# CM3133: Games Prototyping

Please use Unity version: 2022.3.8f1

## Introduction

Today’s practical class is aimed at looking at triggers, transitions and a little bit of audio work.

## Step one: Grab the Moodle project from last week

We’re not going to start from a blank state here – we’re going to build on what was done before. It doesn’t really matter for this project if you begin from last week or the week before, or even your own project. We’re not changing any of the core functionality of anything here – we’re just adding in some new things.

## Step two: Grab the Assets from the Website and Make them into Sprites

The new asset you’ll be using today is BlackHole.png. You’ll make this into a sprite the same way you did in previous weeks.

Also grab Tardis.mp3 for later.

## Step Three: Create Black Hole Prefab

Step one is to create our black hole prefab. Create a script called BlackHole and attach it to the sprite we created in step two. Give it a polygon collider. Adjust your script so it has the following code:

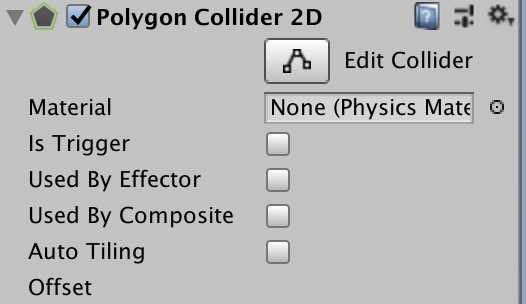
using System.Collections;  
using System.Collections.Generic;  
using UnityEngine;  
  
public class BlackHole : MonoBehaviour {  
    public string whereToGo;  
  
    // Use this for initialization  
    void Start () {    }  
      
    // Update is called once per frame  
    void Update () {  
        this.transform.Rotate(new Vector3(0, 0, 1f));  
    }

}

All this script does for now is cause the black hole to rotate, but we’ll make it do more in a moment. Run your program with this new black hole and see what it does to any asteroids you push into it. Notice how they (should) get trapped in the arms of the polygon as opposed to bounce off of an invisible rectangle or circle.

## Step Four: Our Trigger Zone

Our next step is to turn this collider into a trigger, which we do by setting the ‘is trigger’ checkbox in the inspector for the polygon collider:



Run your project and you’ll find that your black hole no longer blocks objects when they enter its collider – they’re still triggering collisions, but those are handled by a different set of functions. Adjust your black hold script so that it contains a new function:

 private void OnTriggerEnter2D(Collider2D collision)  
    {  
        GameObject gob = collision.gameObject;  
        Rigidbody2D colrig;  
  
        if (gob.tag == "Asteroid")  
        {  
            colrig = gob.GetComponent<Rigidbody2D>();  
  
            colrig.AddForce(this.gameObject.transform.up \* 500);  
            colrig.AddTorque(Random.Range(-10, 10));  
        }  
  
  
    }

Make sure your asteroids are tagged with “Asteroid” since that’s how we’re going to differentiate between asteroids and the player. Tag your spaceship with “Player” so that we can tell when that enters the zone too.

Try this out, and experiment with different values for force and torque. See what happens when you change OnTriggerEnter2D to OnTriggerStay2D or OnTriggerExit2D.

You should find that when Asteroids enter the black hole they get accelerated and put in a new rotation – note the use of AddTorque here rather than Rotate – that adds a persistent amount of angular momentum to the sprite through the physics engine.

## Step Five: A new scene

Create a new scene – it doesn’t matter what you call it, but let’s say for the sake of argument it’s called Scene2. Drop a spaceship prefab in there, along with a black hole prefab. Note that your BlackHole 1 “wheretToGo” and this is the scene to which the player should be moved when they move the ship into the black hole.

For the black hole on your first scene, set it to go to Scene2. For the black hole in your second scene, set it to go back to the first one.

