Wednesday 30th of Mar 2022

Unity WORKSHOP

Unity Engine Fundamentals

Prepared by the Artificial Intelligence Research Center (AIRC), Ajman University, Ajman, UAE

Workshop Outlines

- ~~Workshop Overview~~
- ~~Project Demo~~
- ~~Introduction to Unity 3D Engine~~
- ~~Basic Game Units~~
- ~~Community and Support~~
- ~~Introducing Prefabs~~
- ~~Your First Script~~
- ~~Basic Input Controlling~~
- Practicing Methods
- Physics Laws
- Rocket Vertical movement
- Rocket Rotation
- Rocket Movement and Rotation Constraints
- Unity Audio Introduction
- Play Audio Clip
- Objects Collision
- Switch Statement
- Swap between Scenes (Game Levels)
- Using Invoke
- Multiple Audio Clips
- Introduction to Unity Assets Store
- Make Rocket Look Spiffy
- Practicing with Particles
- Move Obstacles
- Quit Application
- Design The Game Levels
- Cinemachine and Camera
- Build and Publish the game

Day 2 Outlines

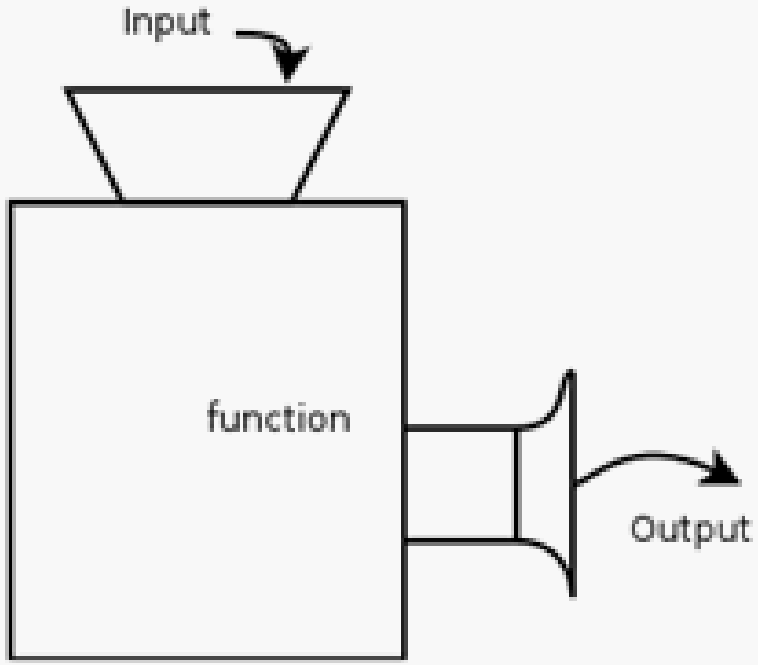
- Practicing Methods
- Physics Laws
- Rocket Vertical movement
- Rocket Rotation

1. Practicing Methods

Organize your code into methods

What is a Method/Function

- An essential part of any programming language
- Containing a block of code that **performs a specific task**.
- Functions serve as organization tools that **keep your code neat and tidy**. In addition, functions make it easy to **reuse the instructions** you've created as needed with different data



unity

Method Declaring VS Calling

Declaring

```
<return type> methodName(<parameters>)
{ //start of the method code
   // body of the method
}//end of the method code
```

Calling

```
methodName(<parameters>)
```

Method Declaration Options

- Return Type :
  - Void : The method returns nothing

```
void MethodName()
{
    //body of the method
}
```

  - Datatype: The method returns a variable of a datatype e.g. int, float

```
float MethodName()
{
    //body of the method
    float x=5f;
    return x;
}
```

- Parameters :
  - List of parameters: must use the same parameters with same type, order in the declaration and calling of method
  - No parameters

```
float MethodName(int x, float y)
{
    //body of the method
    return 2*y;
}
```

Examples of Methods

Declaring

```
Void CleanYourRoom()
{
    //Things to do
}
```

Calling

```
CleanYourRoom()
```

Examples of Methods

Declaring

```
bool CleanYourRoom(int time)
{
    //Things to do
    If (done within time)
        return true;
    else
        return false;
}
```

Calling

```
CleanYourRoom(5)
```

C# Methods : Practice! (1/2)

- Create void method named "ProcessThrust" and call it in the update method
  - Name : ProcessThrust
  - Parameters : None
  - Return type : Void
  - Method body : if space key is hit log the message "Thrusting"
  - Call it in the body of update method

C# Methods : Practice! (2/2)

- Create void method named "ProcessRotate" and call it in the update method
  - Name : ProcessRotate
  - Parameters : None
  - Return type : Void
  - Method body : if "A" key is hit log the message "Rotate Left", else if "D" key is hit log the message "Rotate Right"
  - Call it in the body of update method

2. Physics Laws

Let the rocket Fly !

