**UNITY:**

1. Go to assets and create c# script.
2. Then open it in monodevelop.

* **Mest Renderer:** Gives the 3D object its physical shape and properties like cast shadow, receive shadow.
* **Material:** Used to change color of the object.
  + **Create your own material:** 
    - Go to project window
    - Right click on asset
    - Create 🡪 material
* **Transparency:** Transparency is heavy to render. To make an object transparent:
  + Decrease alpha color of the material
  + Rendering mode of the material 🡪 Transparent
* **Light:** 
  + **Directional:** They are very far away from the scene (Like sun). So changing the direction (x-y-z) doesn’t affect the intensity of the light on the object.
    - Rotation affects the light
  + Spot: Used as a spot light
  + Point: Used as a bulb
* **Particle System:** Similar to other objects.
  + If placed inside another object, the position becomes relative to that object.
  + **Gravity Modifier:** Applies gravity to the particles
  + **Simulation space:**
    - Local: The particles follow the particle systemgame object.
    - World: The particles are independent.
  + Rate: Number of partilcles over time.
  + Angle:
* **Rigid body component**: Adds physics to the object.
* **Assets:** There are game packs that we can use to set the environment for our game. You don’t need to re-invent the wheel.
  + **Window🡪 AssetStore**
* **Script: Add as a component to the game object.**

Code:

using UnityEngine;🡪 code will use something already implemented in unity.  
using System.Collections;🡪 code will use certain codes already written in c#.  
  
public class cube : MonoBehaviour 🡪 name of class is cube which extends mono behavior.  
{  
  
    public float sizeModifier = 2.5f;-------  
    public string newName = "Cubey"; |----same as java  
    public bool isRotated = false;----------

*//used for initialization*

void Start(){  
        Debug.Log ("Started!");---when play button is pressed, ‘started!’ in console  
    }  
  
    *//updates once per frame*  
    void Update(){  
        Debug.Log ("Updating");----constantly displays ‘updating’ in console.

60 frames/second = update method is called 60 times each second.   
    }

}

void Update(){  
        transform.localScale = Vector3.one\*sizeModifier;  
    }

updates the scale of cube while in the game.

string improveName(string originalName){  
        return("-[" + originalName + "]-"); ----defining our own method.  
    }

void Start(){

transform.name = improveName(newname);

}

It used the our method with newName as a parameter. Then when play is clicked, the name is changed to whatever name we have set, with a little decoration on it.

void Update(){

transform.localScale = Vector3.one\*sizeModifier;

if(isRotated)

{

transform.localEulerAngles = Vector3.one \* angle;

}

}

This helps update the rotation if is rotated is checked.

public GameObject cubePrefab;----datatype gameObject, variable name = cubePrefab

public int cubeNumber = 10;---

|----we can change value through unity itself.

public float cubeArea = 10f;---

// Use this for initialization

void Start () {

for(int i = 0; i<cubeNumber; i++)

{

GameObject cubeObject = Instantiate(cubePrefab); creating and object named cubeObject, and starting cubePrefab. We have stored a cube in unity in the cubePrefab variable. SO , this will create cube.

cubeObject.transform.position = new Vector3(

Random.Range(-cubeArea, cubeArea),

Random.Range(-cubeArea, cubeArea),

0);----this is just to transform the position of cubes in x, y and z direction.

}

}

**INPUT:**

if (Input.GetKey("space"))

{

Debug.Log("You pressed space");

}

GetKey is used to identify the key. It gives Boolean value. In 60 frames per second, an update happens 60 times. So, if we press space one time, then the message is displayed 6o times.

if (Input.GetKeyDown("space"))

{

Debug.Log("You pressed space");

}

Here, the update happens only when the key is pressed.

Public float speed = 2.5f;

void Update () {

if (Input.GetKey("right"))

{

transform.position += Vector3.right \* speed\*Time.deltaTime;

}

if (Input.GetKey("left"))

{

transform.position += Vector3.left \* speed \* Time.deltaTime;

}

+= because we need to add the direction each time the button is pressed.

Vector3.right means (1,0,0)

Time.deltaTime gives us how many seconds have passed since the last update. This is useful for making the experience of users same no matter what the performance for their devices are.

