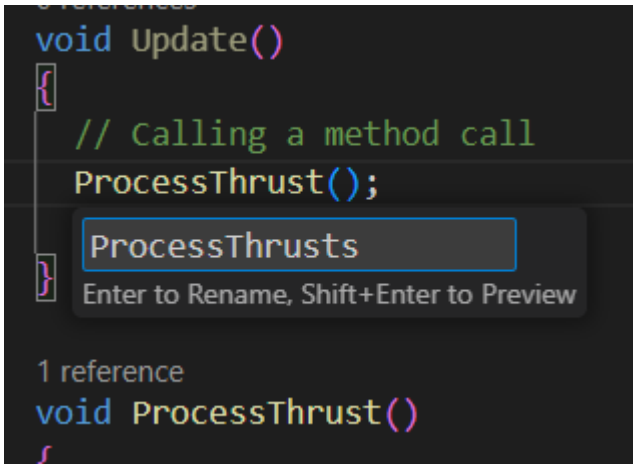
Sample code so far as the picture

```
// Update is called once per frame
0 references
void Update()
{
  // Calling a method call
  ProcessInput();
}

1 reference
void ProcessInput()
{
    if (Input.GetKey(KeyCode.Space))
    {
        Debug.Log("Pressed the Space - Amazing");
    }
}
```

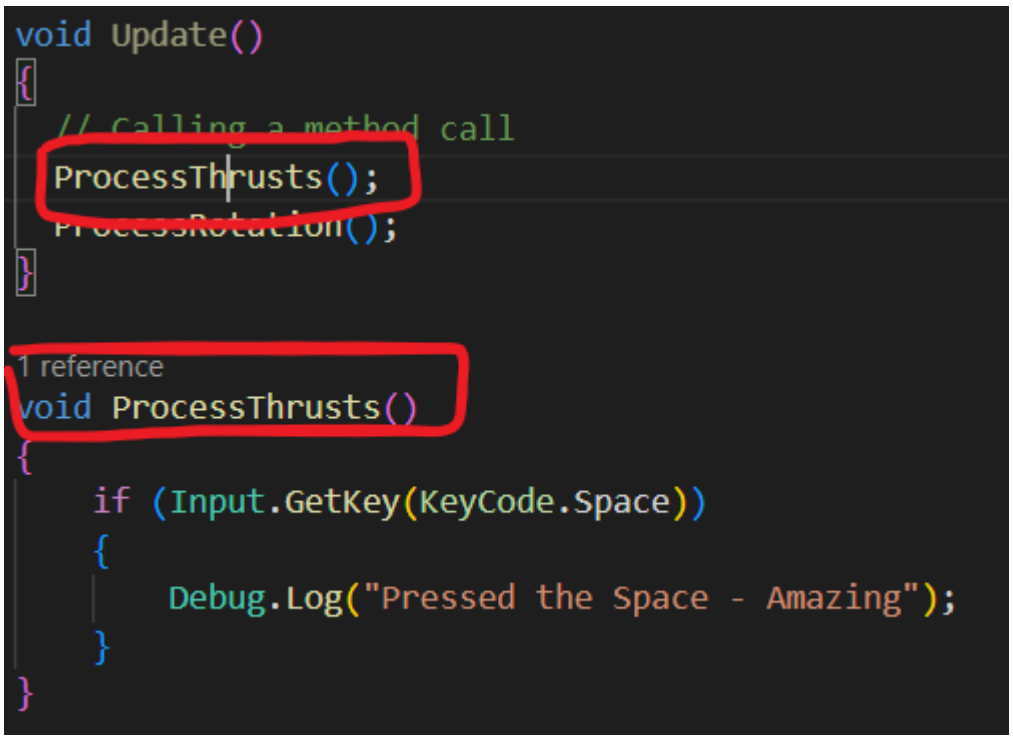


```
// Update is called once per frame
0 references
void Update()
{
  // Calling a method call
  ProcessThrust();
  ProcessRotation();
}

1 reference
void ProcessThrust()
{
    if (Input.GetKey(KeyCode.Space))
    {
        Debug.Log("Pressed the Space - Amazing");
    }
}

1 reference
void ProcessRotation()
{
  if(Input.GetKey(KeyCode.A))
    {
        Debug.Log("Rotate to the Left");
    }

 else if(Input.GetKey(KeyCode.D))
    {
        Debug.Log("Rotate to the Right");
    }
}
```





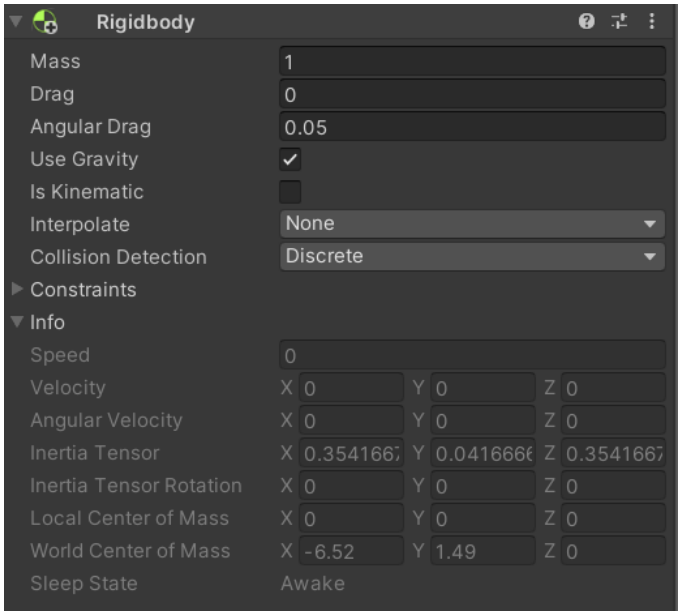
When you see the name gets changed.



ADD RELATIVE FORCE: -

First, we need to add a Rigid body, to apply select component -> Rigid body.

Make sure you enable gravity.

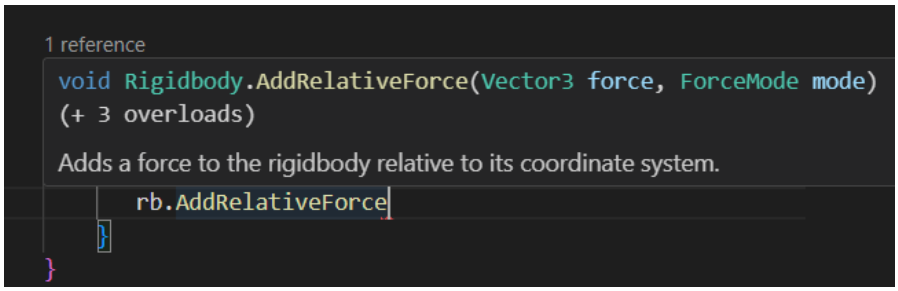


While testing it so far, the gravity is working.

While pressing the space bar we need to add Force? What's the possibility?

Note:-

The reason we use AddRelativeForce.



Basically, vector3 contains X, Y, and Z axes.

But a Vector in a 2D game has only x and y axes.

Vector is both direction and magnitude.

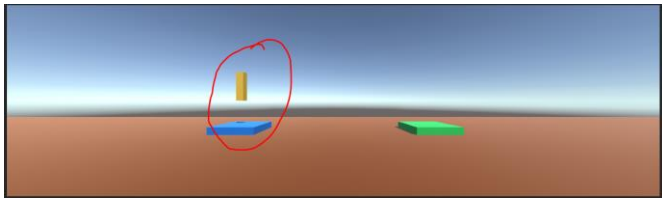
```
ProcessRotation();
'Vector3' is a type, which is not valid in the given
context (CS0119)
struct UnityEngine.Vector3
Representation of 3D vectors and points.
View Problem (Alt+F8)    No quick fixes available
        rb.AddRelativeForce(Vector3)
    }
}
```

Code so far as the photo.

```
0 references
void Start()
{
    // We are getting the rigidbody component.
    // Caching a reference to our component.
    rb = GetComponent<Rigidbody>();
}

// Update is called once per frame
0 references
void Update()
{
  // Calling a method call
  ProcessThrusts();
  ProcessRotation();
}

1 reference
void ProcessThrusts()
{
    if (Input.GetKey(KeyCode.Space))
    {
        rb.AddRelativeForce(Vector3.up * 10);
    }
}
```

Result so far:

The rocket moves up towards the y axis.



VARIABLES FOR THRUSTING: -

Setting up a tuning variable to see how much thrust is applied to the rocket.

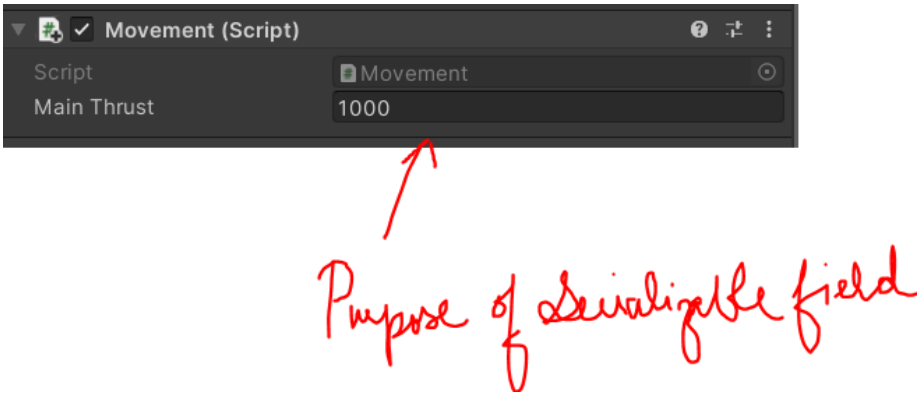
Also make it frame rate independent.

```
// Add a RigidBody variable

1 reference
[SerializeField] float mainThrust = 100;
2 references
Rigidbody rb;

// Start is called before the first frame update
0 references
void Start()
{
    // We are getting the rigidbody component.
    // Caching a reference to our component.
    rb = GetComponent<Rigidbody>();
}

// Update is called once per frame
0 references
void Update()
{
  // Calling a method call
  ProcessThrusts();
  ProcessRotation();
}

1 reference
void ProcessThrusts()
{
    if (Input.GetKey(KeyCode.Space))
    {
        // Adding Time.deltaTime to make it frame independent
        // mainthrust to move an object quickly
        rb.AddRelativeForce(Vector3.up * mainThrust * Time.deltaTime);
    }
}
```

- **Why SerializableField is used?**

This field is used to make the private variables accessible within the unity editor without making them public.

It also helps maintain encapsulation by allowing the convenient adjustment of properties of the GameObject.



TRANSFORM.ROTATE():-

We'll be rotating by changing the z-axis value in rotation.

```
1 reference
void ProcessRotation()
{
  if(Input.GetKey(KeyCode.A))
    {
      transform.Rotate(Vector3.forward);
    }
```

Vector3 Vector3.forward { get; }

Shorthand for writing Vector3(0, 0, 1).

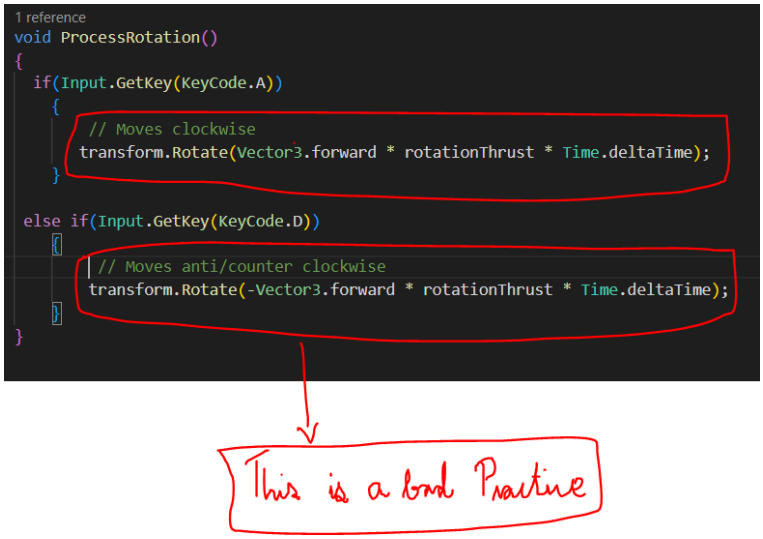
```
void ProcessRotation()
{
  if(Input.GetKey(KeyCode.A))
    {
      // Moves clockwise
      transform.Rotate(Vector3.forward);
    }

  else if(Input.GetKey(KeyCode.D))
    {
      // Moves anti/counter clockwise
      transform.Rotate(-Vector3.forward);
    }
}
```

Next, we'll make it frame-independent.

```
2 references
[SerializeField] float rotationThrust = 1f;
2 references
```

```
1 reference
void ProcessRotation()
{
  if(Input.GetKey(KeyCode.A))
    {
        // Moves clockwise
       transform.Rotate(Vector3.forward * rotationThrust * Time.deltaTime);
    }

 else if(Input.GetKey(KeyCode.D))
    {
        // Moves anti/counter clockwise
        transform.Rotate(-Vector3.forward * rotationThrust * Time.deltaTime);
    }
}
```

But remember this is also bad practice to have multiple things in a code.

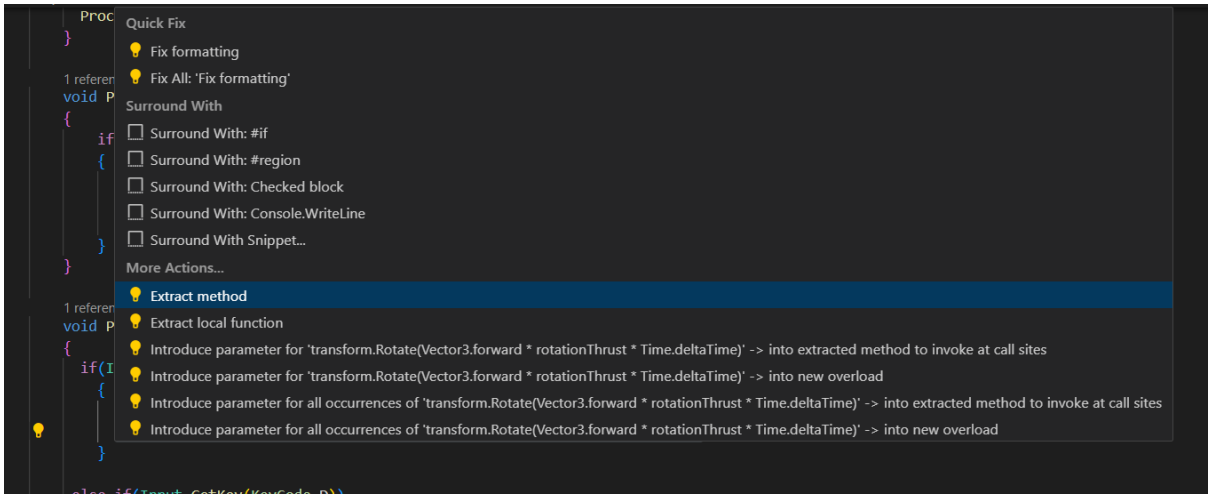


- **Make code readable and expandable.**

  We can create methods.

```
// Moves clockwise
transform.Rotate(Vector3.forward * rotationThrust * Time.deltaTime);
```

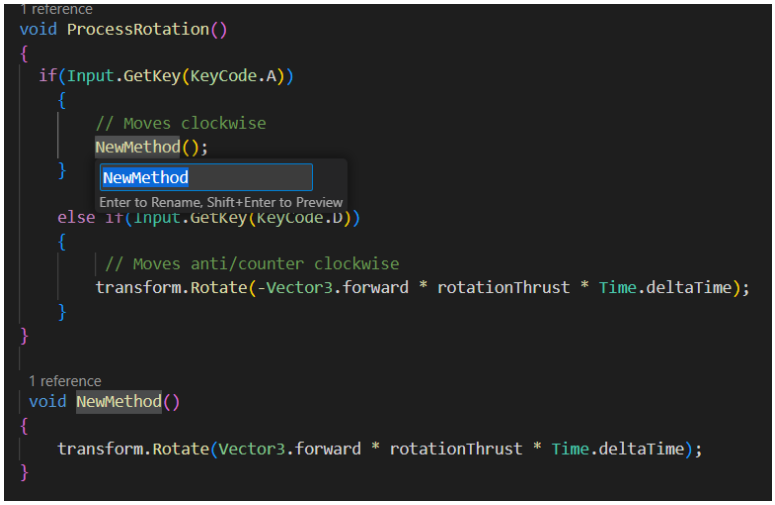
Then press Ctrl + . and select Extract Method.