Go to File 🡪 Build Settings🡪Add Open Scenes, to add scene 2 for the project, or you can drag and drop the new scene into the list before it can be accessed with the scene manager, but once you’ve done this you can put in the full code for the Black Hole trigger:

private void OnTriggerEnter2D(Collider2D collision)  
    {  
        GameObject gob = collision.gameObject;  
        Rigidbody2D colrig;  
  
        if (gob.tag == "Asteroid")  
        {  
            colrig = gob.GetComponent<Rigidbody2D>();  
  
            colrig.AddForce(this.gameObject.transform.up \* 500);  
            colrig.AddTorque(Random.Range(-10, 10));  
        }  
  
**if (gob.tag == "Player") {**

**SceneManager.LoadScene(whereToGo);  
        }**  
    }

You will need to add “using UnityEngine.SceneManagement;” on the head of this script, otherwise, the system won’t be able to find Scene Manager.

Run this and you should find that your ship can now cycle through your scenes when it flies into the black hole. But you should also see that your GameState object is lost when you move to the second scene and if you were storing important information in there that would be a problem.

## Step Six: Fixing the Game State object

There are two things we need to do to fix our GameState object. The first is to make sure it doesn’t get destroyed when we move to the next scene and we need to adjust our GameStateManager a little farther by making sure that if try to add a GameState object when one already exists, we just get rid of it.

Add this and you should find your game state persists exactly as you would like.

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class GameStateManager : MonoBehaviour

{

public GameObject prefab;

public int howMany;

int score;

public int getScore()

{

Debug.Log(score);

return score;

}

public void setScore(int s)

{

score = s;

}

public void adjustScore(int s)

{

score += s;

Debug.Log("Score is " + score);

}

// Use this for initialization

void Start()

{

GameObject tmpAsteroid;

AsteroidMove am;

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

{

tmpAsteroid = Instantiate(

prefab,

new Vector3(Random.Range(-12, 12), Random.Range(-4, 4), 0),

Quaternion.identity

);

am = tmpAsteroid.GetComponent<AsteroidMove>();

am.mx = Random.Range(-5, 5);

am.my = Random.Range(-5, -5);

am.rotz = Random.Range(-5, 5);

}

DontDestroyOnLoad(this.gameObject);

if (FindObjectsOfType(this.GetType()).Length > 1)

{

Destroy(this.gameObject);

}

}

// Update is called once per frame

void Update()

{

if (Input.GetKeyUp(KeyCode.UpArrow))

{

adjustScore(1);

}

}

}

## Step Seven: A little bit of sound

Finally, we’re going to have little bit of sound that plays when we move between our scenes. Moving between scenes destroys everything (unless we use a different kind of loading method we won’t talk about just yet) and as such we can’t store the sound where it’s most appropriate – which is to say, on the black hole or the spaceship. If we start a sound playing from then, it’ll stop instantly when the sprites are destroyed.

Instead, we add an AudioSource component to our game state manager and drag a sound clip into the appropriate location:



And then we adjust our BlackHole a little more so that it does the usual code of finding the GameState object, and we’ll trigger the sound file when we move through the black hole:

using System.Collections;  
using System.Collections.Generic;  
using UnityEngine;  
using UnityEngine.SceneManagement;  
  
public class BlackHole : MonoBehaviour {  
    public string whereToGo;  
    **public GameObject gs;**    // Use this for initialization  
    void Start () {  
**gs = GameObject.Find("GameState");**    }  
      
    // Update is called once per frame  
    void Update () {  
        this.transform.Rotate(new Vector3(0, 0, 1f));  
    }  
  
    private void OnTriggerEnter2D(Collider2D collision)  
    {  
        GameObject gob = collision.gameObject;  
        Rigidbody2D colrig;  
  
        if (gob.tag == "Asteroid")  
        {  
            colrig = gob.GetComponent<Rigidbody2D>();  
  
            colrig.AddForce(this.gameObject.transform.up \* 500);  
            colrig.AddTorque(Random.Range(-10, 10));  
        }  
  
        if (gob.tag == "Player") {  
            **gs.GetComponent<AudioSource>().Play();**            SceneManager.LoadScene(whereToGo);  
        }  
    }  
}

## Step Eight: Your tasks

At this point, the assessment has been released so you should be working towards the first submission there – have a think about what game you might want to develop and start sketching out some thoughts for how it might look, what assets you’ll need, and what functionality you’ll need to implement and where.