Rigidbody

- Rigidbodies enable your Game Objects to act under the control of physics.
- Steps :
  - Add a Rigidbody component
  - Check Rigidbody properties
  - Run the game

https://docs.unity3d.com/Manual/class-Rigidbody.html

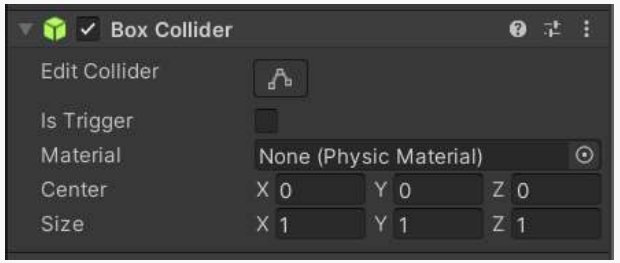
Colliders

- Collider components define the shape of a Game Object for the purposes of physical collisions.
- Steps :
  - Add a collider
  - Check the collider types
  - Control the collider size and position
  - Run the game again

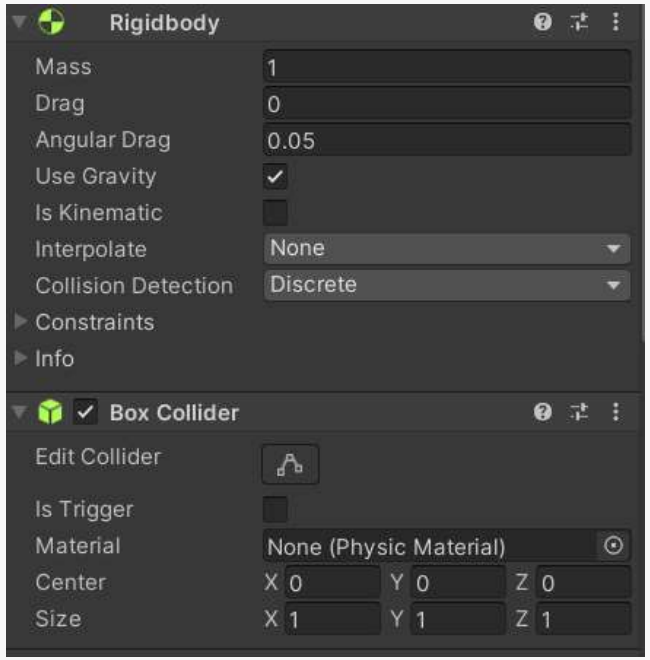
https://docs.unity3d.com/Manual/CollidersOverview.html