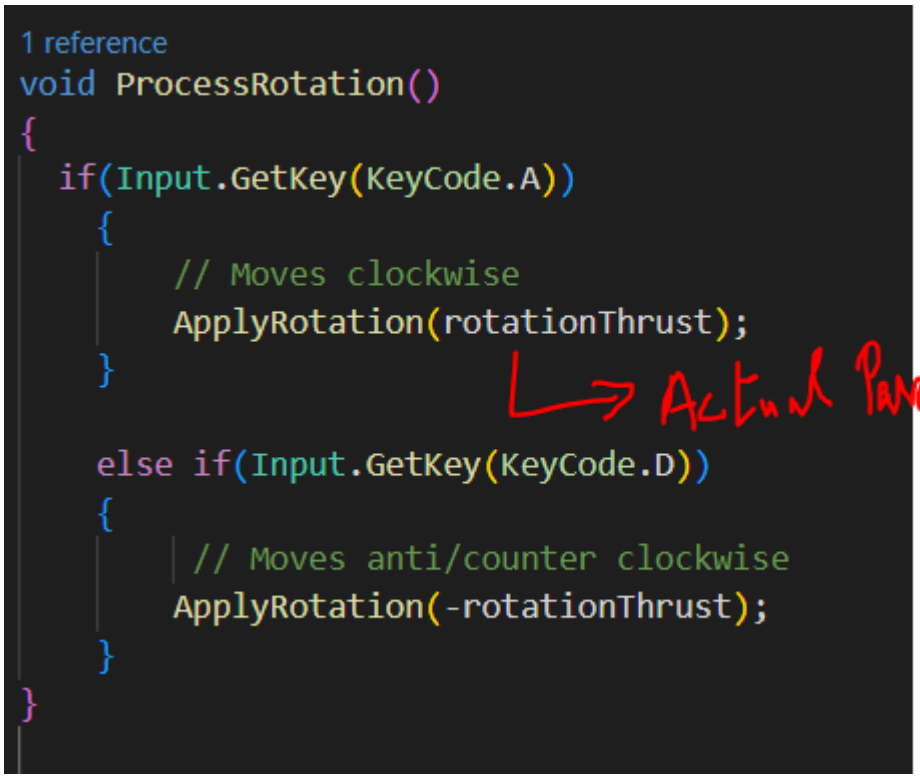

```
1 reference
void ProcessRotation()
{
  if(Input.GetKey(KeyCode.A))
    {
        // Moves clockwise
        NewMethod();
    }

    else if(Input.GetKey(KeyCode.D))
    {
        // Moves anti/counter clockwise
        transform.Rotate(-Vector3.forward * rotationThrust * Time.deltaTime);
    }
}

1 reference
private void NewMethod()
{
    transform.Rotate(Vector3.forward * rotationThrust * Time.deltaTime);
}
```

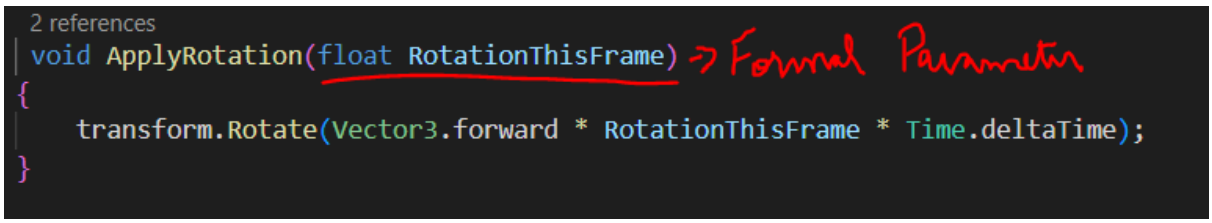
New trick:

- **Double click and Press F2**



- **Then Change the Method Name to Apply Rotation.**





Actual Parameter:-

Actual parameters are the values that are passed to a function when it is ***called***. They are also known as ***arguments***. These values can be ***constants***, ***variables***, ***expressions***, or even other function calls. When a function is called, the actual parameters are evaluated, and their values are copied into the corresponding formal parameters of the called function.

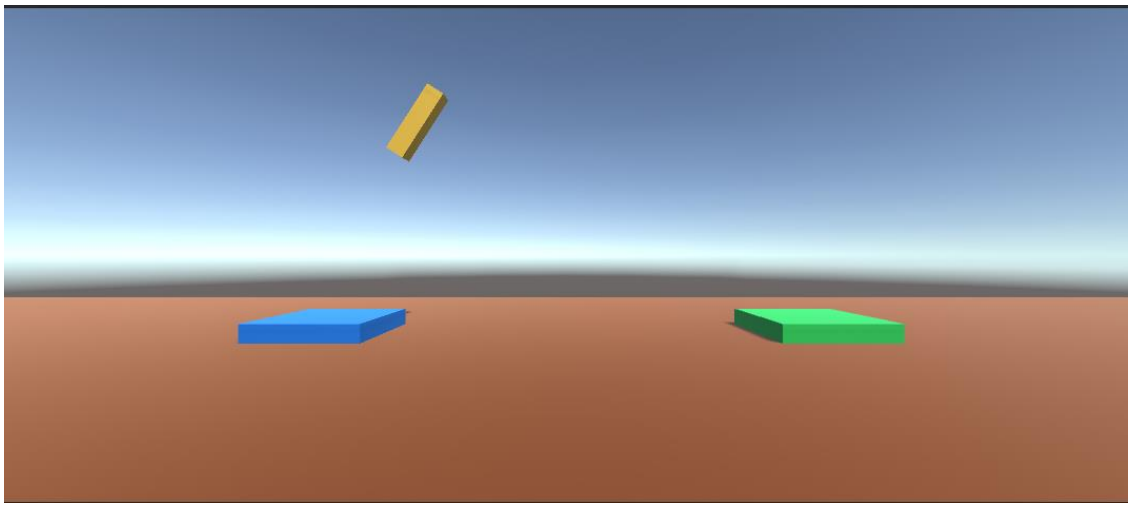
```
int result = add(2, 3);
```

Formal Parameter: -

Formal parameters are the variables declared in the function header that are used to receive the values of the actual parameters passed during function calls. They are also known as ***function parameters***. Formal parameters are used to define the ***function signature*** and to specify the type and number of arguments that a function can accept.

```
int add(int a, int b);
```

Output so far completed.



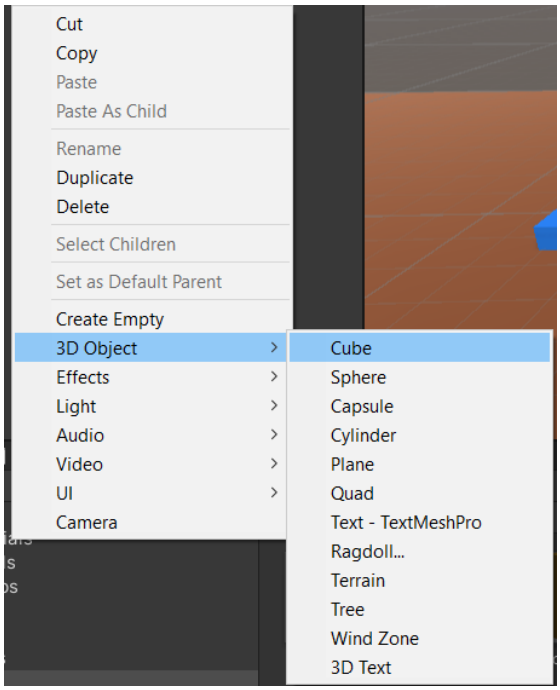
RIGIDBODY CONSTRAINTS: -

If you want to make changes in the instance and the prefab(original).

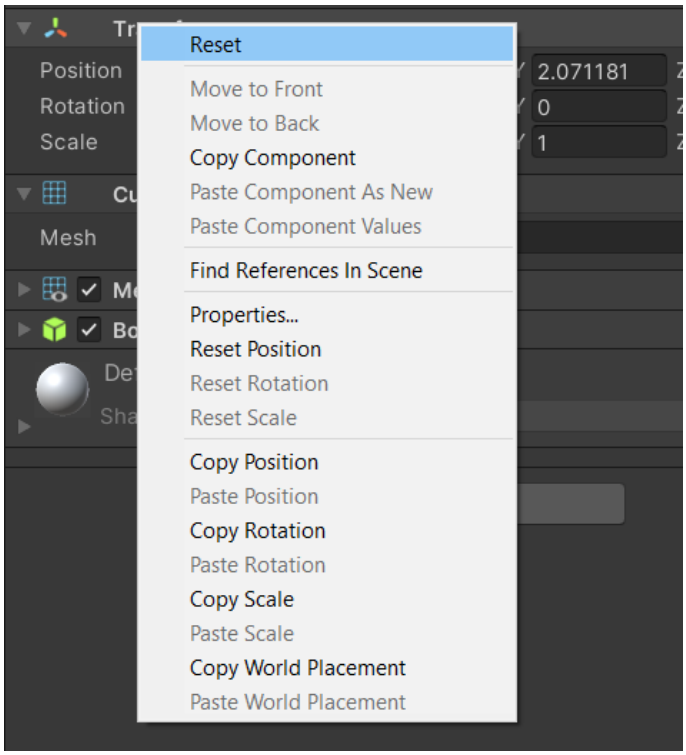
Solution: -

Go to the Rocket in the hierarchy and click overrides and select Apply all.

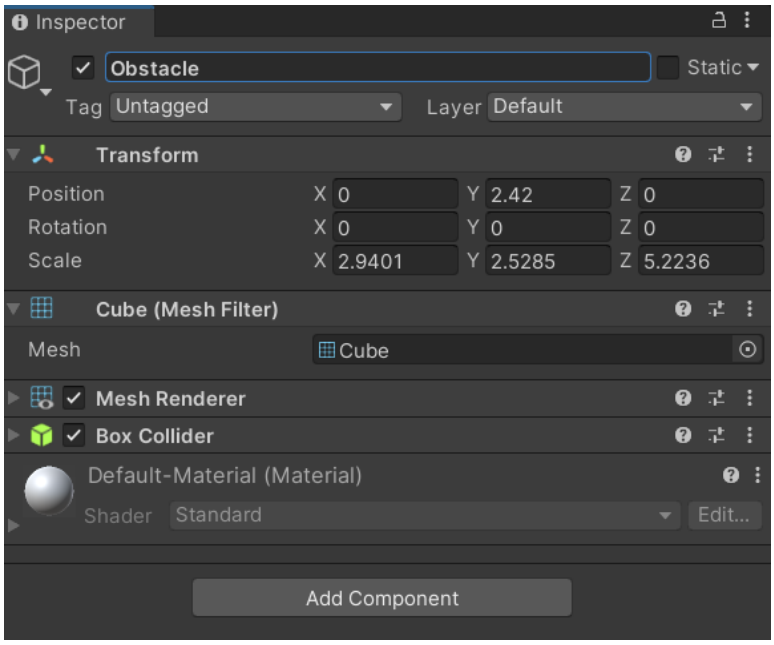
Right click the 3D Object and Select Cube.



For the Particular Cube object go to Transform and select Reset

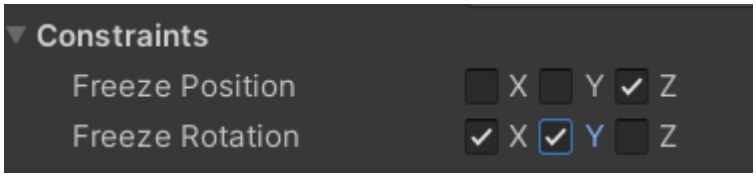


After that bring into the position of y-axis 2.42 after that scale it accordingly and rename it as obstacle.



Later Prefab the obstacle as well.

Create an Obstacle material give it a color drag it to the game object and change the color to pink.



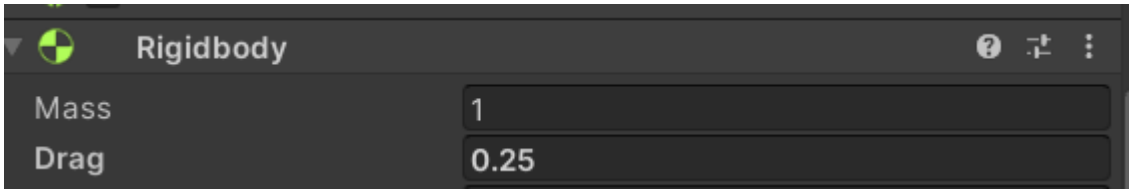
Freeze the respective constraints if you want the rocket to move towards different positions.

Right now, go to the scripts to remove the weird bug where the rocket doesn't move accordingly when it is hit on another object.

Note: Freeze the physics system rotation.

```
2 references
void ApplyRotation(float RotationThisFrame)
{
    rb.freezeRotation = true; // freezing rotation so we can manually rotate
    transform.Rotate(Vector3.forward * RotationThisFrame * Time.deltaTime);
    rb.freezeRotation = false; // unfreezing rotation so the physics system can take over.
}
```

Set the drag to

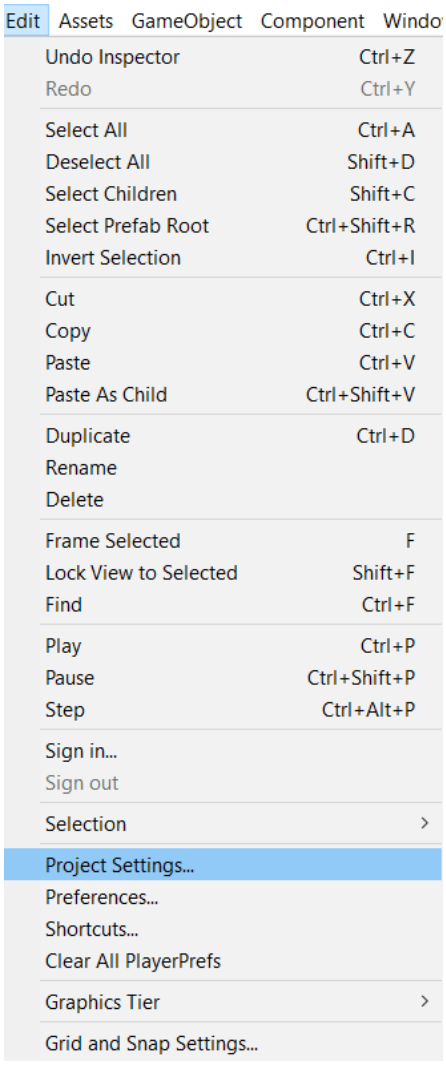


Definition of drag:

Drag is the tendency of an object to slow down due to friction with the air or water that surrounds it. The linear drag applies to positional movement and is set up separately from the angular drag that affects rotational movement.

We need to change the gravity in the world. (Optional)

Step 1:



Source Control Repo:

It is a system for tracking and managing changes to the code and project.

Git- A type of version control system that tracks changes to the file.

Gitlab: Repository hosting service

Repository: Directory or storage space for your project.

