<u>LevelComplete.cs</u>

```
using UnityEngine;
using UnityEngine.SceneManagement;
public class levelcomplete : MonoBehaviour
{
    public void loadnextlevel()  //Loads next level after current level is completed.

    {
        SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex + 1); //loads
next level
    }

    public void loadmainmenu() //Loads mainmenu after gameover UI
    {
        SceneManager.LoadScene("gamestart");
    }
}
```

GameManager.cs

```
using UnityEngine;
using UnityEngine.SceneManagement;

public class gamemanager : MonoBehaviour
{
    bool gamehasended = false; //flag to track if gamehas ended,to prevent multiple call
of Game over
    public float restartDelay = 1f;
    public GameObject completelevelUI;
    public GameObject gameoverUI;

    //Activates LEVEL COMPLETE UI
    public void completelevel1()
    {
        //Debug.Log(" LEVEL WON");
        completelevelUI.SetActive(true);
    }


    //Activates GAME OVER UI.
    public void endgame()
    {
        if (gamehasended == false)
        {
            gamehasended = true;

            Debug.Log("GAME OVER");
            gameoverUI.SetActive(true);
            //Invoke("Restart", restartDelay);
        }
    }

    //void Restart()
    //{
        //SceneManager.LoadScene(SceneManager.GetActiveScene().name);
     //   SceneManager.LoadScene("gamestart");
    //}




}
```

LevelCompleteCollision.cs

```
using UnityEngine;

public class levelcompletecollision : MonoBehaviour
{
    void OnCollisionEnter(Collision collisioninfo)
    {
        //Detects if last object has been dodged by the player hence game won and calls
completelevel().

         if (collisioninfo.collider.tag == "obstacle")
         {
            //Debug.Log("hit somethng");
             FindObjectOfType<gamemanager>().completelevel1();
        }



    }

}
```

LoadSceneScript.cs

```
using UnityEngine;
using UnityEngine.SceneManagement;

public class loadSceneScript1 : MonoBehaviour
{

    public static bool gameispaused = false;
    public GameObject menuUI;     //container of UI


    public void newgame()  //Loads level 1 of the game when called.
    {
        SceneManager.LoadScene("level1");
    }
    public void options()  //Loads the options screen when called.
    {
        menuUI.SetActive(true);
    }

    public void quit()  // Quits the game when called.
    {
        Application.Quit();
    }


    public void mainmenu() //Loads the mainmenu when called.
    {
        Time.timeScale = 1f;
        SceneManager.LoadScene("gamestart");
    }




    public void PauseGame()  //Pauses the game by setting timescale = 0 and activating
Pause Panel.
    {
        Time.timeScale = 0f;
        menuUI.SetActive(true);
        gameispaused = true;

        // SceneManager.LoadScene("gamepaused");
    }

    public void ResumeGame()  //Resumes the game by setting timescale = 1 and
deactivating Pause Panel.
    {
        Time.timeScale = 1;
        menuUI.SetActive(false);
        gameispaused = false;
       // SceneManager.LoadScene("level1");

    }

    public void Update() //function to keep track whether ESC (pause) is pressed or not.
    {
        if (Input.GetKeyDown("escape"))
        {
            if (gameispaused)
            {
                ResumeGame();
            }

            else
            {
                PauseGame();
            }
        }
    }

}
```

PlayerCollision.cs

```
using UnityEngine;

public class playercollision : MonoBehaviour    // Responsible to detect the losing
conddtion of game.
{


    void OnCollisionEnter( Collision collisioninfo)  //Called automatically when an
obstacle hits the player
    {
        if(collisioninfo.collider.tag == "obstacle")
        {
            FindObjectOfType<gamemanager>().endgame();
        }

    }



}
```

Score.cs

```
using UnityEngine;
using UnityEngine.UI;


public class score : MonoBehaviour
{
    public Text scoretext;
    public Transform lastobject;
    // Update is called once per frame
    void Update()                                   //Keeps updating the score of player
by tracking the position of last obstacle
    {
        //Debug.Log(player.position.z);
        scoretext.text = ((lastobject.position.z-411)*-1).ToString("0") ;

    }
}
```

Obsmovement.cs

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

public class obsmovement : MonoBehaviour
{
    public Rigidbody rb;
    public float backforce = 2000f;
    // Update is called once per frame
    void FixedUpdate()                     // Responsible for motion of obstacles, 
providess force
    {
        rb.AddForce(0, 0, - backforce* Time.deltaTime);
    }
}
```

AvatarControllerClassic.cs

```
using UnityEngine;
//using Windows.Kinect;

using System;
using System.Collections;
using System.Collections.Generic;
using System.Runtime.InteropServices;
using System.IO;
using System.Text;



// Avatar controller is the component that transfers the captured user motion to a
humanoid model (avatar). Avatar controller classic allows manual assignment of model's
rigged bones to the Kinect's tracked joints.

public class AvatarControllerClassic : AvatarController
{
       // Public variables that will get matched to bones. If empty, the Kinect will
simply not track it.

        public Transform ClavicleLeft;
        public Transform ShoulderLeft;
       public Transform ElbowLeft;
        public Transform HandLeft;
       public Transform FingersLeft;
       public Transform ThumbLeft;
       public Transform ClavicleRight;
       public Transform ShoulderRight;
       public Transform ElbowRight;
       public Transform HandRight;
       public Transform FingersRight;
       public Transform ThumbRight;
       public Transform HipCenter;
        public Transform Spine;
       public Transform ShoulderCenter;
       public Transform Neck;
       public Transform HipLeft;
        public Transform KneeLeft;
       public Transform FootLeft;
       public Transform HipRight;
        public Transform KneeRight;
       public Transform BodyRoot;



// map the bones to the model.
       protected override void MapBones()
       {
               bones[0] = HipCenter;
              bones[1] = Spine;
              bones[2] = ShoulderCenter;
              bones[3] = Neck;
```

```
            bones[5] = ShoulderLeft;
            bones[6] = ElbowLeft;
            bones[7] = HandLeft;
            bones[11] = ShoulderRight;
            bones[12] = ElbowRight;
              bones[13] = HandRight;

            bones[17] = HipLeft;
             bones[18] = KneeLeft;
             bones[19] = FootLeft;

            bones[21] = HipRight;
             bones[22] = KneeRight;
            bones[23] = FootRight;

            bones[25] = ClavicleLeft;
             bones[26] = ClavicleRight;
            bones[27] = FingersLeft;
            bones[28] = FingersRight;
             bones[29] = ThumbLeft;
            bones[30] = ThumbRight;


            bodyRoot = BodyRoot;

        }

}
```