To move the player up, we could use

if (Input.GetKey("up"))

{

transform.position += Vector3.up \* speed \* Time.deltaTime;

}

But the player will not follow the law of physics. So, we add a component called rigid body to the player. But if we continuously press up button, the player will be on air. So, we use

public float jumpingForce = 300f;

if (Input.GetKey("space"))

{

GetComponent<Rigidbody>().AddForce(0, jumpingForce, 0);

}

We get the rigidbod component and add force to it (vertical force of magnitude ‘jumping force’). If we keep on holding the button, the player will be on air. IF we use GetKeyDown, the player will still be on air if we press space button repeatedly. So, in order to not let player jump more than once before it hits the floor, we use following code:

private bool hitFloor = true;

if (Input.GetKeyDown("space") && hitFloor)

{

hitFloor = false;

GetComponent<Rigidbody>().AddForce(0, jumpingForce, 0);

}

void OnCollisionEnter(Collision collision)

{

if(collision.transform.name == "Plane")

{

hitFloor = true;

}

}

**We called onCollisionEnter method, which detects if an object has collided. Here, we are using if statement to determine if the collision has occurred with an object of name “Plane”. If yes, then the hitFloor is set to true.**

transform.position += Vector3.back \* speed \* Time.deltaTime;

**This makes player move in z-direction. Vector3.forward is also used.**

public float rotatingAngle = 5f;

if (Input.GetKey("right"))

{

transform.RotateAround(transform.position, Vector3.up, rotatingAngle);

}

**RotateAround is a method to rotate an object around. The parameters are**

1. **Rotation point- given by transform.position(x,y,z position of the object)**
2. **Rotatin axis – given by Vector3.up (0,1,0)---rotation on y-axis.**
3. **Rotation angle(float) – given by the variable rotatingAngle.**

**Similarly,**

if (Input.GetKey("left"))

{

transform.RotateAround(transform.position, Vector3.up, -rotatingAngle);

}

**Here, -rotatingAngle so that the cube rotates in opposite direction.**

if (Input.GetKey("up"))

{

transform.position += transform.forward \* speed \* Time.deltaTime;

}

if (Input.GetKey("down"))

{

transform.position -= transform.forward \* speed \* Time.deltaTime;

}

**Here, when we press up, the cube goes forward in the direction in which it is faced. Transform.forward gives the position of the object when it is rotated in a certain direction and moved forward.‘speed’ is a variable which determines the speed of the object. Time.deltaTime is used to make the experience of all users similar by giving the time since the last update.**

**Vector3.forward is a constant which always gives (0,0,1)**

**To change the camera when mouse is clicked:**

public GameObject[] gameCameras;

public int cameraNum;

1. **We create a method that activates a certain camera based on the argument of the method:**

void activeCamera(int cameraNumber)

{

for (int i = 0; i < gameCameras.Length; i++)

{

gameCameras[i].SetActive(i == cameraNumber);

}

}

1. **Then, we create another method that changes camera:**

void changeCamera(int addOne)

{

cameraNum += addOne;

if (cameraNum >= gameCameras.Length)

{

cameraNum = 0;

}

if (cameraNum < 0)

{

cameraNum = gameCameras.Length - 1;

}

activeCamera(cameraNum);

}

**We are adding the ‘addOne’ parameter to the cameraNum. If cameraNum>=4, cameraNum gets back to camera 0. And if cameraNum<0, cameraNum goes to camera 3.Then we use the activeCamera method.**

1. **Then we use update method to update what happens if mouse is clicked**

void Update () {

if (Input.GetMouseButtonDown(0))----- **0 is left mouse click**

{

changeCamera(1);

}

if(Input.GetMouseButtonDown(1))-------**1 is right mouse click**

{

changeCamera(-1);

}

}

**We put 1 because we will add 1 to the cameraNum. -1 is right button is clicked.**

1. **We can add a camera that can chase the player.**

public Transform target; -----transform is a datatype.

void Update () {

transform.LookAt(target);

}

Add a script to a camera. We drag the player to the target variable of the selected camera in unity and then write the above code.