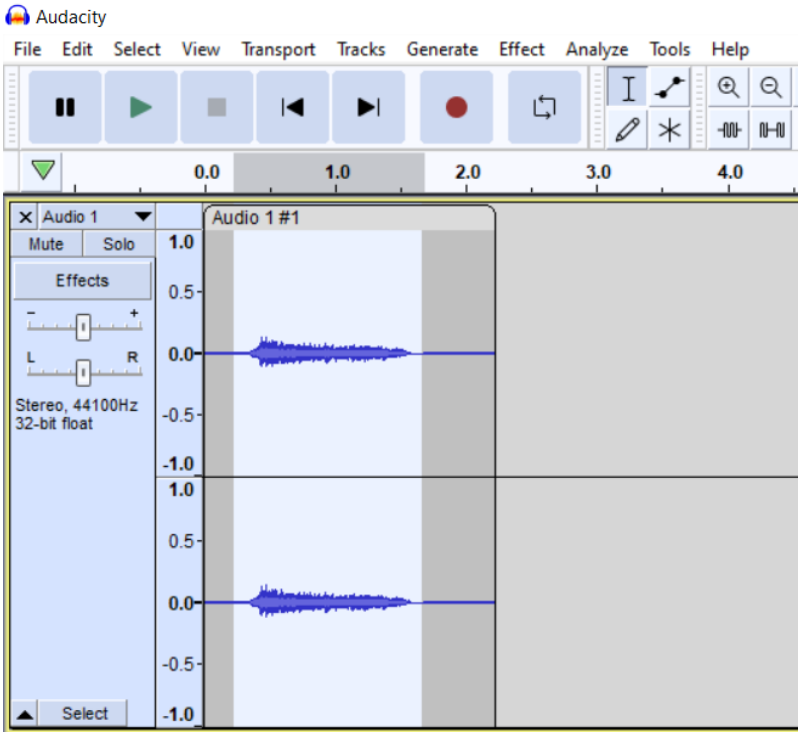
SourceTree: Desktop Client to help view your repository.

Unity Audio Introduction:

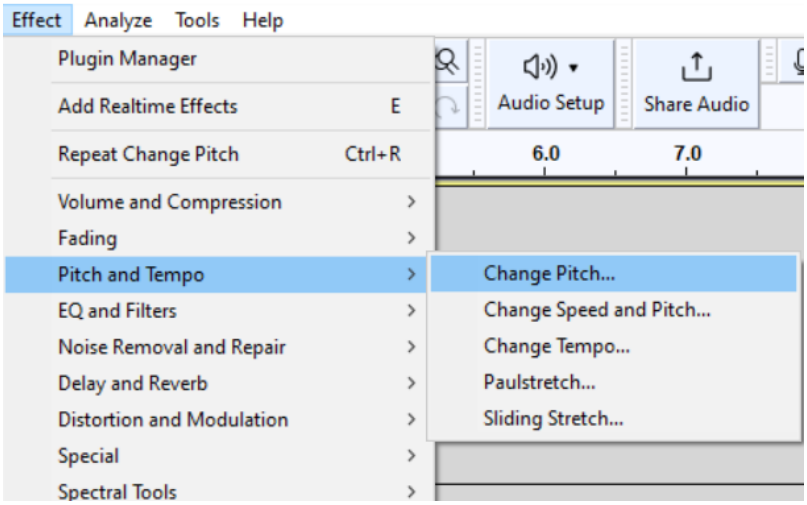
We need three things: -

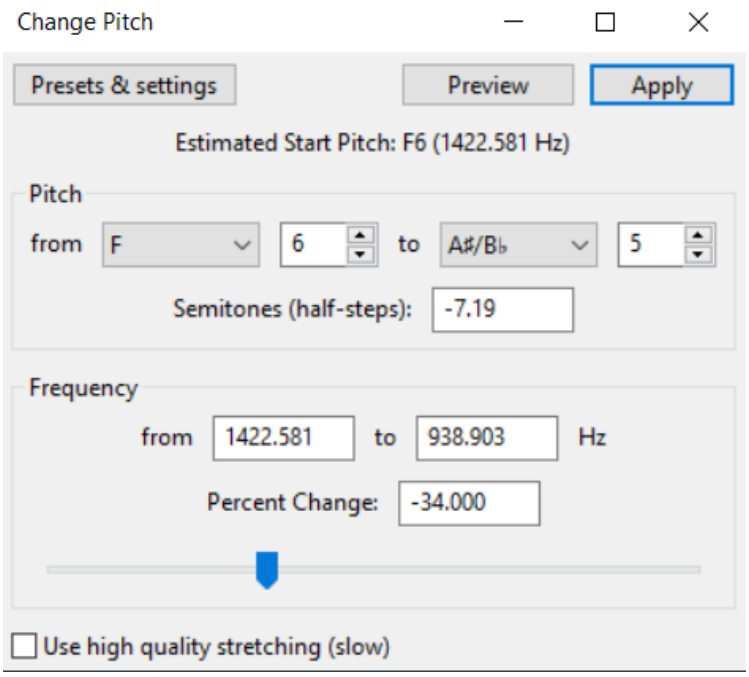
Audio listener is in the main camera.

- Add an audio source for the rocket game object.
- Go to freesound.org.(only if you are making games for fun)
- Instead, we can use audacity (Download them)



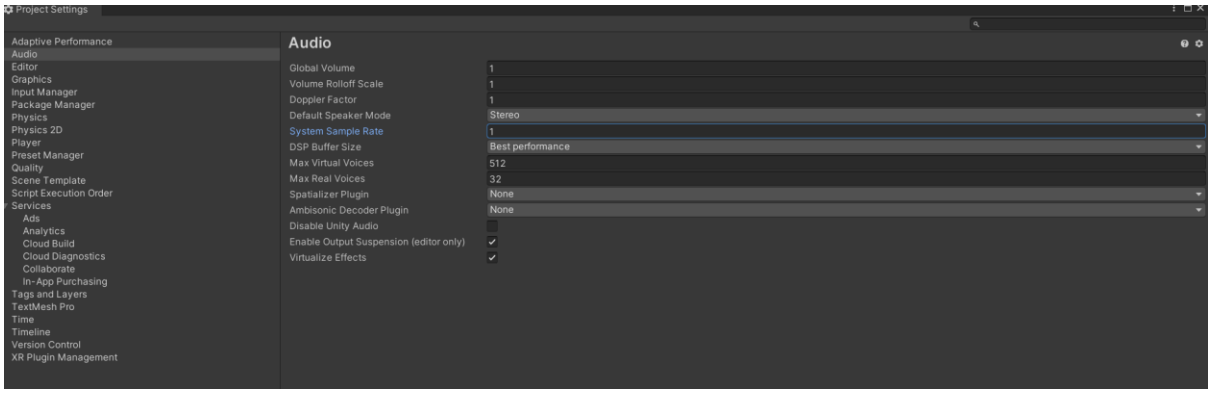
Create a sound from audacity at the same time we can also change the pitch.





After that save it as a .ogg file and then copy it in the asset folder.

Another way to solve the audio problem is to go to the edit settings and set the system sample rate as 1.



Code that is implemented.

```
4 references
AudioSource audioSource;
```

```
1 reference
void ProcessThrusts()
{
    if (Input.GetKey(KeyCode.Space))
    {
        // Adding Time.deltaTime to make it frame independent
        // mainthrust to move an object quickly
        rb.AddRelativeForce(Vector3.up * mainThrust * Time.deltaTime);
        if(!audioSource.isPlaying)
        {
            audioSource.Play();
        }
    }
    else
    {
        audioSource.Stop();
    }
}
```

SWITCH STATEMENT: -

Keep the launch pad tag name as **friendly.**

Keep the landing pad tag name as **finish.**

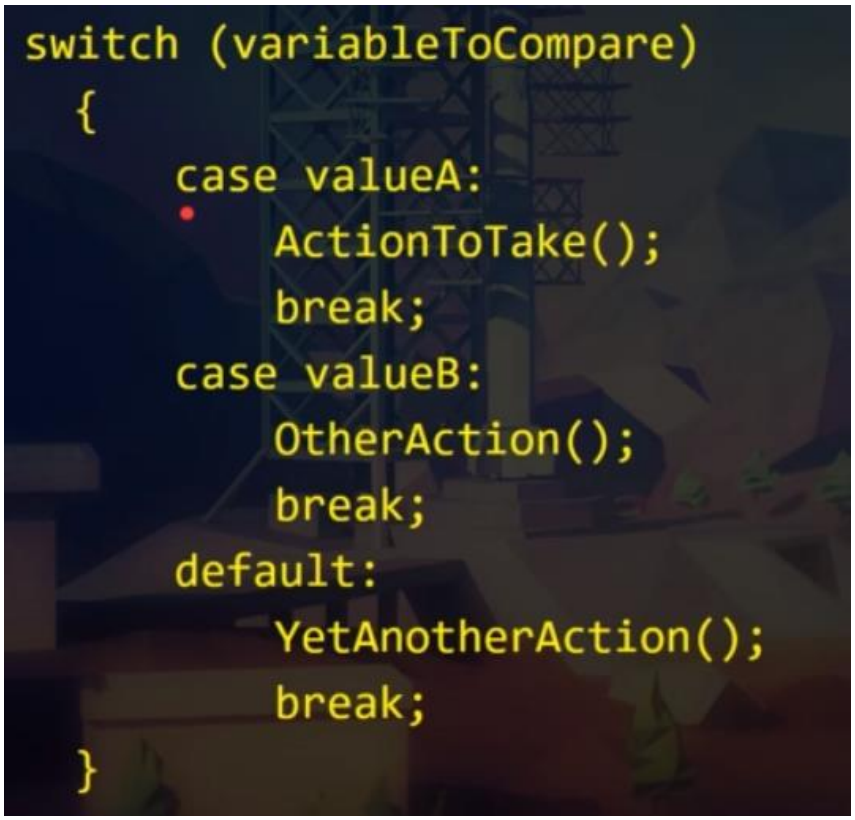
Create a new object called a sphere and keep the y-axis position as 7.98.

Create a new C# Script

What are switch statements?

- They are conditional statements like If/Else Statements.
- Allow ud to compare a single variable to a series of constants(i.e things that don't change or have a "constant" value).

- Syntax of Switch statement.



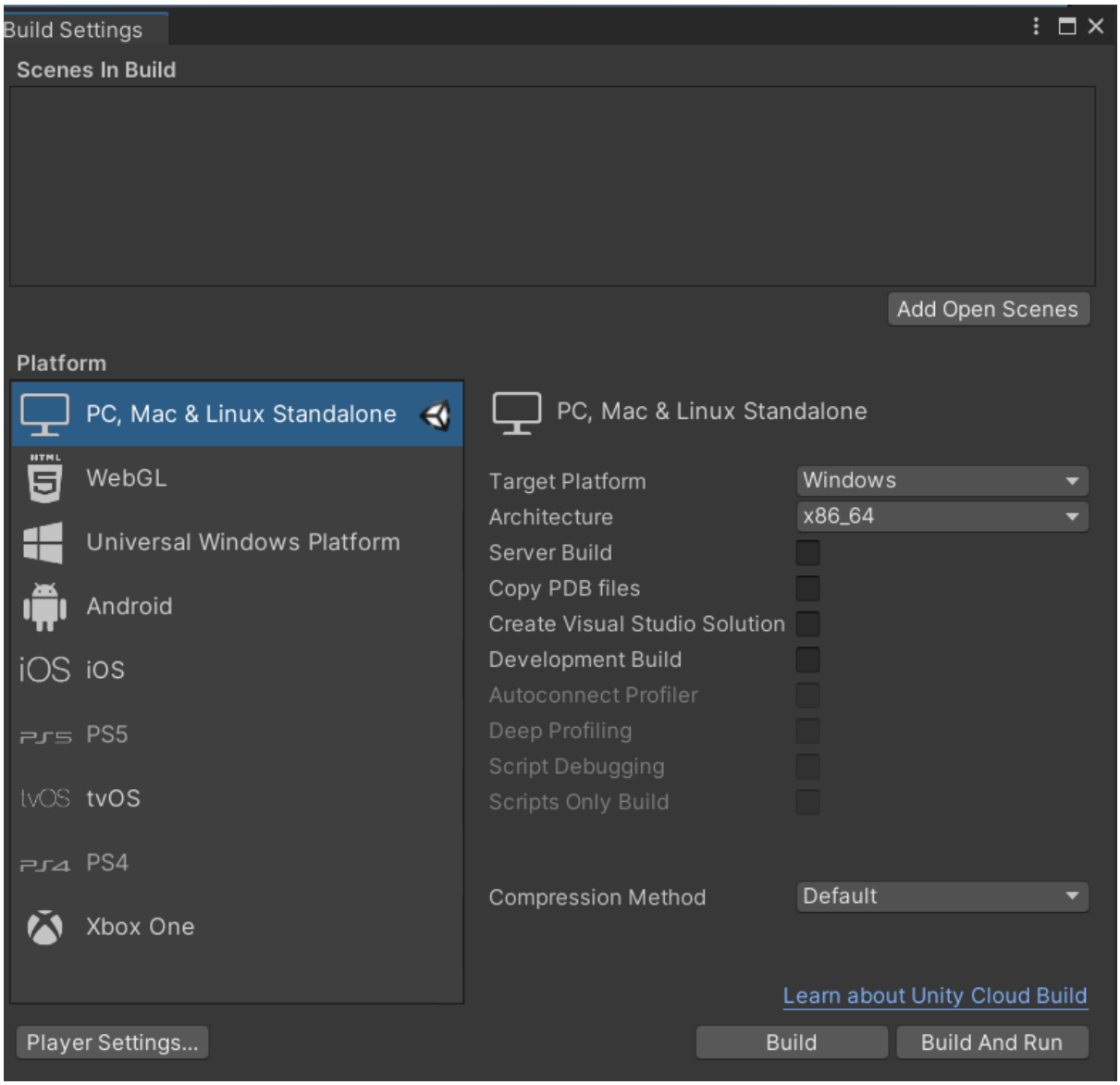
Attach the collision handler to the Rocket.

Code snippet so far.

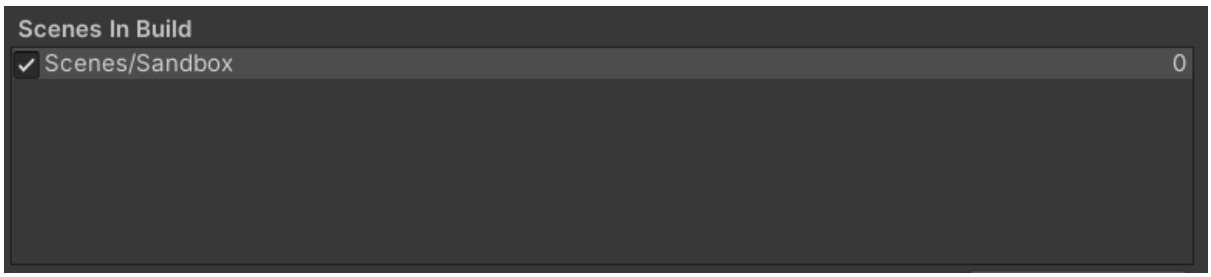
```
0 references
public class CollisionHandler : MonoBehaviour
{
    0 references
    void OnCollisionEnter(Collision other)
    {
        switch (other.gameObject.tag)
        {
            case "Friendly":
                Debug.Log("This is a friendly game object");
                break;
            case "Finish":
                Debug.Log("Congrats the level is completed");
                break;
            case "Fuel":
                Debug.Log("The fuel has been picked");
                break;
            default:
                Debug.Log("I'm not satisfied at all");
                break;
        }
    }
```

Respawn Using Scene Manager:

To load the scene, we have to use the build settings.



Click Add Open Scene or go to the Scene Folder and drag the Scene to the Build Setting.