Colliders and Rigidbodies

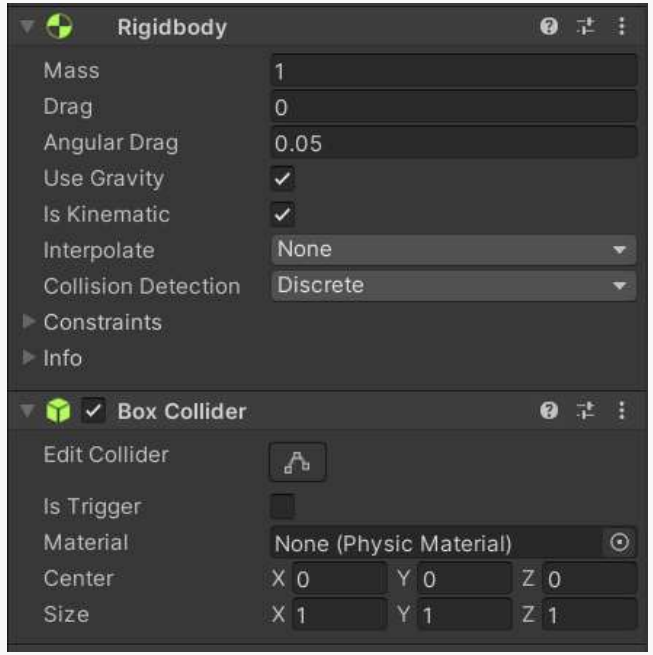
Static Collider



Rigidbody Collider

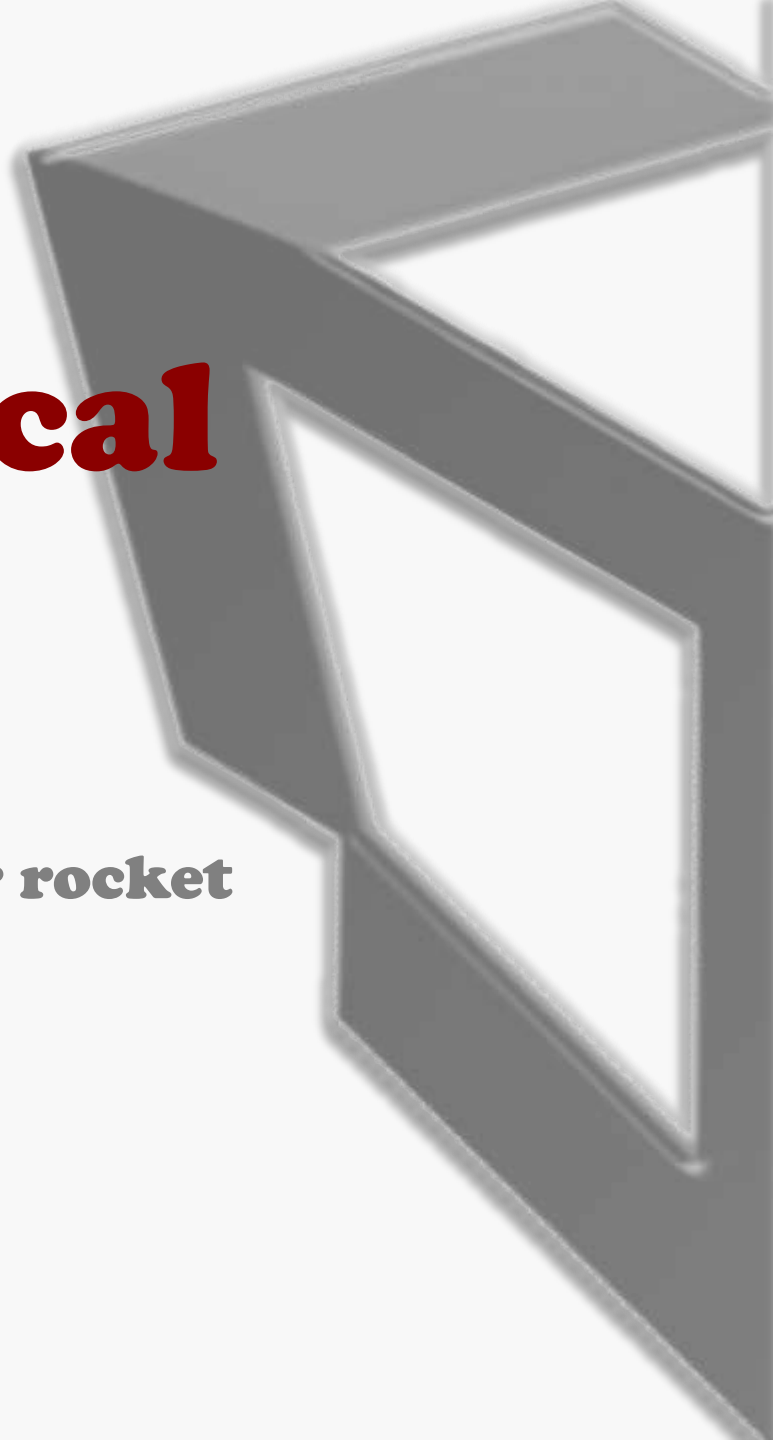


Kinematic Rigidbody Collider



3. Rocket Vertical movement

Control the vertical movement of your rocket



Access Component of GameObject

- Why to use it ?
  - To control the object and modify its component settings during the run of the game
- How to use it ?
  - GetComponent<ComponentType>()

AddRelativeForce method of rigid bodies

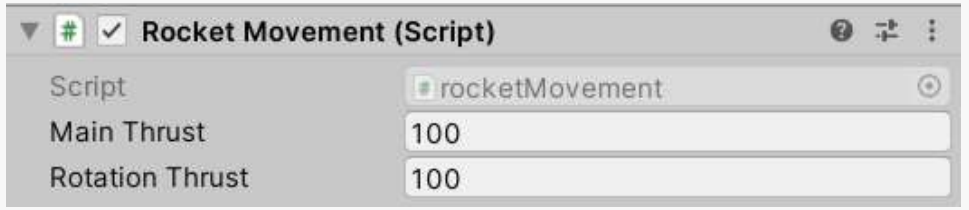
- Why to use it ?
  - Adds a force to the rigidbody relative to its coordinate system (Launch the rocket).
- How to use it ?
  - Access the rigid body component
  - Use the AddRelativeForce method of rigid body

```
Rigidbody_var.AddRelativeForce(force vector)
```

SerializedField Variables

Why to use it ?

- Create a variable and be **able to modify its value from the inspector (Tune a value)**



How to use it ?

[SerializeField] datatype variable_name=<value>;

```
[SerializeField] float mainThrust=100f;
```

Frame Rate independent execution

- Why to use it ?
  - Different devices run your game at different speed
- How to use it ?
  - Multiply the movement by the frame rate(Time.deltaTime)

Frame per second	10	100
Duration of frame	0.1 s	0.01s
Distance per second	1	1

Rocket Vertical movement

- Access the **rigid body** component
- More specifically the **addrelativeforce** fields
- Use **Vector3.up** In order to move it with y-axis
- Add **mainThrust** variable and make it serialized
- Make it indepentened on the slow/fast computers using **Time.deltaTime**

4. Rocket Rotation

Add left and right rotation ability to our rocket

Rotation – Z axis

- Access the **Transform** component
- More specifically the **rotation** fields
- Use **Vector3.forward** → In order to move it with z-axis
- Add **RotationThrusting** variable and make it serialized
- Make it indepentened on the slow/fast computers using **Time.deltaTime**