**To make the bullet go forward:**

void Start () {

GetComponent<Rigidbody>().AddForce(Vector3.forward\*bulletSpeed);

}

**The problem with this is that all the bullet goes in the same direction.**

**To move the bullet in different directions, we use following commands:**

**For Bullet Script:**

public class Bullet : MonoBehaviour

{

public float bulletSpeed = 2000f;

public Vector3 shootingDirection;----**datatype is vector3.**

public float lifeTime = 3;

// Use this for initialization

void Start()

{

// **we add force to the rigidbody of magnitude shootingDirection\*bulletSpeed.**

GetComponent<Rigidbody>().AddForce(shootingDirection \* bulletSpeed);

}

// Update is called once per frame

void Update()

{

//decrease lifeTime each second

lifeTime -= Time.deltaTime;

if (lifeTime<= 0)

{

//Destroy gameObject this component is attached to

Destroy(gameObject);

}

}

}

**For player script:**

public class player : MonoBehaviour {

public GameObject bulletPrefab;

// Use this for initialization

void Start () {

}

// Update is called once per frame

void Update () {

if (Input.GetMouseButtonDown(0))//when left mouse button is clicked.

{

//bulletObject is a variable of datatype GameObject which stores the bulletPrefab created by Instantiate method.

GameObject bulletObject = Instantiate(bulletPrefab);

//bulletVariable is a variable of class dataype called 'Bullet'. ‘Bullet’ is a script which is a component of bulletPrefab.So, we are getting the component and storing in the variable bulletObject.

Bullet bulletVariable = bulletObject.GetComponent<Bullet>();

//Then the bulletVariable acceses the shooting Direction of the Bullet component.

//we create new vector for the shootingDirection as it is not initialized yet.

bulletVariable.shootingDirection = new Vector3(

Random.Range(-0.2f, 0.2f),//Determining the range of x-direction

Random.Range(0f, 0.15f),//Determining the range of y-direction

1/\*z-direction\*/).normalized;//for calculations and no roundoff error, we have to normalize the vection. It makes the

//direction with value one.

}

}

}

**normalize sets your vector3's magnitude to 1.  A vector is a combination of magnitude and direction.  Think of normalizing as stripping out the magnitude while keeping the vector3's direction same.  A magnitude of 1 may easily be manipulated, for example, you can multiple a normalized vector3 with some float speed variable so the bullet travels further per frame.**

**For exploding cube prefabs:**

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class ExplosionCubeScript : MonoBehaviour {

//declaring explosion force

public float explosionForce = 300f;

//declaring life time of the exploding cubes

public float lifeTime = 3f;

// Use this for initialization

void Start () {

//initializing direction in which the cubes will move

Vector3 explosionDirection = new Vector3(

Random.Range(-1f, 1f),

Random.Range(-1f, 1f),

Random.Range(-1f, 1f)).normalized;

//randomizing the explosion force

float RandomExplosion = Random.Range(150f, explosionForce);

//getting the RigidBody body component associated with this script and adding force to it.

GetComponent<Rigidbody>().AddForce(explosionDirection \* RandomExplosion);

}

// Update is called once per frame

void Update () {

//each second life time will decrease by 1

lifeTime -= Time.deltaTime;

if (lifeTime <= 0)

{

Destroy(gameObject);

}

}

}

**For Explosion GameObject:**

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class ExplosionScript : MonoBehaviour {

//creating GameObject array named cubePrefabs which will store the prefabs

public GameObject[] cubePrefabs;

//initialing the amount of cubes that will explode

public int amountOfCubes = 3;

// Use this for initialization

void Start () {

//creating loop so that we can instantiate as many cubes as we want

for(int i = 0; i<amountOfCubes; i++)

{

//Instantiating Random cube prefab from array and then storing it in the gameobject variable named storePrefab

GameObject storePrefab = Instantiate(cubePrefabs[Random.Range(0,cubePrefabs.Length)]);

//getting the position of cube prefab and then changing it to the position of the explosion gameObject

storePrefab.transform.position = transform.position;

}

}

// Update is called once per frame

void Update () {

}

}

**Declare collision method:**

public GameObject explosionPrefab;

private void OnCollisionEnter(Collision collision)

{

if(collision.transform.tag == "triggerExplosion")

{

GameObject explosionObject = Instantiate(explosionPrefab);

explosionObject.transform.position = transform.position;

}

}

**Here, we first go to wall and then add new tag named triggerExplosion. Then we use collision method in bulletScript to find the tag. If it hits the tag then, we create explosinObject variable which instantiates explosionPrefab. The explosionPrefab stores the the explosion prefab, which stores the explodingCube prefab. The position of the explosionObject is set to the position of bullet.**