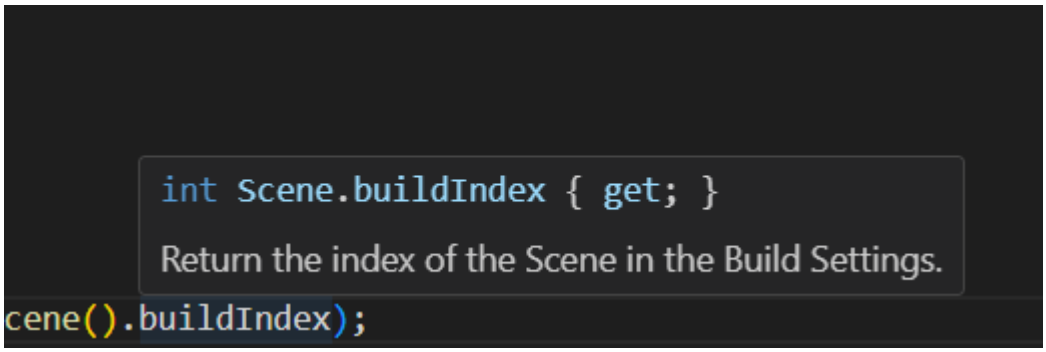


These are referred to as Levels.

Documentation Link for Scene Manager.

https://docs.unity3d.com/ScriptReference/SceneManagement.SceneManager.html

Note: Use the index 0.



Code so far.

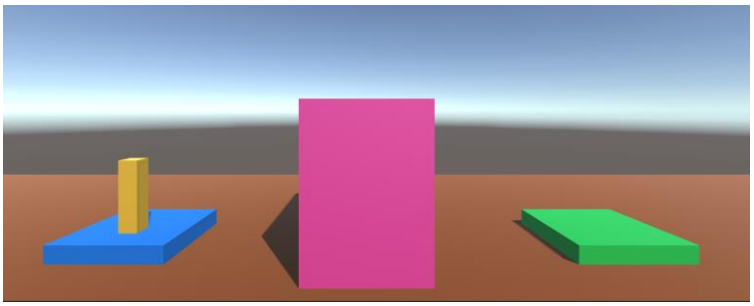
```
public class CollisionHandler : MonoBehaviour
{
    0 references
    void OnCollisionEnter(Collision other)
    {
        switch (other
        {
            case "Frie
                Debug.Log("This is a friendly game object");
                break;
            case "Finish":
                Debug.Log("Congrats the level is completed");
                break;
            case "Fuel":
                Debug.Log("The fuel has been picked");
                break;
            default:
                ReloadLevel();
                break;
        }
    }

    1 reference
    void ReloadLevel()
    {
        // We can either use a index or string.
        //SceneManager.LoadScene(0);
        int currentSceneIndex = SceneManager.GetActiveScene().buildIndex;
        SceneManager.LoadScene(currentSceneIndex);
    }

}
```

```
void Debug.Log(object message) (+ 1 overload)
Logs a message to the Unity Console.
```

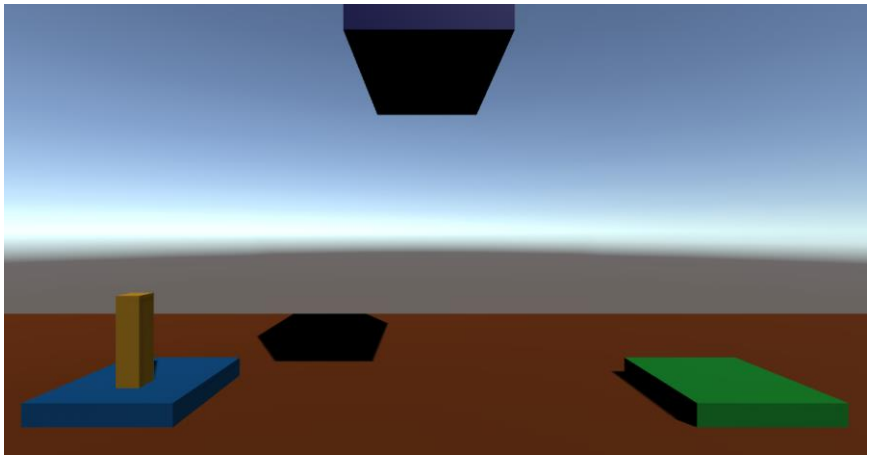
Load Next Level.

To go to the next level.

- **Duplicate the scene and make minor changes to the level. Like in the picture**

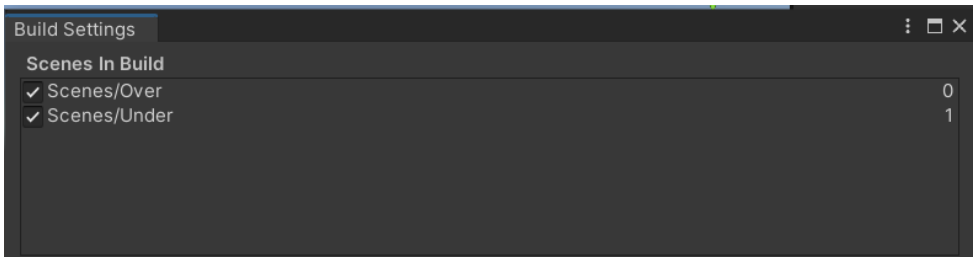


Level 1



Level 2

- **Arrange accordingly in the build settings.**

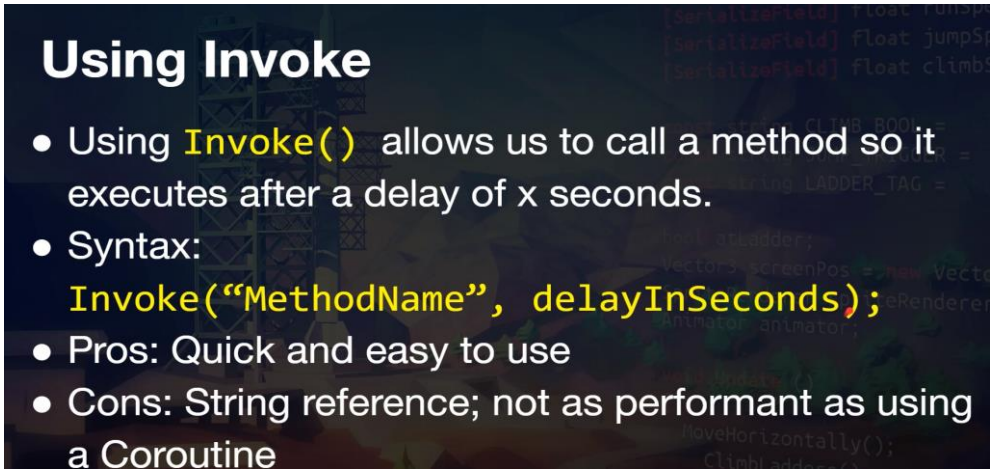


- **Code for changing the level and looping back to the same level is:**

```
public class CollisionHandler : MonoBehaviour
{
    0 references
    void OnCollisionEnter(Collision other)
    {
        switch (other.gameObject.tag)
        {
            case "Friendly":
                Debug.Log("This is a friendly game object");
                break;
            case "Finish":
                //Debug.Log("Congrats the level is completed");
                LoadNextLevel();
                break;
            case "Fuel":
                Debug.Log("The fuel has been picked");
                break;
            default:
                ReloadLevel();
                break;
        }
    }


    1 reference
    void LoadNextLevel()
    {
        // This Loads to the next level
        int currentSceneIndex = SceneManager.GetActiveScene().buildIndex;
        int nextSceneIndex = currentSceneIndex + 1; // i.e index position is 0 and 1 so  1 + 1 = 2
        // If next scene index is 2 and if it is equal to the scene count in build go back to the first level
        if(nextSceneIndex == SceneManager.sceneCountInBuildSettings)
        {
            nextSceneIndex = 0; // Goes back to level 1
        }
        SceneManager.LoadScene(nextSceneIndex);
    }
```

Using Invoke.

What is Invoke?

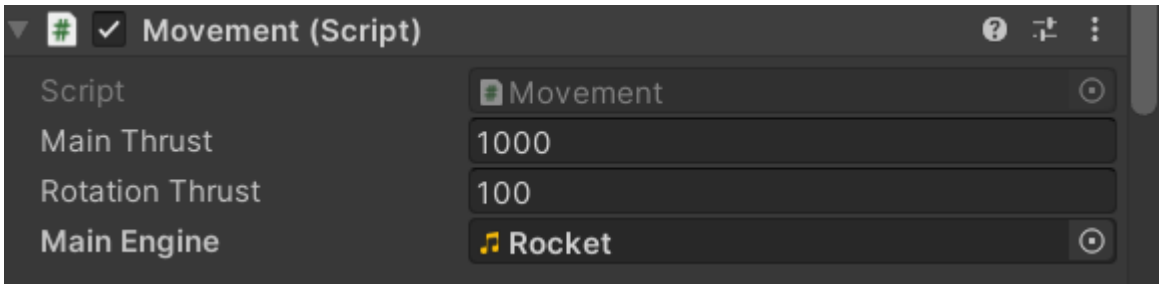


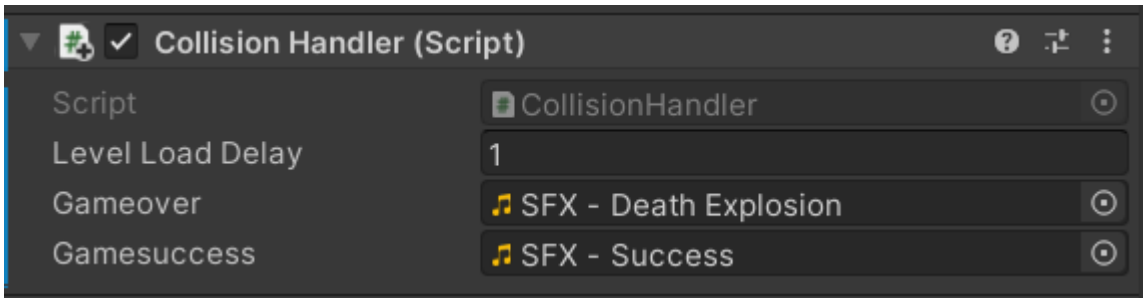
Code Implementation:

```
0 references
public class CollisionHandler : MonoBehaviour
{
2 references
[SerializeField] float  levelLoadDelay = 1f; // Set the delay for 1 second
    0 references
    void OnCollisionEnter(Collision other)
    {
        switch (other.gameObject.tag)
        {
            case "Friendly":
                Debug.Log("This is a friendly game object");
                break;
            case "Finish":
                //Debug.Log("Congrats the level is completed");
                StartSucessSequence(); // Ctrl + . Helps in generating the method
                break;
            default:
                StartCrashSequence();
                break;
        }
    }
```

```
void StartCrashSequence()
{
    GetComponent<Movement>().enabled = false;
    Invoke("ReloadLevel", levelLoadDelay); // Invoke allows us to call a method so it executes after a delay of x seconds
}

1 reference
void StartSucessSequence()
{
    GetComponent<Movement>().enabled = false;
    Invoke("LoadNextLevel", levelLoadDelay); // Invoke allows us to call a method so it executes after a delay of x seconds
}

 0 references
 void LoadNextLevel()
{
    // This Loads to the next level
     int currentSceneIndex = SceneManager.GetActiveScene().buildIndex;
     int nextSceneIndex = currentSceneIndex + 1; // i.e index position is 0 and 1 so  1 + 1 = 2
     // If next scene index is 2 and if it is equal to the scene count in build go back to the first level
     if(nextSceneIndex == SceneManager.sceneCountInBuildSettings)
     {
         nextSceneIndex = 0; // Goes back to level 1
     }
    SceneManager.LoadScene(nextSceneIndex);
}

0 references
void ReloadLevel()
{
    // We can either use a index or string.
    //SceneManager.LoadScene(0);
    // This reloads the Level during game over
    int currentSceneIndex = SceneManager.GetActiveScene().buildIndex;
    SceneManager.LoadScene(currentSceneIndex);
}
```

Triggering Multiple Audio Clips:





Implementation of the code:

```
0 references
public class CollisionHandler : MonoBehaviour
{
2 references
[SerializeField] float  levelLoadDelay = 1f; // Set the delay for 1 second
1 reference
[SerializeField] AudioClip gameover;
1 reference
[SerializeField] AudioClip gamesuccess;

3 references
AudioSource audioSource;

0 references
void Start()
{
  audioSource = GetComponent<AudioSource>();
}
    0 references
```

```
1 reference
void StartCrashSequence()
{
    // Add SFX Upon Crash
    //audioSource = GetComponent<AudioSource>();
    audioSource.PlayOneShot(gameover);
    GetComponent<Movement>().enabled = false;
    Invoke("ReloadLevel", levelLoadDelay); // Invoke allows us to call a method so it executes after a delay of x seconds
}

1 reference
void StartSucessSequence()
{
  //audioSource = GetComponent<AudioSource>();
  // Add SFX Upon Crash
  audioSource.PlayOneShot(gamesuccess);
  GetComponent<Movement>().enabled = false;
  Invoke("LoadNextLevel", levelLoadDelay); // Invoke allows us to call a method so it executes after a delay of x seconds
}
```

- **Bool Variable for State:**

```
3 references
bool isTransition = false; // Keep the initial transition state as false, currently we are playing the game and not bumping onto something
```

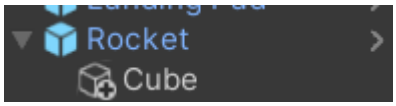
```
0 references
void OnCollisionEnter(Collision other)
{
  if (isTransition) // If the transition is false
  {
    return; // Return to normalcy
  }

   switch (other.gameObject.tag)
   {
      case "Friendly":
          Debug.Log("This is a friendly game object");
          break;
      case "Finish":
         //Debug.Log("Congrats the level is completed");
          StartSucessSequence(); // Ctrl + . Helps in generating the method
          break;
      default:
          StartCrashSequence();
          break;
   }


}
```

```
1 reference
void StartCrashSequence()
{
   // Add SFX Upon Crash
   //audioSource = GetComponent<AudioSource>();
   isTransition = true; // We set transition true
   audioSource.Stop();
   audioSource.PlayOneShot(gameover);
   GetComponent<Movement>().enabled = false;
   Invoke("ReloadLevel", levelLoadDelay); // Invoke allows us to call a method so it executes after a delay of x seconds
   }
```

```
1 reference
void StartSucessSequence()
{
  //audioSource = GetComponent<AudioSource>();
  // Add SFX Upon Crash
  isTransition = true; // We set transition true
  audioSource.Stop();
  audioSource.PlayOneShot(gamesuccess);
  GetComponent<Movement>().enabled = false;
  Invoke("LoadNextLevel", levelLoadDelay); // Invoke allows us to call a method so it executes after a delay of x seconds
}
```

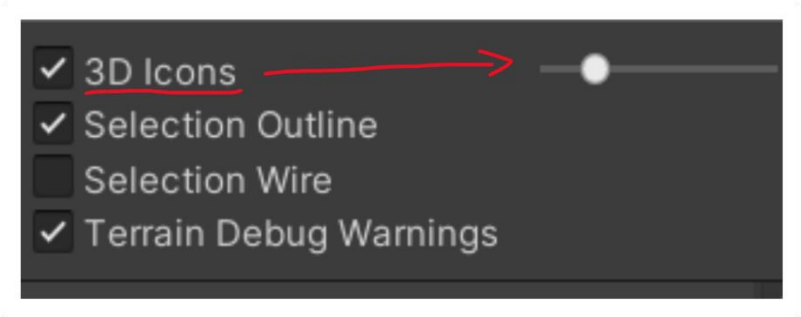
- **Make Rocket Look Spiffy:**

  Note If you make changes to the instance it's not necessary that the prefab be changed.

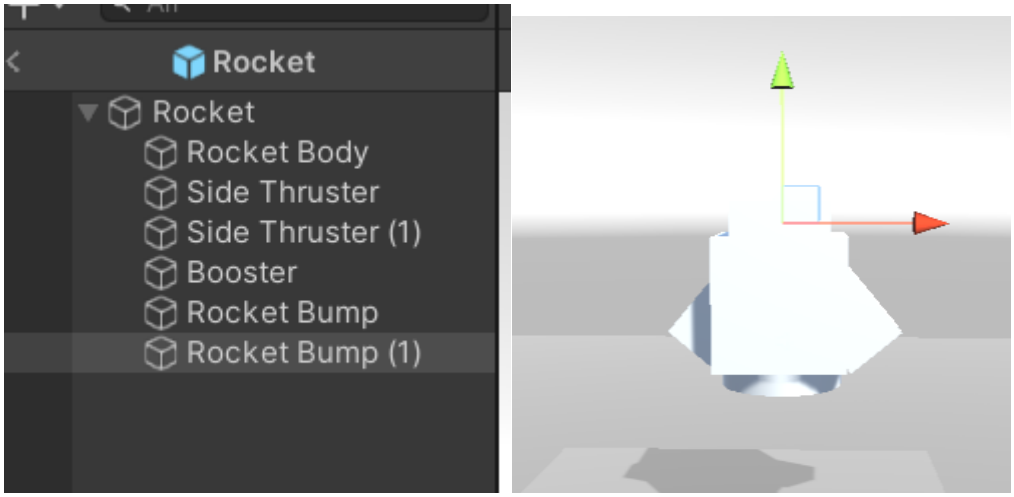


The cube will sit as a child of the Rocke Game Object

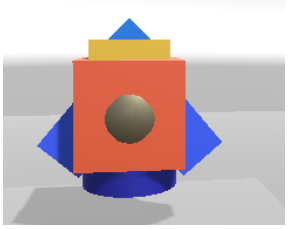
If you want to reduce the 3D icon, you can go to gizmos and reduce the size there.



In the prefab options, we can arrange a lot of child objects independently without affecting the entire scene.

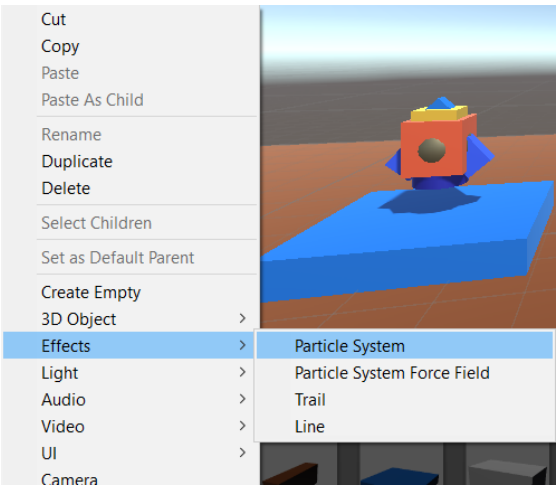


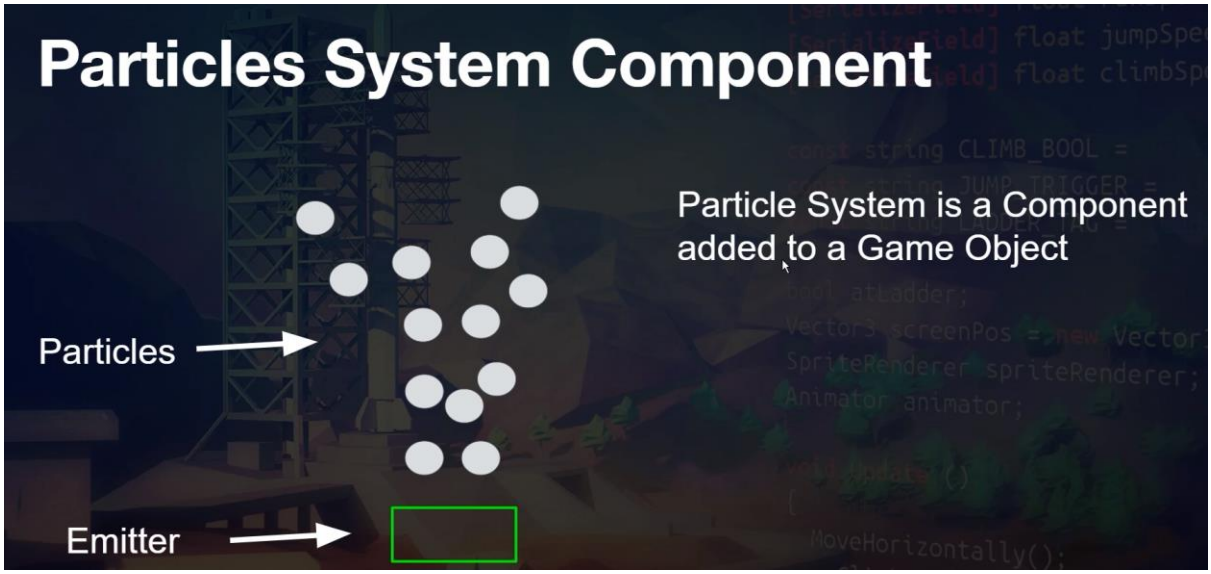
So, the result of adding materials is.



- **How to trigger the Particle System:**

  Where to select a particle system.





To paste some files don't do it directly make sure that you're in the folder that you want.

Another important thing while keeping the particles as a child of a parent make sure you copy those and don't take those prefabs which is present in the folder.

Make sure looping is turned off or else the particle will be still present. Make sure player on awake is also off.

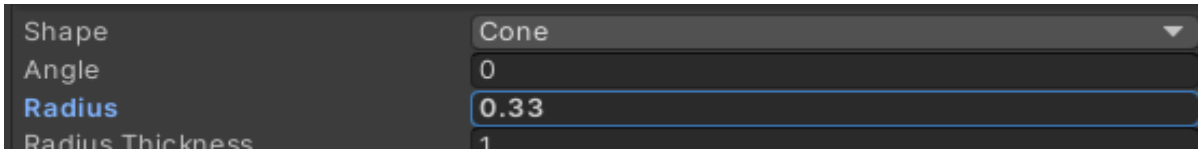
Code for Implementation: (THIS IS A COLLISION HANDLER SCRIPT)

```
1 reference
[SerializeField] ParticleSystem particle_gameover;
1 reference
[SerializeField] ParticleSystem particle_gamesuccess;
```

```
1 reference
void StartCrashSequence()
{
    // Add SFX Upon Crash
    //audioSource = GetComponent<AudioSource>();

    isTransition = true; // We set transition true
    audioSource.Stop();
    audioSource.PlayOneShot(gameover);
    particle_gameover.Play(); // Triggers the gameover particle
    GetComponent<Movement>().enabled = false;
    Invoke("ReloadLevel", levelLoadDelay); // Invoke allows us to call a method so it executes after a delay of x seconds
    }

1 reference
void StartSucessSequence()
{
  //audioSource = GetComponent<AudioSource>();
  // Add SFX Upon Crash
  isTransition = true; // We set transition true
  audioSource.Stop();
  audioSource.PlayOneShot(gamesuccess);
  particle_gamesuccess.Play(); // Triggers the success particle
  GetComponent<Movement>().enabled = false;
  Invoke("LoadNextLevel", levelLoadDelay); // Invoke allows us to call a method so it executes after a delay of x seconds
}
0 references
```

- **Particles for Rocket Boosters**

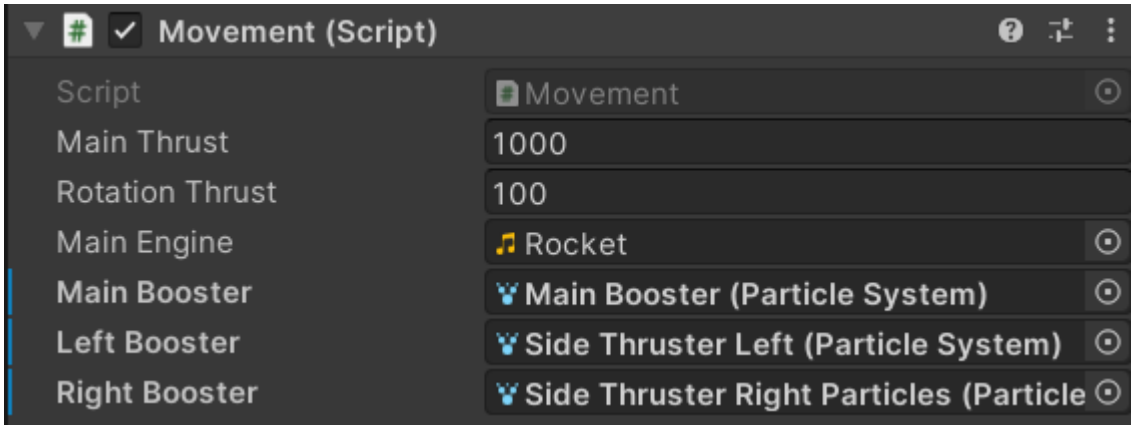
To change the Radius of a particle we can do as per the picture.



Logical Explanation of the Code

```
3 references
[SerializeField] ParticleSystem MainBooster;
3 references
[SerializeField] ParticleSystem LeftBooster;
3 references
[SerializeField] ParticleSystem RightBooster;
```

As per the inspector



Particle effect for the space bar.

```
1 reference
ProcessThrusts()

if (Input.GetKey(KeyCode.Space))
{
    // Adding Time.deltaTime to make it frame independent
    // mainthrust to move an object quickly
    rb.AddRelativeForce(Vector3.up * mainThrust * Time.deltaTime);

    if(!audioSource.isPlaying)
    {
      audioSource.PlayOneShot(mainEngine);
    }
    if(!MainBooster.isPlaying) { MainBooster.Play(); }// Checks if the rocket booster particle is not playing,If yes there will be a particle effect

}
else
{
     audioSource.Stop();
     MainBooster.Stop();
}
```

Particle Effect for the Left button:

```
1 reference
void ProcessRotation()
{
  if(Input.GetKey(KeyCode.A))
    {
        // Moves clockwise
        ApplyRotation(rotationThrust);
        if(!LeftBooster.isPlaying) // Checks if the rocket booster left particle is not playing, If yes there will be a particle effect
        {
            LeftBooster.Play(); // Play the Particle
        }
    }
```

Particle effect for the Right button:

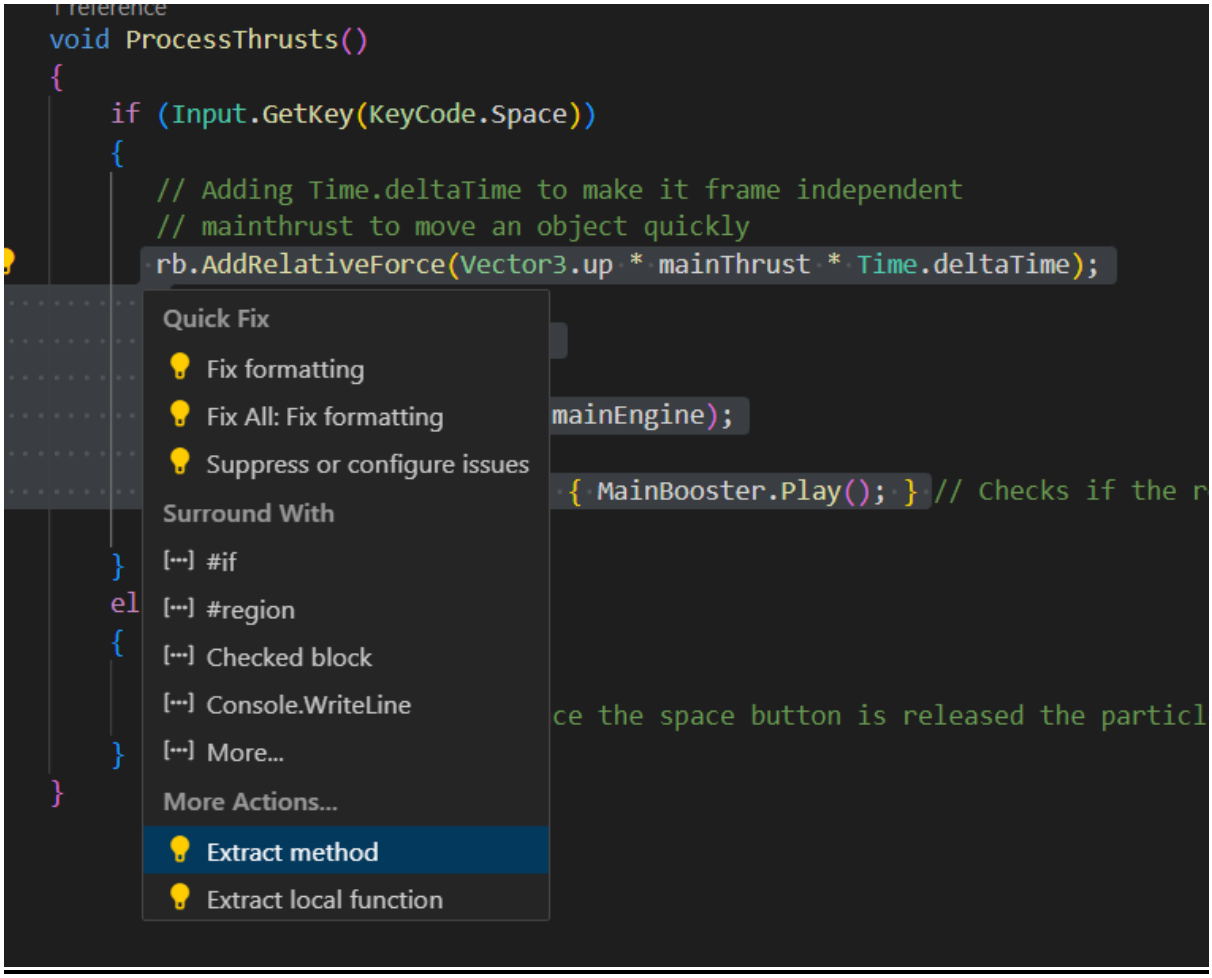
```
    else if(Input.GetKey(KeyCode.D))
    {
        // Moves anti/counter clockwise
        ApplyRotation(-rotationThrust);
        if(!RightBooster.isPlaying) // Checks if the rocket booster right particle is not playing, If yes there will be a particle effect
        {
            RightBooster.Play(); // Play the Particle
        }
    }
```

Particle effect for the button when released for left, right and space.

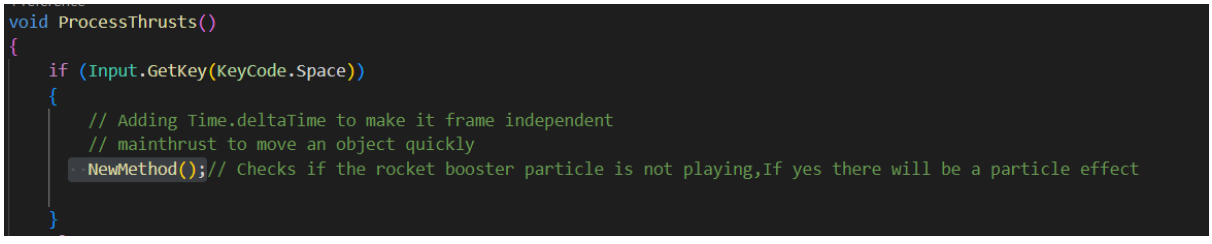
```
else
{
    LeftBooster.Stop(); // If the key press is released Stop the particle effect
    RightBooster.Stop(); // If the key press is released Stop the particle effect
}
```

```
else
{
    audioSource.Stop();
    MainBooster.Stop(); // Once the space button is released the particle effect gets stopped
}
```

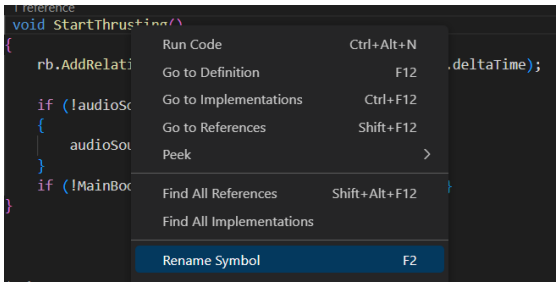
- **Refactor with exact methods: -**



This is the result.