**Another important thing to note is that we can make changes to prefab but we need to hit apply button at top for the changes to be take effect in all prefabs of the same prefab.**

**Virtual reality part 1:**

* **VR needs 2 cameras to simulate both the left and right eyes.**
* **We will will use Unity API which has classes that we can use to make VR games.**
* **Download google vr sdk for unity:**

<https://developers.google.com/vr/unity/>

* **Go to file**🡪 **build setting**🡪 **Android, then player settings** 🡺 **virtual reality supported and then cardboard.**
* **Then import googleVrpackage for unity.**
* **Download google VR sdk and then import the package to unity. After importing, the sdk will be in asset folder. Unity package works as a zip file. There are many files compressed inside a file.**
* **To get two cameras for VR, go to GoogleVr, then prefabs, and then GvrEditorEmulator.prefab.**
* **To get VR reticle, go to GoogleVr, prefabs then GVRreticle.Make it a child of main camera. Add GVREventsSystem in the scene too. When your view clashes with interactive object, the pointer shows up.**

**There was a problem in play mode. The cube doesn’t show up when we use virtual realilty. So we create a new game Object named Player. Then we drop main camera in player. The position of main camera is 0,0,0. The position of player is 0,3.7,-19.4.**

**RaycastHit test:**

**//***variable hit with type RaycastHit*

private RaycastHit hit;

void Update () {

//*calling Raycast method from Physics class.*

*//Transform.position gives the position of source of Raycast*

*//transform.forward gives the direction of Raycast*

*//stores the object hit by raycast in the variable hit.*

if (Physics.Raycast(transform.position, transform.forward, out hit))

{

Debug.Log(hit.transform.name);

}

}

**The camera changes position in play mode so put the camera inside a gameobject. Put the player script inside the camera. The position of gameobject should be 0, 3.27, -19.4. I tried 0,1,0 but it worked. I am not sure what’s the problem but it seems to work.**

**Get the door down**

**NOTE: We don’t put the door cube inside the castle gameobject because it will mess up the position of the door when lowering it. So, just put the door cube outside the castle.**

**Door Script:**

public class Door : MonoBehaviour {

//It will store the final position of the door

public Vector3 loweredPosition;

//It will be used to hold the door to its current position until the player looks at the button.

public Vector3 targetPosition;

// Use this for initialization

void Start () {

//We set the targetPosition to the current position of the door.

targetPosition = transform.position;

}

// Update is called once per frame

void Update () {

//the position of the door is moved to the the targetposition with animation. Until the lowerDoor method is called the targetposition will

//be the same as the current position of door.

transform.position = Vector3.Lerp(transform.position, targetPosition, Time.deltaTime);

}

//Once this method is called, the targetPosition is set to the loweredPosition which will let the update method to lerp the door to the loweredposition.

public void LowerDoor()

{

Debug.Log("accessed LowerDoor");

targetPosition = loweredPosition;

}

}

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Button Script:**

public class Button : MonoBehaviour {

//variable with type of Door

public Door door;

//variable with type PlayerObject

public PlayerObject playerObj;

// Use this for initialization

void Start () {

}

// Update is called once per frame

void Update () {

}

//method to order door to go down and player to go inside castle.

public void OnLook()

{

Debug.Log("accessed OnLook");

door.LowerDoor();

playerObj.moveToCastle();

}

}

Player Script:

public class Player : MonoBehaviour {

//hit is the variable with datatype RaycastHit

private RaycastHit hit;

// Use this for initialization

void Start () {

}

// Update is called once per frame

void Update () {

//Raycast method from Physics class.

//transform.position = source of raycast

//transform.forward = direction of ray

//out hit = store the object hit by raycast in hit variable

if (Physics.Raycast(transform.position, transform.forward, out hit))

{

//the object hit by the raycast is button cube. If it has component Button (script)

if (hit.transform.GetComponent<Button>() != null)

{

Debug.Log("Raycast good");

//get the Button script and set the OnLook() method. The OnLook() method will set the LowerDoor() method. The LowerDoor() method

//will set the value of targetPosition to LoweredPosition.

hit.transform.GetComponent<Button>().OnLook();

}

}

}

}

**MOVE THE PLAYER INSIDE CASTLE:**

**PlayerObject script: We are moving playerobject instead of player because the camera is inside the playerobject and if we move the player itself, the position will get messed up.**

public class PlayerObject : MonoBehaviour {

//used to confirm if the player has entered inside the castle. This is used

//in the GameController script to change the infotext.

public bool enteredCastle = false;

//Store the destination of the player which we set using unity.

public Vector3 PlayerDestination;

//store the current position of the target

public Vector3 targetPosition;

// Use this for initialization

void Start () {

//the current position of the playerobject is stored in targetPosition

targetPosition = transform.position;

}

// Update is called once per frame

void Update () {

//moving the player to the targetPosition which is currently its position.

transform.position = Vector3.Lerp(transform.position, targetPosition, Time.deltaTime);

}

//Once this method is called, the targetPosition is set to the playerDestination and enteredCastle is set to true.

public void moveToCastle()

{

Debug.Log("Move Successful");

targetPosition = PlayerDestination;

enteredCastle = true;

}

}

FOR INFO TEXT TO APPEAR(GameController SCRIPT):

Note: Unity automatically generates lighting when the game restarts. TO avoid it, go to windows, then lighting and uncheck auto.

using System.Collections;

using UnityEngine.SceneManagement;//used for restarting the scene

using System.Collections.Generic;

using UnityEngine;

public class GameController : MonoBehaviour {

//textMesh is a datatype used to store 3d text

public TextMesh infoText;

public TextMesh enemyCount;

//used to get reference to playerObject so that we can know if the player has entered castle

public PlayerObject player;

//all the enemies are inside enemyContainer in the hierarchy which we will use to count the children.

public Transform enemyContainer;

//used to store the enemies remaining

private int enemyRemaining;

//used to set time for restarting game

private float restartTime = 6f;

// Use this for initialization

void Start () {

//show these texts when the game starts

infoText.text = "Hold Alt key and Look at the button to move inside Castle";

enemyCount.text = "Enemies Remaining: ";

}

// Update is called once per frame

void Update () {

//In each update, the enemyRemaining will contain the total number of enemies that are left inside the enemycontainer.

//GetComponentsInChildren<Enemy> will give you the array of elements with script 'Enemy'. We only need lenght, hence ".length"

enemyRemaining = enemyContainer.GetComponentsInChildren<Enemy>().Length;

//In each frame the enemyCount.text is updated

enemyCount.text = "Enemies Remaining: "+enemyRemaining;

//If player.enteredCastle is set to true, the infoText is changed.

if (player.enteredCastle)

{

infoText.text = "Shoot all the enemies.\nPress left mouse button to shoot.";

}

//if enemies remaining is 0 then the infotext is changed.

if(enemyRemaining <= 0)

{

restartTime -= Time.deltaTime;

infoText.text = "Good Job Commandar!" + "\nRestarting the game in:" + (int)System.Math.Ceiling(restartTime);//converting restartTime to int

//loading new scene which is similar to the active scene.

if(restartTime <= 0)

{

SceneManager.LoadScene(SceneManager.GetActiveScene().name);

}

}

}

}

**WHACK A MOLE:**

1. **Put camera with (0,0,0) inside player gameobject with necessary vertices.**
2. **Create table, floor, mole**
3. **After creating one mole, make its prefab.**
4. **Manually duplicate and place each moles at its place.**
5. **To update all moles at once, just select the main mole, then make changes and hit apply.**
6. **Read blender guide for whack a mole art**

**USE THE BLENDER ASSETS IN UNITY:**

1. **Import asset by right clicking in asset area**
2. **Hide molecontainer**
3. **Drag and drop the whackAMole box in the table**
4. **Delete all table objects**
5. **Then unhide the molecontainer and use top view and select y-axis to look from top. Then position every mole in their proper holes**
6. **Set original height and hidden height of all moles in unity**
7. **Drag the mole into hierarchy🡪 Put it inside one of the mole prefab🡪 Then set its position (0,0,0)🡪 resize it to fit the prefab, then remove the mesh renderer and mesh filer, hit apply.**
8. **Import hammer. Set it’s position where your hand would be. Then use hammer logic.**

**BUILD FOR ANDROID**

1. **Build settings🡪 switch to android🡪 add open scene**
2. **Player settings🡪 VR supported**
3. **Other settings🡪 identification 🡪 package name = com.vr.whackAmole(any name is ok).**
4. **Scroll down🡪 minimum API level 🡪 level 21**
5. **Build**