If you want to change the name of the function, click **Rename Symbol.**



You can even click f2 to Rename the Func name

- **Add Cheats/Debug Keys:**

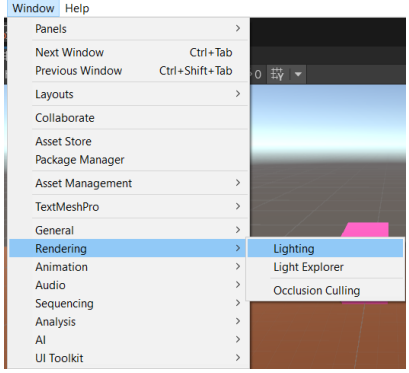
```
1 reference
void NextLevelKey()
{
    if (Input.GetKeyDown(KeyCode.L))
    {
        LoadNextLevel();
    }

    else if (Input.GetKeyDown(KeyCode.C)) // Once the button is clicked the CollisionDisabled will change from false to true
    {
        CollisionDisabled = !CollisionDisabled; //toggle collision i.e false = true and true = false
    }
}
```

```
void OnCollisionEnter(Collision other)
{
    if (isTransition || CollisionDisabled) // If the transition is false
    {
        return; // Return to normalcy
    }

     switch (other.gameObject.tag)
    {
        case "Friendly":
            Debug.Log("This is a friendly game object");
            break;
        case "Finish":
            //Debug.Log("Congrats the level is completed");
            StartSucessSequence(); // Ctrl + . Helps in generating the method
            break;
        default:
            StartCrashSequence();
            break;
    }


}
```

- **Make environment from cubes:**

  Use can use optional cine machine which is built in unity.

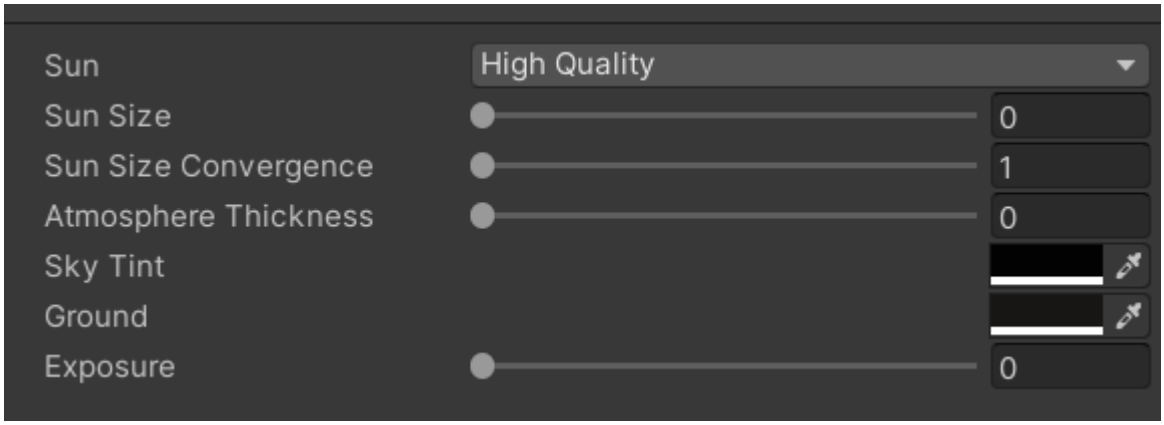
  ➔ To open a lighting tab, you have to go as per the picture.



➔ To change the Sky box material in the environment, just drag the material color from the material folder.
➔ From the Inspector change Standard to Skybox and make it procedural.

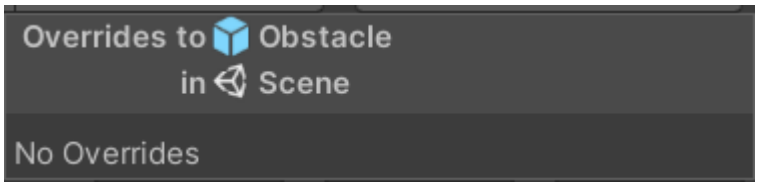
➔ Make the changes here:

Final Changes



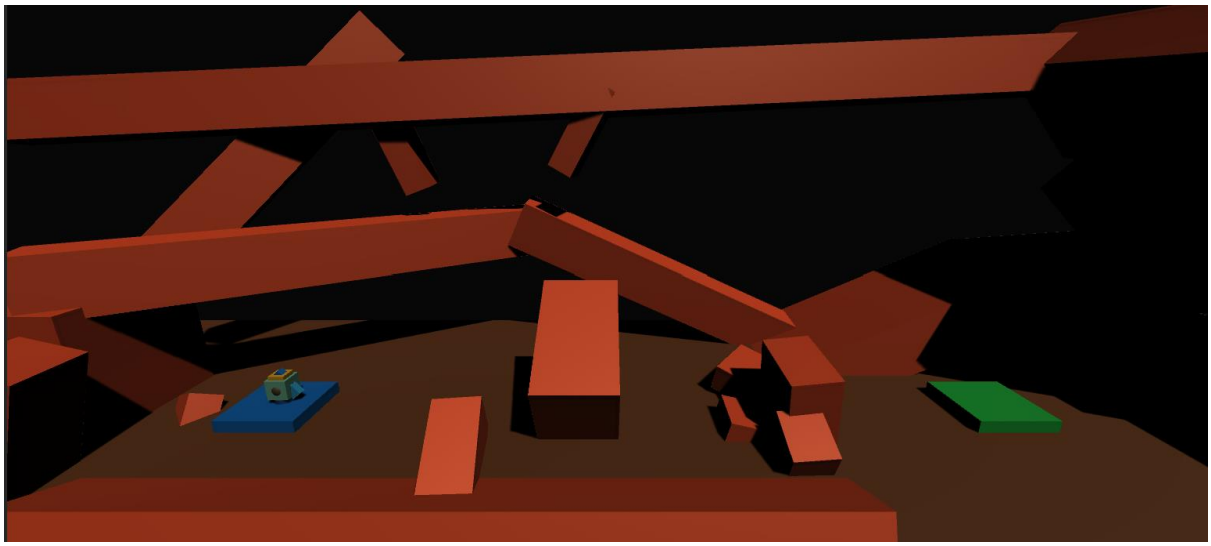
Next:

- Change the main camera from blue to black in the **Inspector.**
- After that change clear fangs to Solid Color.



If you apply overrides here the changes will also affect the respective Prefabs

Final Result

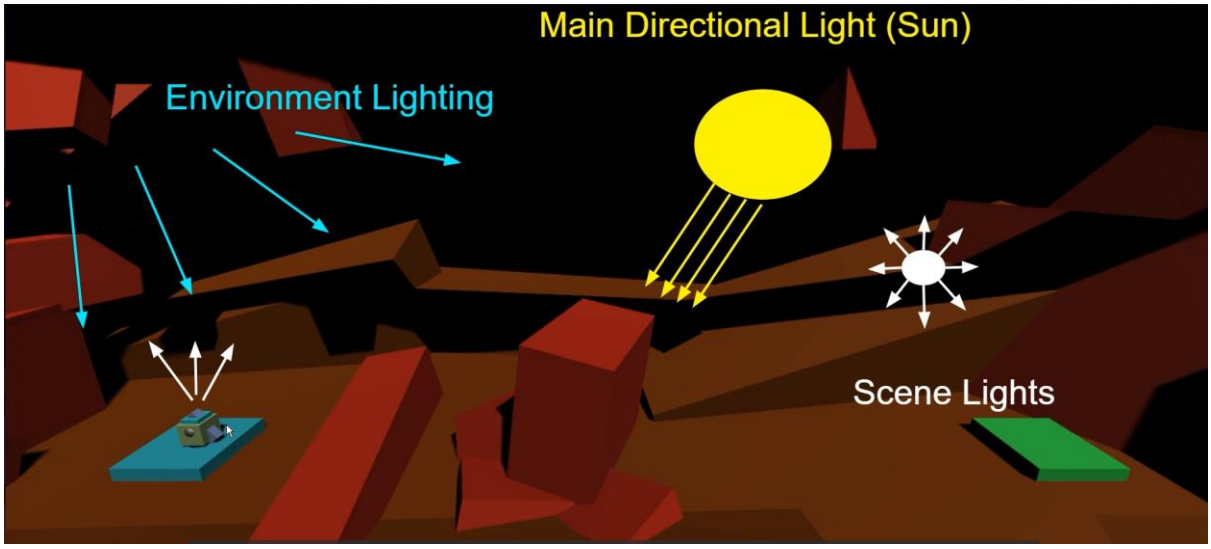


- **How to add Lights in Unity?**

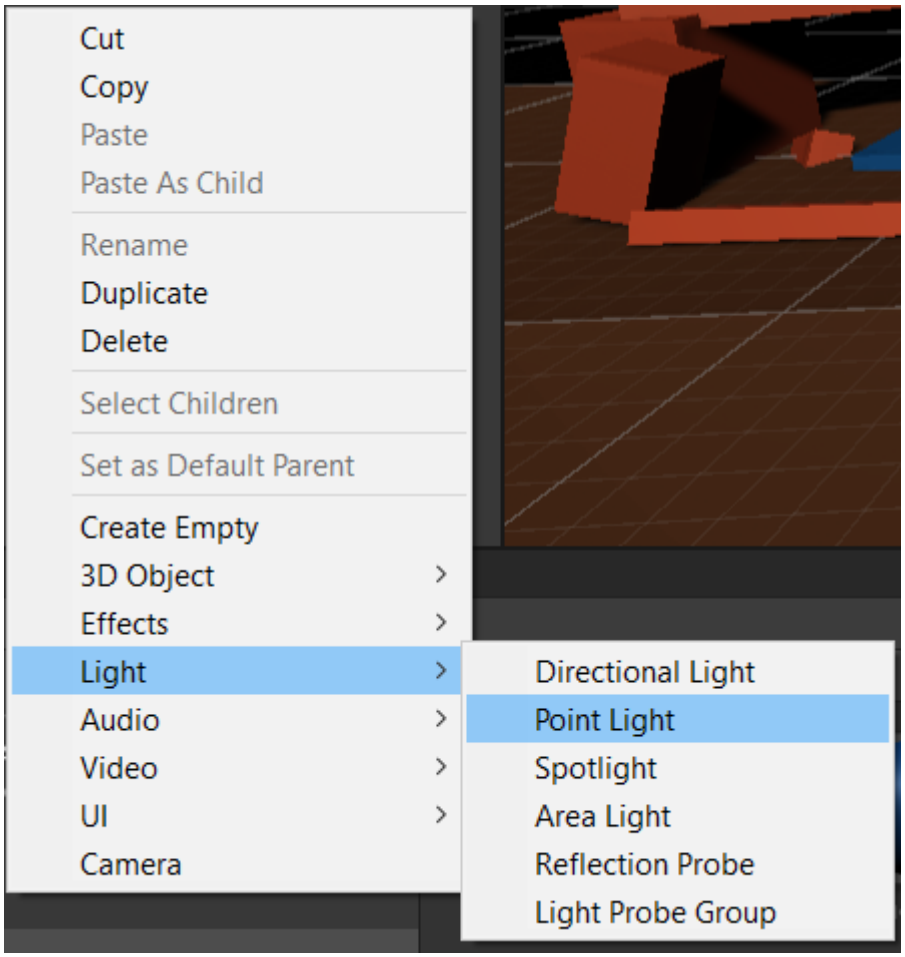
  We have Main Directional Light which comes from the Sun.

  And we have some Environmental Lighting

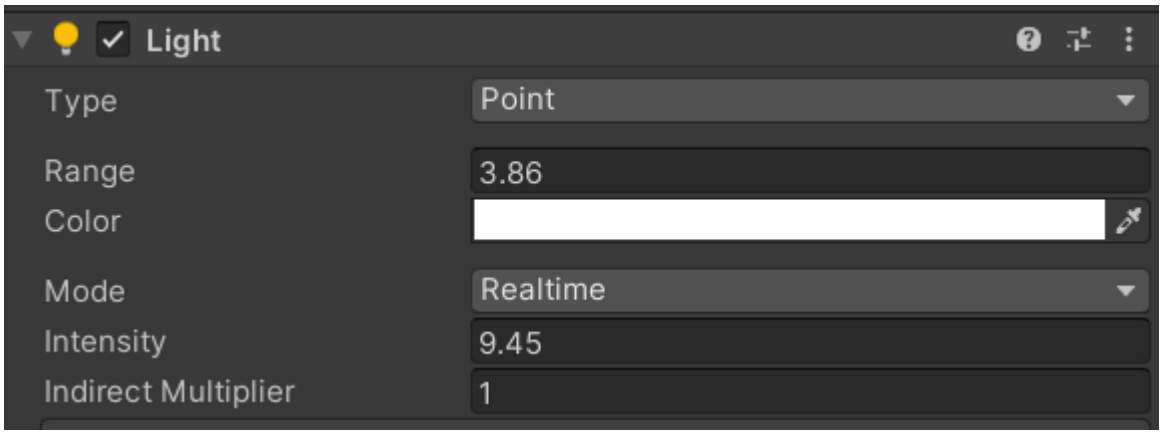
  We can also add lights to the game objects.

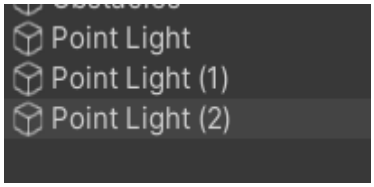


Select the Point Light

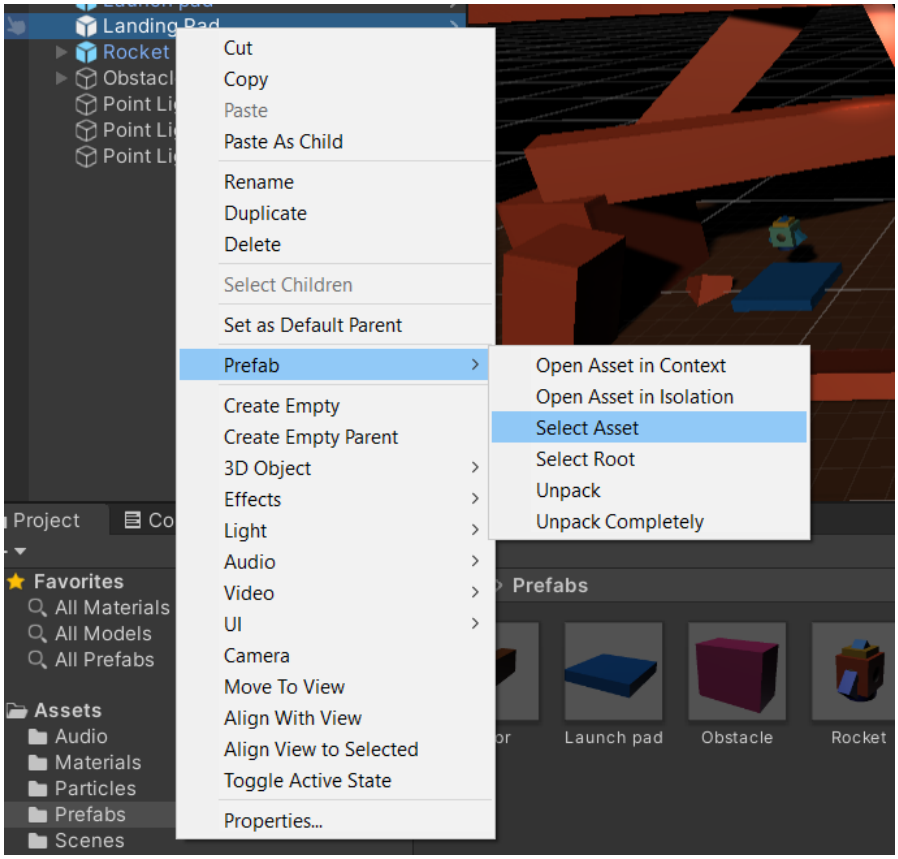


At point light you can change the range and intensity

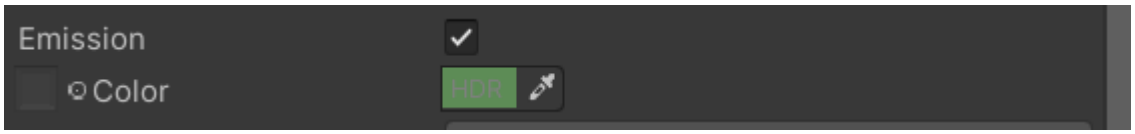




You can duplicate the spotlight and place it in various areas

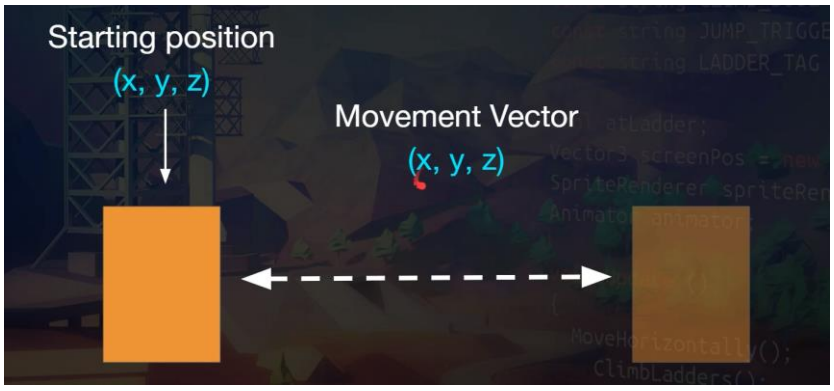


From the Material Folder, you can enable emission as per you wish.

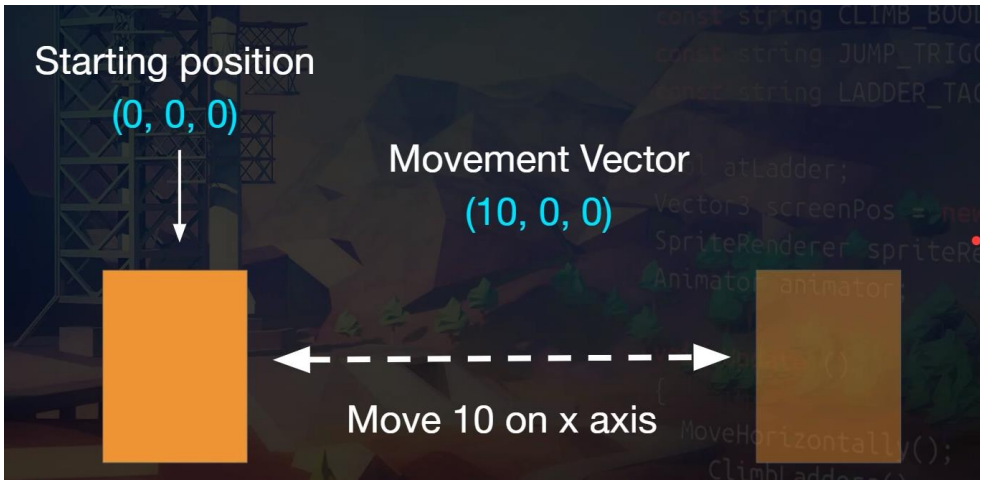


- **Move Obstacles with code:**

  To move in a consistent direction. (x,y,z) are 3D Vectors

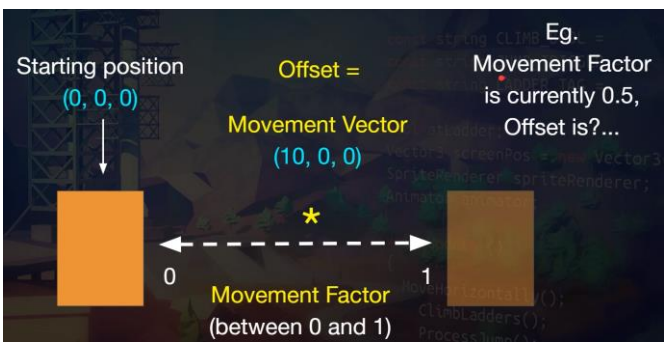


- ➔ For example, we might start at 0,0,0 right at the origin of our world.
- ➔ Then we have a movement vector of 10,0,0 i.e. moving ten on the x-axis and 0 on the two axes.
- ➔ So we will move 10 on the X and if we're applying a movement vector, so if we're adding a movement vector to our starting position in.



To Oscillate (Back and Forth)

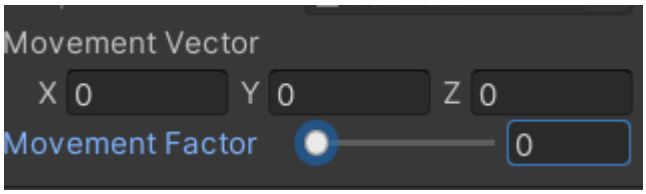
We can apply a Movement Factor. It is going to be 0 and 1.



That is **10 * 0.5 = 5**, **0 * 0.5 = 0**, **0 * 0.5 = 0 (5,0,0)**

```
0 references
[SerializeField] [Range(0,1)] float MovementFactor;
```

The Range will give this output to the inspector.



It goes from 0 to 1.

Code so far

```
0 references
public class Oscillator : MonoBehaviour
{
    2 references
    Vector3 StartingPosition;
    1 reference
    [SerializeField]  Vector3 MovementVector;
    1 reference
    [SerializeField] [Range(0,1)] float MovementFactor;
```

```
0 references
void Start()
{
    StartingPosition = transform.position; // This is the current position

}
```

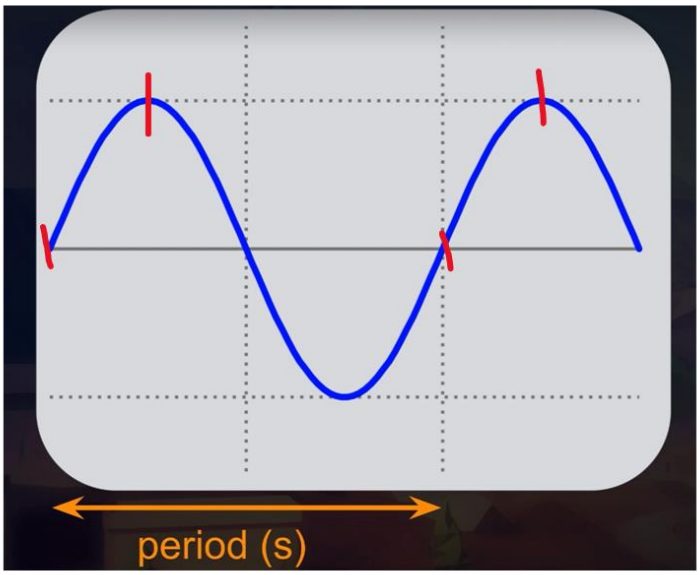
```
void Update()
{
    Vector3 offset = MovementVector * MovementFactor; // First Calculate the Offset.
    transform.position = StartingPosition + offset; // After that change the transform.position to the new position
}
```

- **Mathf.Sin() for Oscillation:**

  https://docs.unity3d.com/ScriptReference/Mathf.Sin.html

What is amplitude?

⇨ It is the total height of the wave and that can be seen there.



What is a Period?

⇨ It is the amount of time before it repeats itself.
As long as you look at the repeating unit.

What is tau?

It represents the circle constant r = 2 * 3.14(PI)

Code Implementation:

```
2 references
Vector3 StartingPosition;
1 reference
[SerializeField]  Vector3 MovementVector;
// [SerializeField] [Range(0,1)] float MovementFactor;
2 references
float MovementFactor;
1 reference
[SerializeField] float period = 2f;
// Start is called before the first frame update
0 references
void Start()
{
    StartingPosition = transform.position; // This is the current position

}
```

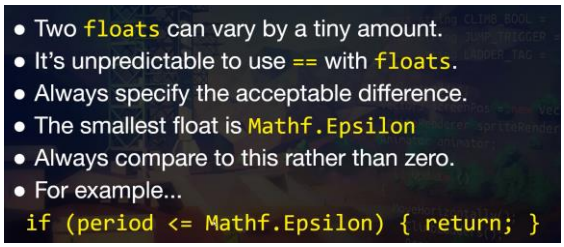
```
// Update is called once per frame
0 references
void Update()
{
    float cycles = Time.time / period; // Continually growing over time

    // Const is a value that doesn't change
    // It represents the circle constant
    const float tau = Mathf.PI * 2; // Constant value of 6.283 (3.14 * 2)

    float rawSinWave = Mathf.Sin(cycles * tau); // Recalculated to go from 0 to 1 so its cleaner

    MovementFactor = (rawSinWave + 1f) / 2f;


    Vector3 offset = MovementVector * MovementFactor; // First Calculate the Offset.
    transform.position = StartingPosition + offset; // After that change the transform.position to the new position
}
```

- **Protecting Against Nan Error:**

  Nan stands for **not a number.**

When the period is 0 we get Nan error because 10/0 cannot be divided by 0.

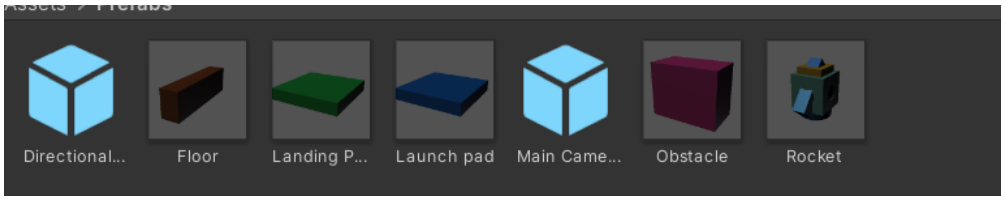
```
transform.position assign attempt for 'Obstacle' is not valid. Input position is { NaN, NaN, NaN }.
UnityEngine.Transform:set_position (UnityEngine.Vector3)
```



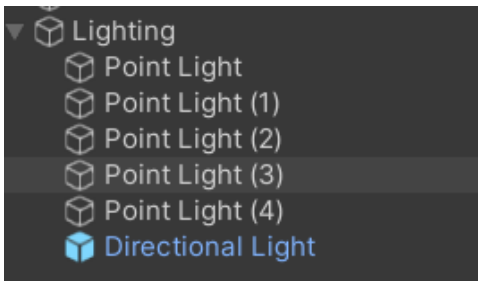
```
void Update()
{
    if(period <= Mathf.Epsilon) {return;} // Removes the NaN Error message
    float cycles = Time.time / period;  // Continually growing over time


    // Const is a value that doesn't change
    // It represents the circle constant
    const float tau = Mathf.PI * 2; // Constant value of 6.283 (3.14 * 2)

    float rawSinWave = Mathf.Sin(cycles * tau); // Recalculated to go from 0 to 1 so its cleaner

     MovementFactor = (rawSinWave + 1f) / 2f;


    Vector3 offset = MovementVector * MovementFactor; // First Calculate the Offset.
    transform.position = StartingPosition + offset; // After that change the transform.position to the new position
}
```

- **Designing Level Moments:**

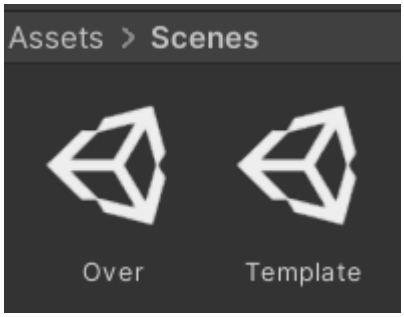
Prefab the **main camera** and **directional light** like in the picture below.



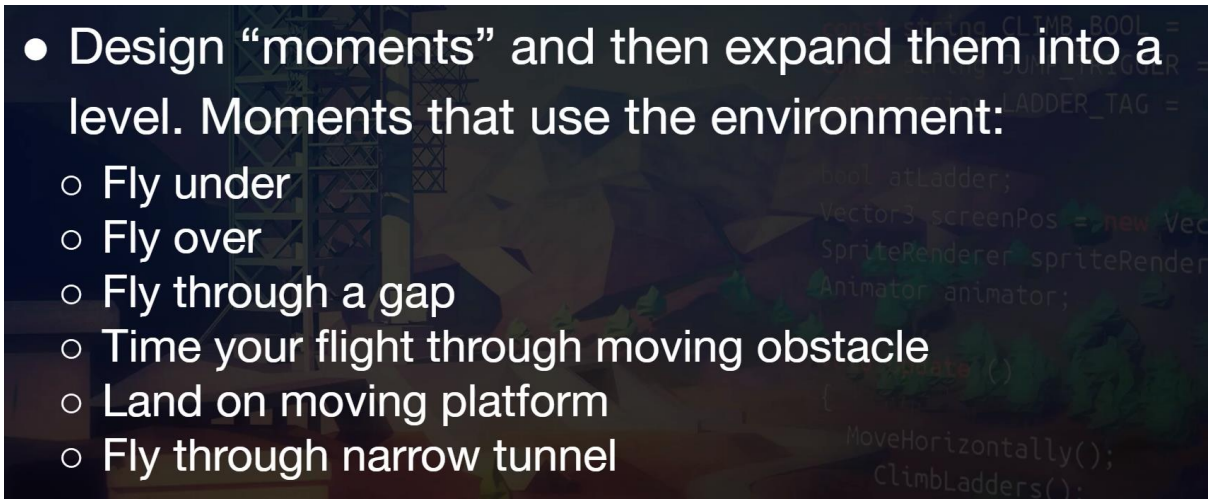
Group all lights into a one-parent file called Lighting.

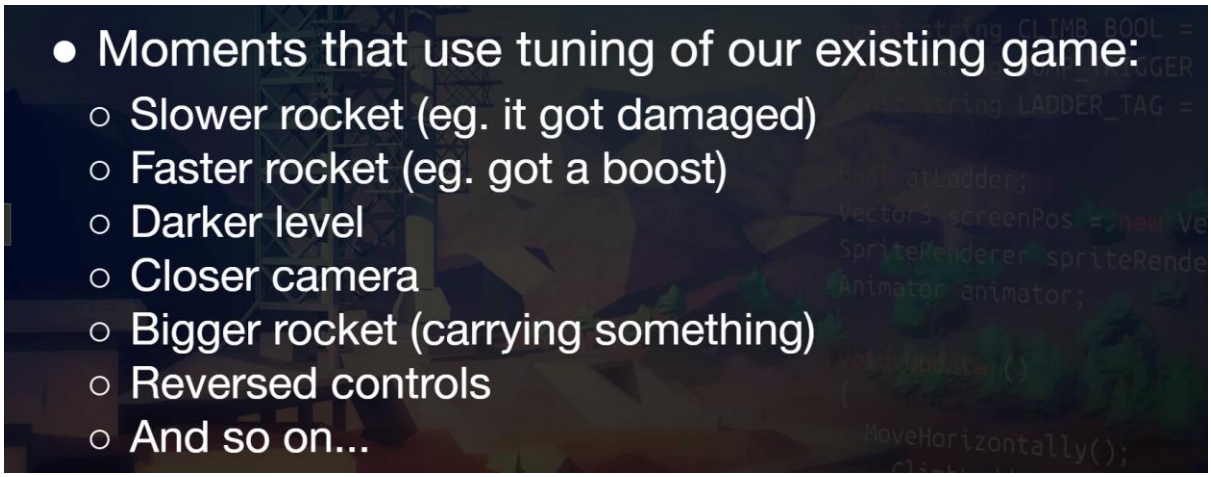


In the Scenes folder duplicate Over and name it as Template.



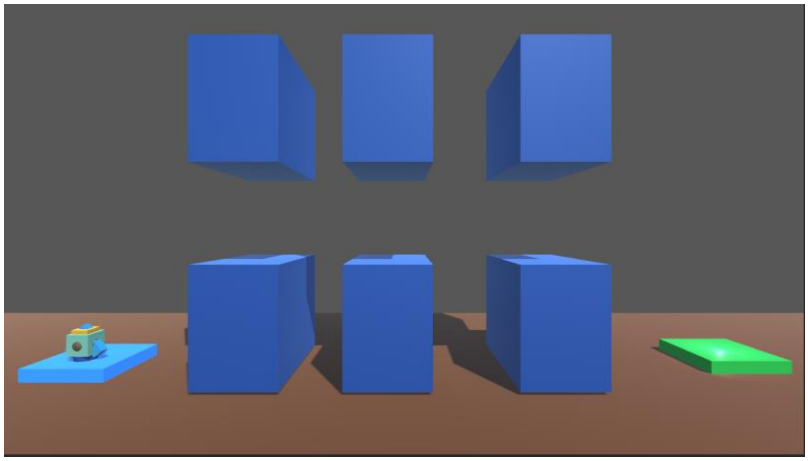
Useful Game Design Approach

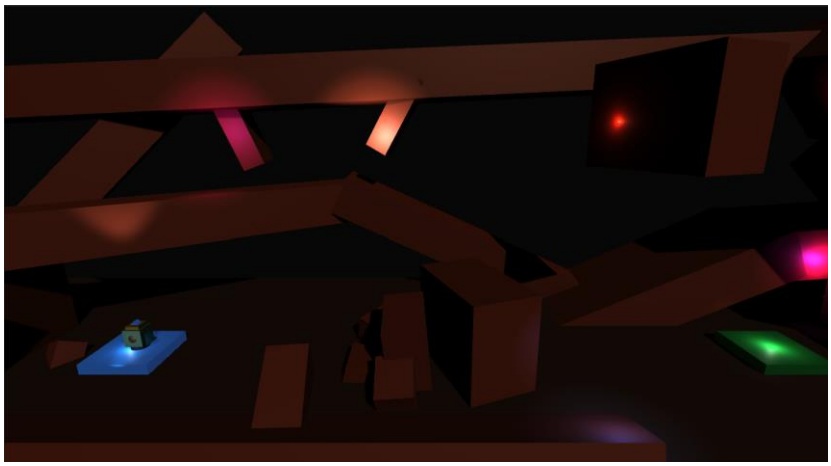


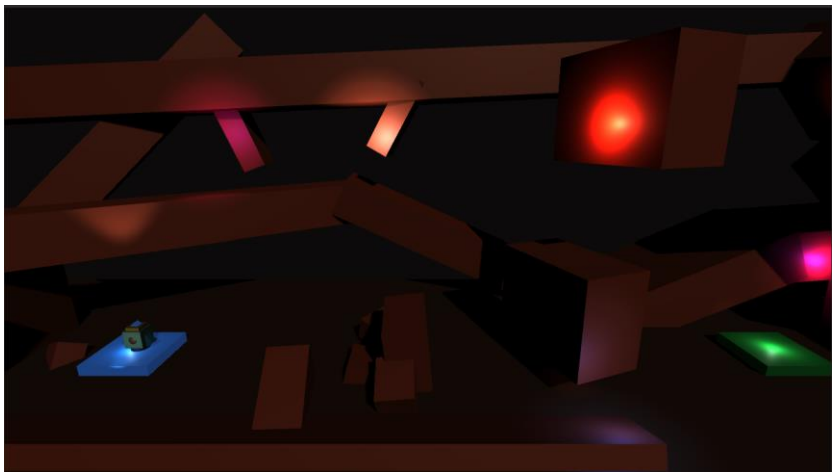


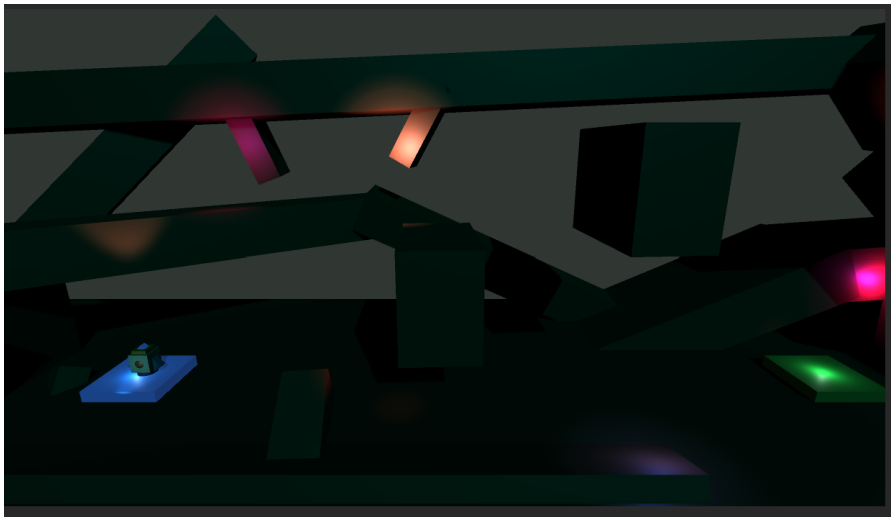
Changing the aspect ratio to 16:9 is advisable to avoid a stretch kind of problem. The 16:9 ratio keeps the screen static/fixed.

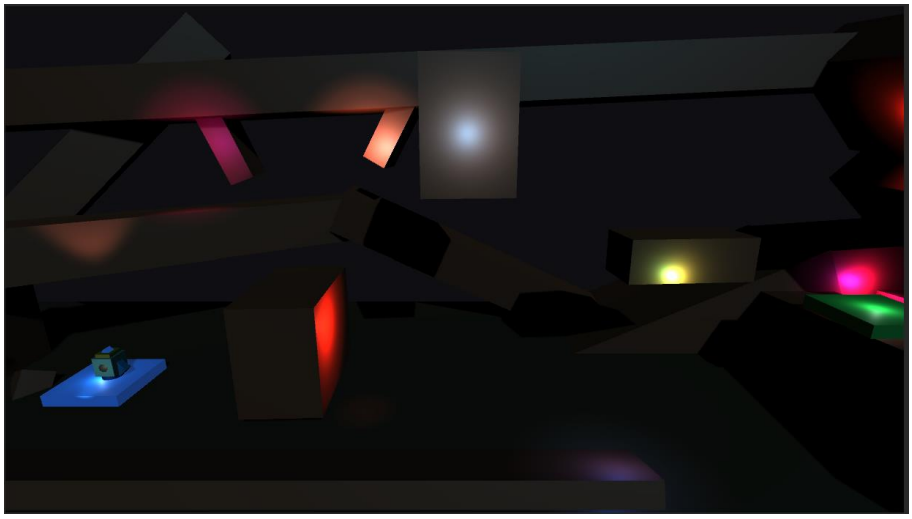
<u>Basic levels created.</u>











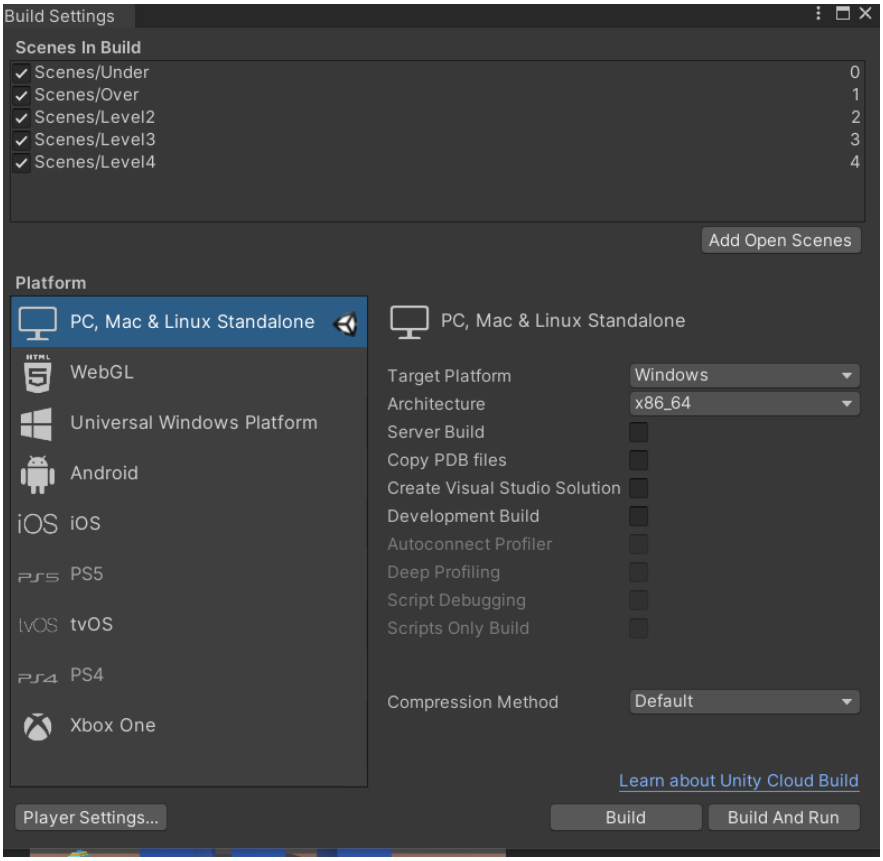
- **Quit Application:**

```
public class QuitApplication : MonoBehaviour
{
    // Update is called once per frame
    0 references
    void Update()
    {
      if(Input.GetKeyDown(KeyCode.Escape))
      {
      Application.Quit();
      Debug.Log("Escape button is Clicked");
      }
    }
```

⇨ Put this code in the Rocket asset, by opening the prefab and attaching it.

- **How to Build and Publish a Game:**

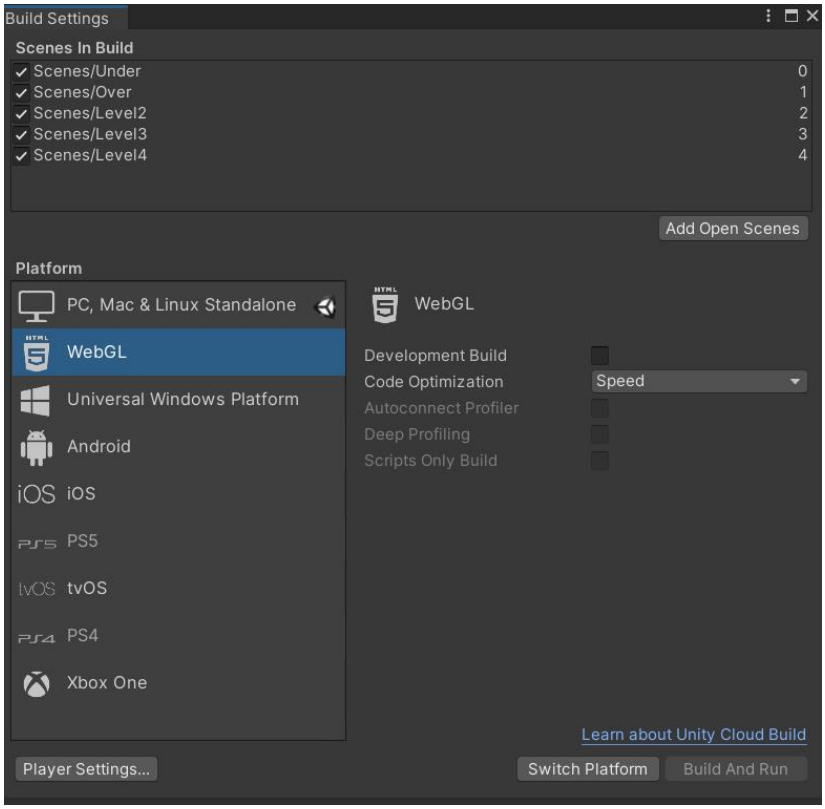
Step 1: Go to Build Settings



Step 2: After that click on Build

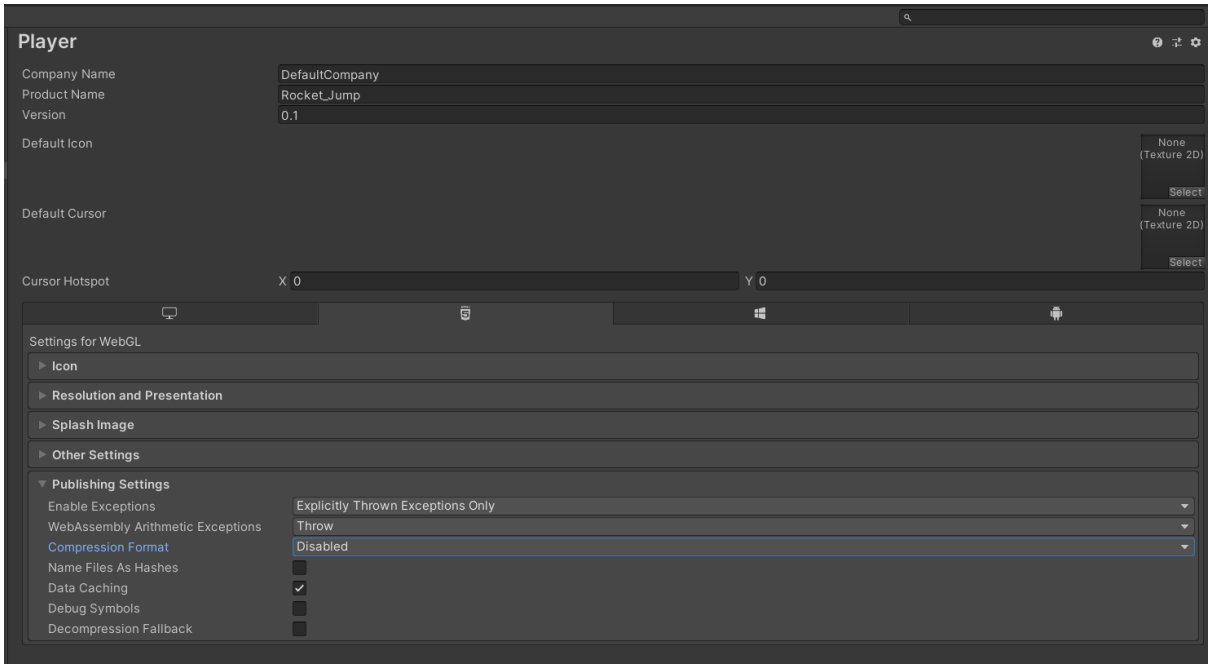
For Webgl option:

Click on the WebGl after that switch the platform. But make sure you install the build first.



Step 2:

If you're using the Unity 2020 version go to project settings and make the compression format default.

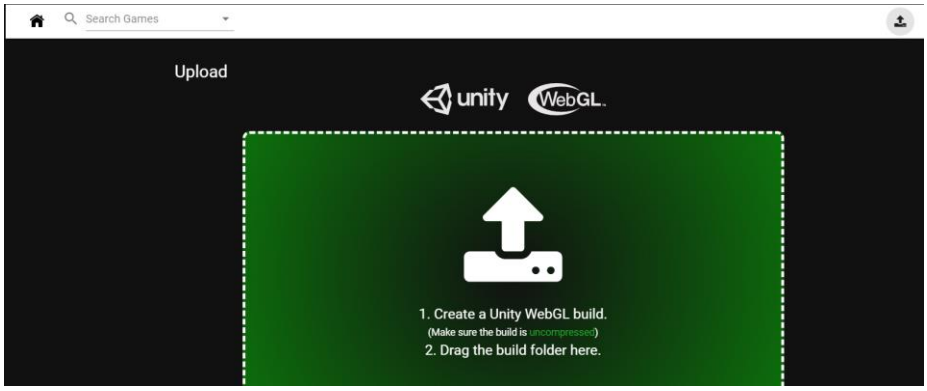


After that click on build. The game will be stored in that specific folder.

HOW TO PUBLISH THE GAME IN THE WEBSITE

Step 1: Login via this website https://sharemygame.com/

Step 2: Click the Upload Button



Step 3: Drag and Drop the WebGl build

Step 4: Give the Project Name and Enable the option "Consider this game for front page" after that click save upload.

Step 5: Test your game

Note: