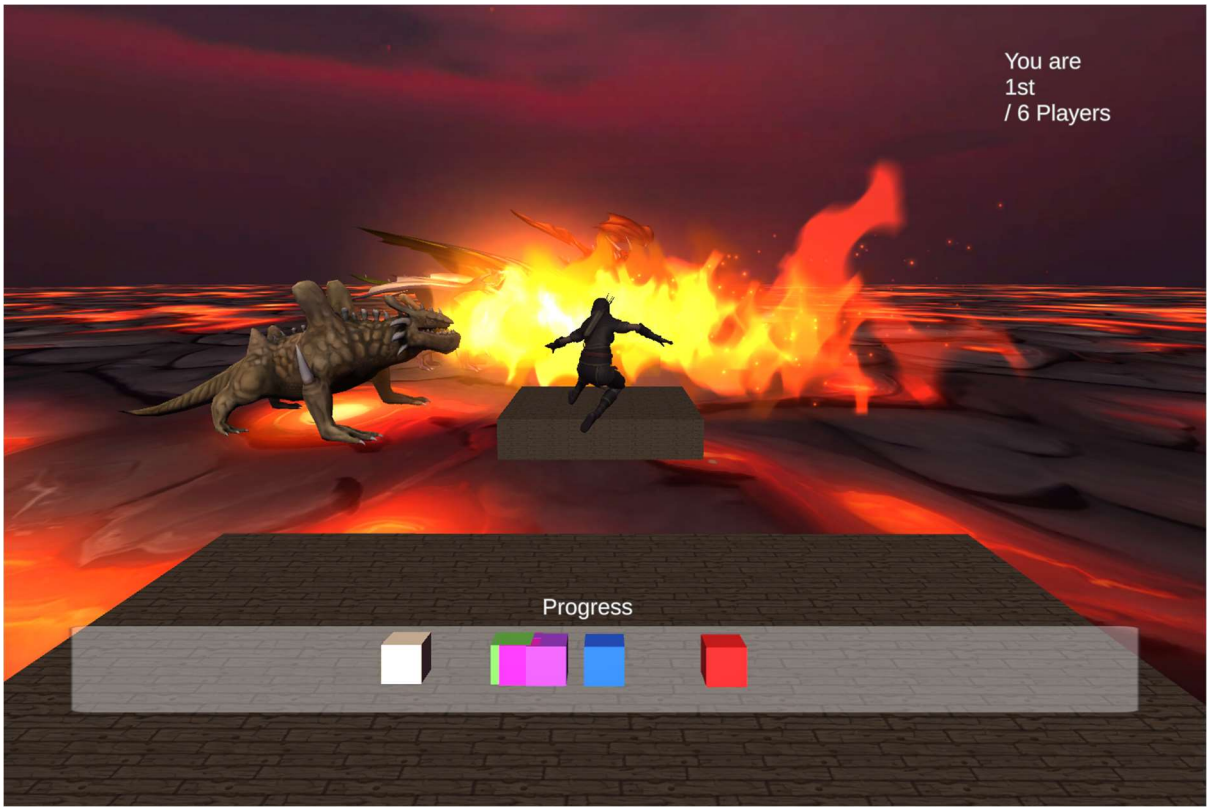
Dragon Run



By Maxwell Maia

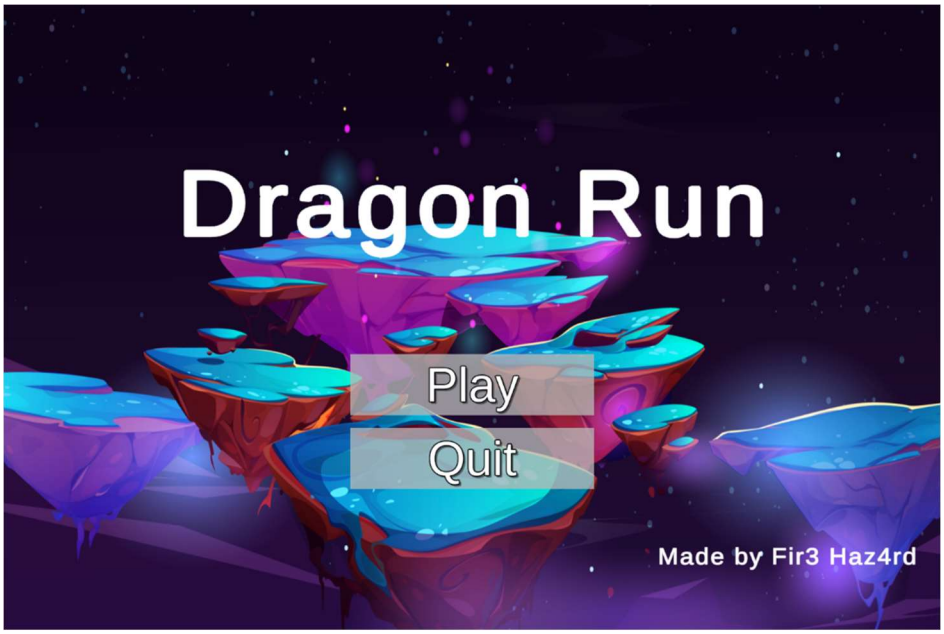
Student ID 21236277

Table of Contents

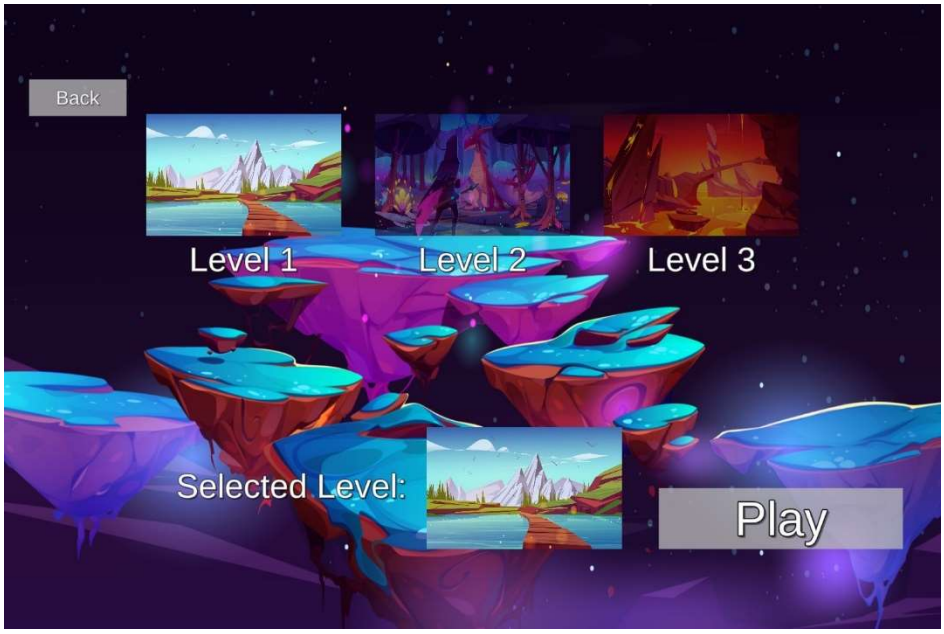
Game Description

Dragon run is a single-player 3rd person 3D platformer race to finish! Race AI opponents through a parkour course that is sprinkled with giant dragons ready to blast you away with their fire-y breaths. Avoid traps and get to the end of the level first!

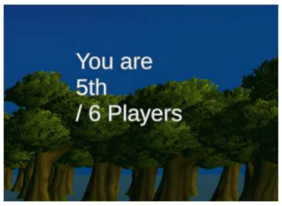
Main Menu



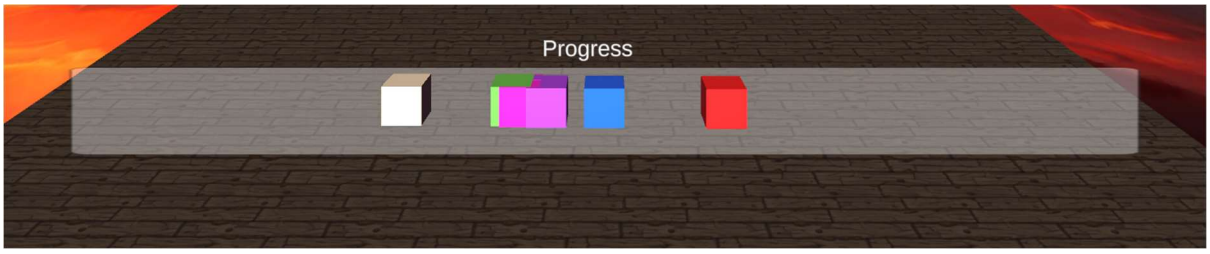
Level Select



The game features a progress tracking system in the form of a position at the top right of the screen. There are 6 players, one is you, the other 5 are AI.



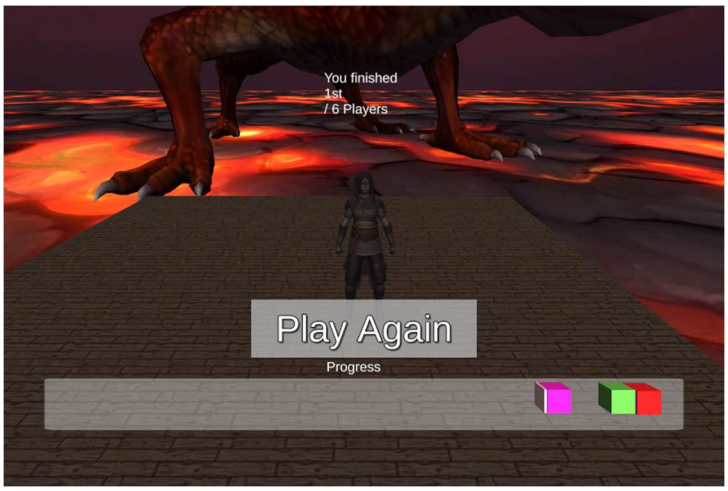
At the bottom of the screen is a Race Progress Bar which represents every player's (player and AI) current progress of the race. You cannot see the AI in the game world, only their progress is represented in the Race Progress Bar. Each AI has a random speed and a random chance to touch a trap. When the race begins, the AI begin moving towards the end of a race. As they move, their cubes in the progress bar move. The AI will randomly "touch a trap" which sets their progress back. You are the red cube, the red cube moves as you move.



In the background, the progress is represented by a floating point number from 0 to 100. Levels may have varying lengths so extra consideration is taken here. When the AIs increase their progress towards the finish line and decrease their progress when touching a trap, these increases and decreases are scaled by the race distance, which is calculated from the start line to the finish line.

The game also features a checkpoint system. If you are hit by one of the dragon's fire blasts or fail a parkour jump and land on the ground below, you will teleport to the last checkpoint that you reached. The checkpoint system initializes itself at when a level loads, so for map makers/level designers there is no need to worry about checkpoint placement. The checkpoint system will make sure to not create checkpoints over a gap/parkour jump.

Post Game Screen



Movement Mechanics:
- 100ms of Coyote time.

- 150ms Pre-jump. A pre-jump is when you press the jump button before touching the ground. As long the player pressed jump within a certain amount of time (150ms) before touching the ground, the game will automatically jump for the player when the player touches the ground.

- Running on the ground adds force to a rigidbody. So the player feels acceleration.

- The player has a speed limiter that limits the player's movement speed (maximum running speed).

- While in the air, the player can move to control themselves in the air (10% of the force of running on the ground). This is useful for slight adjustments while in the air like trying not to overshoot a platform during a parkour jump.

Currently there are 3 levels. Every level's design (placement of traps and platforms) is the same but this could easily be changed by a level designer. The skyboxes and some textures are different to illustrate that these are different levels.

Instructions

Please lower your volume before you open the game as the game might be loud on your computer.

Select a level and click play.

Aim of the game:

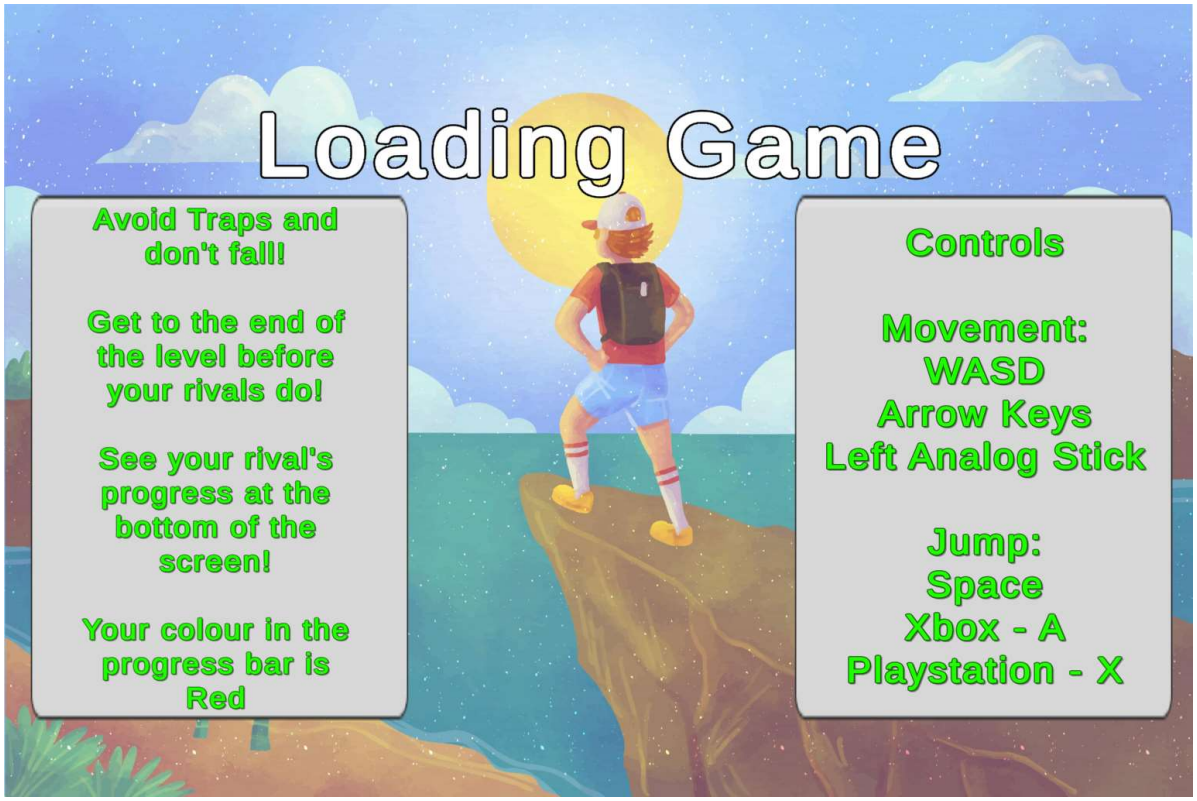
Get the end of the level before your AI rivals do. Avoid the dragon's fire traps and don't fall off of the platforms. You can see your rival's progresses at the bottom of the screen once the race begins. Your progress is represented by the red cube at the bottom of the screen.

Run past the start line to begin the race.

Controls:

Keyboard: WASD to move and spacebar to jump.
Controller: Analog stick to move and "Submit" button which is A on Xbox, and X on PlayStation to jump. (Controller support not tested).

Instructions are available to the player in a fake loading screen as well as in text in world space at the area before the start of the race.

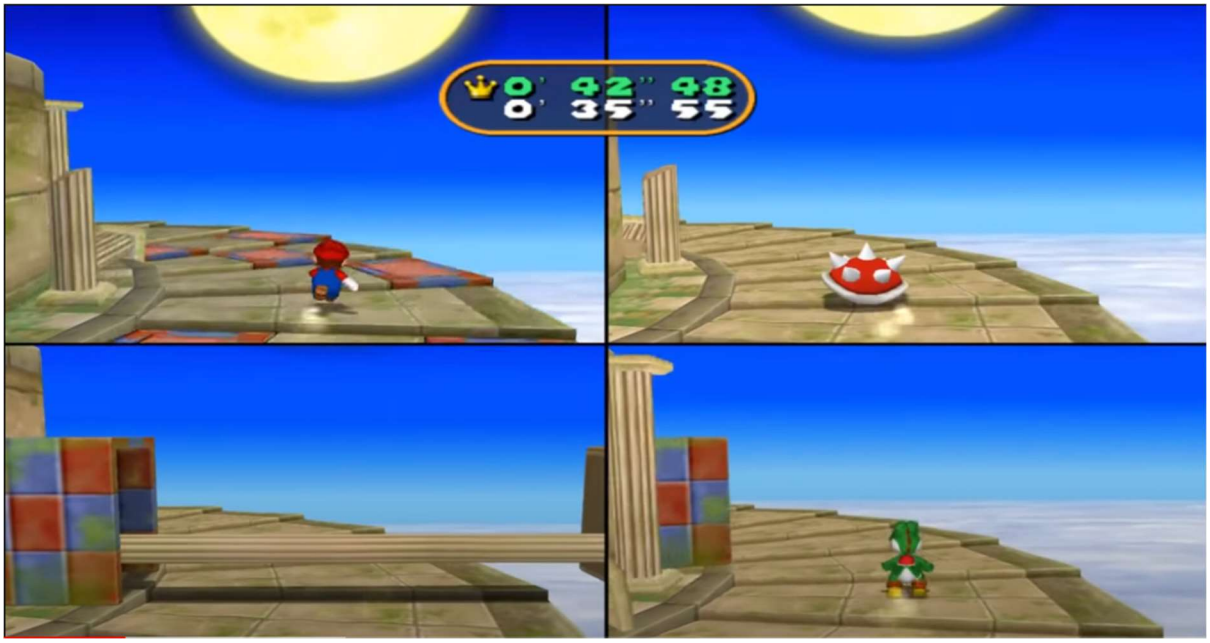


When you get to the end of the race Click Play again to try another level.

Can you come first?

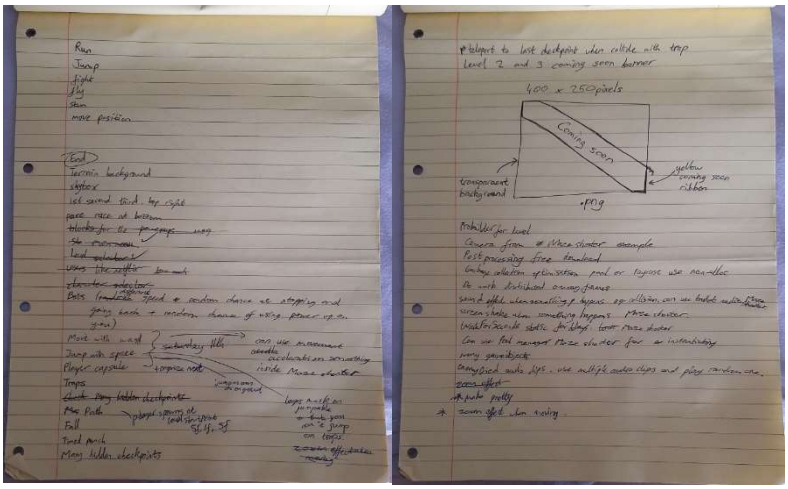
Development

Started by looking for ideas. I wanted to make a game that I could play with my friends if I extended it to multiplayer one day. I would make the single player version of it for my project. I looked at Mario Party 4-player minigames on YouTube. I played this minigame before and found it fun so I decided to make something like it. My version of the game would be a straight path as opposed to the spiral staircase path in Mario Party for simplicity.



The next stage of development was ideation. I would think about the my game over a period of a week and I wrote down ideas that I liked even if it didn't make sense. Eventually I sat down simplified my ideas and cut out the ones that didn't fit. I then wrote down all of the features that I want on a piece of paper.

My early notes. Many features didn't make it in.



Then I started making the thing.

I started with a script for the camera to follow a GameObject. I created a simple scene where a ball rolls down a hill and hits a wall. I used this to test my SmoothCameraMovement script. I used Vector3.SmoothDamp which gradually changes a vector towards a desired goal over time. The vector that I want to change is the camera transform's position vector. The desired goal is the position that we want the camera to be when the player is stationary for a while. This is the player's position plus an offset. I had a problem with camera jitter. The issue occurred because the object's movement is tied to physics. The movement of the camera should be in-sync with the physics updates. Solved by putting camera movement code into FixedUpdate.

I then moved on to creating the Main Menu and Level Select GUI. Straightforward graphic design. I had onclick events on the buttons to change between the GUI's and load a new scene. These were later reworked into GameManager with state transitions. Currently the buttons call methods of the GameManager to change the game's state and then the game state will determine what is shown on screen.

I copied the GameManager from Sam Redfern's MazeShooter into my project and started changing it to how I want it. By the end of the project it doesn't look like MazeShooter's code at all. This gave me a good base to work from since MazeShooter has game states.

In GameManager I added states and a method, SetGameState, that transitions between states also changing which UI elements are active and inactive.

I had an issue: when I load into the a new scene, the GameManager and Canvas would be destroyed so I couldn't keep track of the game state and the UI wasn't available when I started a level. To solve this I made the GameManager object and Canvas object persist into other scenes by adding my Persistent script to the GameObjects. Persistent just uses DontDestroyOnLoad.

I added jumping using a sphere cast. Sphere cast best describes the shape of the player capsule. I had a problem where jump inputs were being ignored. At first I thought the player was bouncing when they hit the ground and that bounce increased their distance enough from the ground that the raycast wouldn't reach the ground. Later I realised that the error had to be due to it being in the FixedUpdate method instead of the Update method because sometimes when I press jump while grounded it would also ignore the jump too (so it wasn't to do with bouncing, it was to do with not being in Update).

I spent a lot of time making the jump mechanic feel good since it is the core component of avoiding traps. I added Coyote time. Issue: Infinite jumps possible during coyote time. Added a boolean hasJumped and this is checked before jumping. For jumping I used a sphere cast to check if the player was close to the ground. Sometimes while falling, the player could press space right before touching the ground. This would "jump", adding force to the player, but since the player was falling down, that force wouldn't jump them as high. Solution: Set velocity to 0 before jump to ensure all jumps are equal. I also added the ability to prejump. As long the player pressed jump within a certain amount of time before touching the ground, the game will automatically jump for the player when the Player touches the ground. This eliminates the moments of the jump input being ignored and they player saying: "but I pressed jump" when actually the player pressed jump a few milliseconds too early.

Added a model and animation. Animation seemed to be easy but it isn't. Animations are an optional thing. Considering not doing it. I did it anyway because movement is a key part of the game and the animations and rotation of the player provided useful feedback while

playing the game. I made animation transitions and conditions. There are 2 parameters in the Animation controller: isJumping and Speed. The model has a running and idle animation. It switches between the running and idle animation based on the current speed of the player capsule. The jumping animation starts when the player jumps and ends when the player touches the ground. To do this: the isJumping parameter is changed after the sphere cast that checks if the player is grounded or not in the Player script. Effort went into making the animations transition seamlessly by editing the animation start times and some settings such as Loop Time. In the Player script, the model rotates to face the direction that the capsule is moving in.

The dragon traps were made using a trigger that moves from the dragon and across the player's path. The trigger moves along with the fire of a particle emitter at the dragon's mouth. This was set up by adjusting the speed that the trigger moves until it looks to be the same speed as the fire. The dragon traps use coroutines to start them all at 0.5 second intervals. Each dragon trap uses its own coroutine to cycle it's breathing fire and not breathing fire periods. Spacing each dragon's coroutine 0.5 seconds apart ensures that the execution of all dragon trap's firing is spread out over multiple frames and it also looks cool. The trap's particle emitter is started every fire cycle. The trap's trigger's position is reset and it is moved using Translate. This is so that the trap's hitbox moves with the flames.

Finding suitable checkpoints (ie. not above a hole in the ground). Every 20 units: a while loop with a raycast down checks if the ground is there and if it isn't, then backstep a few units and raycast that location. If the raycast reaches the ground, then it is a suitable checkpoint. This builds up a list of checkpoints. A checkpoints is a float that measures the distance from the start of the race that the checkpoint is located. I had a lot of trouble here because I thought I was raycasting the correct distance but my origin was too high from the ground so I never got any successful raycasts and it took a while to see that the ground was inside a parent object that was displaced downwards so my world coordinate origin for raycast was too far away. I made a method to teleport to last known checkpoint. When the player collides with a trap trigger, they are teleported back to last checkpoint. The ground below the platform is a trigger with the "Trap" layer set. The trigger that moves along with the dragon's fire is a trigger with the "Trap" layer set.

Used CameraShake from Sam Redfern's MazeShooter.

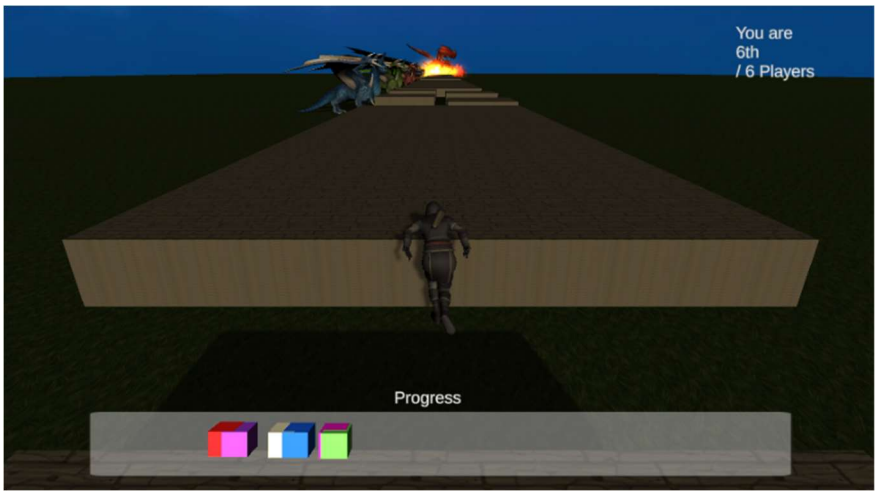
I created the system to calculate the position of players and AI. I created an array of structs for player/AI information. Information about percent of race completed is used to draw a 3D cube representing each player/AI. This is a float called currentRaceProgress that tracks the race progress for a player/AI as a percentage. For the player, every frame, this value is updated to match the player's current progress. I made currentRaceProgress update for AI. AI's have a fixed speed of progression through the race (randomly chosen speed for each AI at the start of the level). AI's have a chance (randomly chosen chance for each AI at the start of the level) to "hit a trap" and therefore lose progress.

I made a method in GameManager that takes in a position as an integer and updates the text component on the GUI with some text that says "You are 1st / 6 Players". A boolean is used to change the text to instead say "You finished…" as opposed to "You are…" so that I can re-use this code for the post game screen. A text mesh pro's text component is updated. A string builder is used to reduce the overhead of string concatenation in unity because this is running in the update method so it should be efficient.

The furthest trap shoots fire first. This helps the player see the traps activating towards them giving them time choose to slow down or speed up past a trap.

I put easy jumps at the start of the race so that the player can practise/learn.

Happy accident: The player can cling onto the edges of platforms if they barely miss a jump and keep holding the movement key to move towards the platform surface. Player can press space to jump. It feels like latching onto the edge and climbing up it.



For some reason when I build the project, the AI moves faster. I am not sure what is going on, I solved it by lowering the AI's speed.

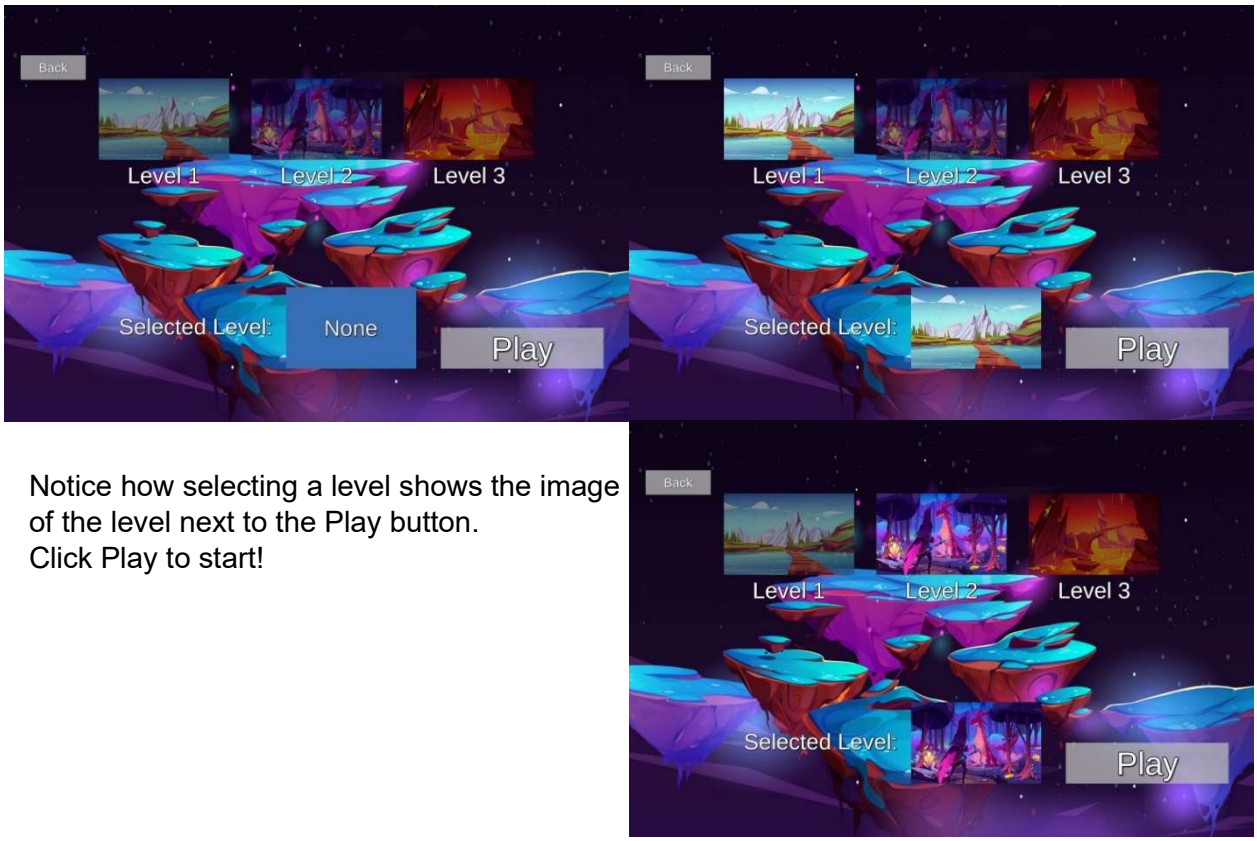
Added Game sounds and music. The GameManager controls the sounds. In each level, the Shakeable camera has an audio source component and a child with another audio source component.

I encountered an issue: The game manager controls the audio sources, e.g. sets the audio clip and starts them. My design has the main menu in one scene and the actual levels in separate scenes. When you start the game and are in the main menu scene, the player had not been created yet, so the audio source that follows the player on the camera had not been created yet. My solution was to create another set of audio sources that would instantiate when the level is loaded and RaceManager (which exists in the level's scene) will pass the audio sources to the GameManager and the GameManager will overwrite the audio sources it used for the main menu. So it works. A side note is that, when Play Again is clicked, you do not load the main menu scene again, the game state just changes and the main menu GUI is shown again. This means that you still have the Audio sources of the level. That is why the overwritten audio source doesn't need to be retrieved again. During development I forgot to send the player back to the main menu scene, and I didn't notice until doing the sounds at the end. Ideally the game should go back to the main menu scene but this works for this version of the game.

The project guidelines said 3 pages of writing about the development maximum, so here is some more less relevant information about the development.

I kept a log text file. After each session I would write about what I did and any problems I encountered and my solution. The purpose of the log file was because to help me remember what happened for this write up.

Level Select GUI



Notice how selecting a level shows the image of the level next to the Play button.
Click Play to start!

I added a fake loading screen to experiment with putting instructions in the loading screen as a time saver for the player. However, games load quicker on different systems so it might not give the player enough time to process the instructions. Also, why put them somewhere they can't go back to? So that is why I added instructions at the start of the level. I left the fake loading screen in the game for demonstration purposes. I would disable it if I was to develop the game further.

I wanted to allow the player an opportunity to familiarize themselves with the instructions and controls. That is why when you load into a level you are a few metres behind the start line. In this area the game is in the "in-game" state. In this state the race has not started and the player has time and some space to read the instructions and controls and move around. They can cross the starting line in their own time to start the race.

During the creation of the system to calculate the position of players and AI: Initially I wanted the RaceManager to calculate and update the race percentages of the player and also manage the progress of the AI. However, I had an issue (seen below) accessing the structs that contain the variables, My options were: 1. make duplicate variables to make calculations in the RaceManager, 2. figure out how to access the information in the player structs of the GameManager, 3. Use getters and setters or 4. Do the calculations in the GameManager. I chose option 4. For some reason this code in the update of the RaceManager wouldn't work:

```
// Update the distance from start line and current race percent for all players
    foreach (GameManager.PlayerStruct p in GameManager.instance.players)
    {
        Debug.Log("p.currentRacePercent: " + p.currentRacePercent);
    }
```

Assets\Scripts\RaceManager.cs(86,30): error CS0426: The type name 'PlayerStruct' does not exist in the type 'GameManager'
As a result I made a workaround that led to storing startLinePoint and raceDistance in GameManager as well as RaceManager. This was option 4.

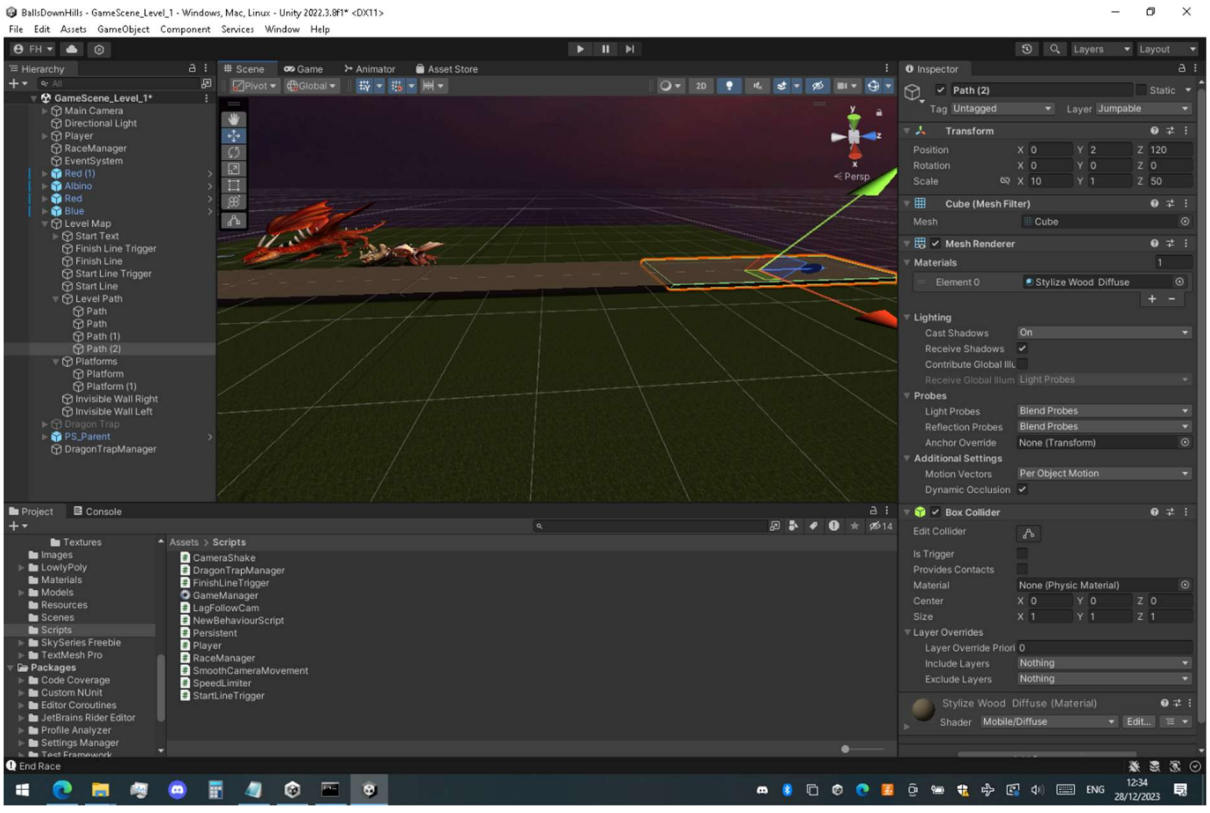
I had problems when clicking “Play Again”. The old cubes stayed on screen. I solved this by destroying the cubes when the "Play Again" button is clicked. The cubes are also destroyed right before they are initialised when starting a new game.

Issue: The player's character looks ghostly white on level 1, I don't know why. Solution: the skybox environment has lighting turned all the way up.

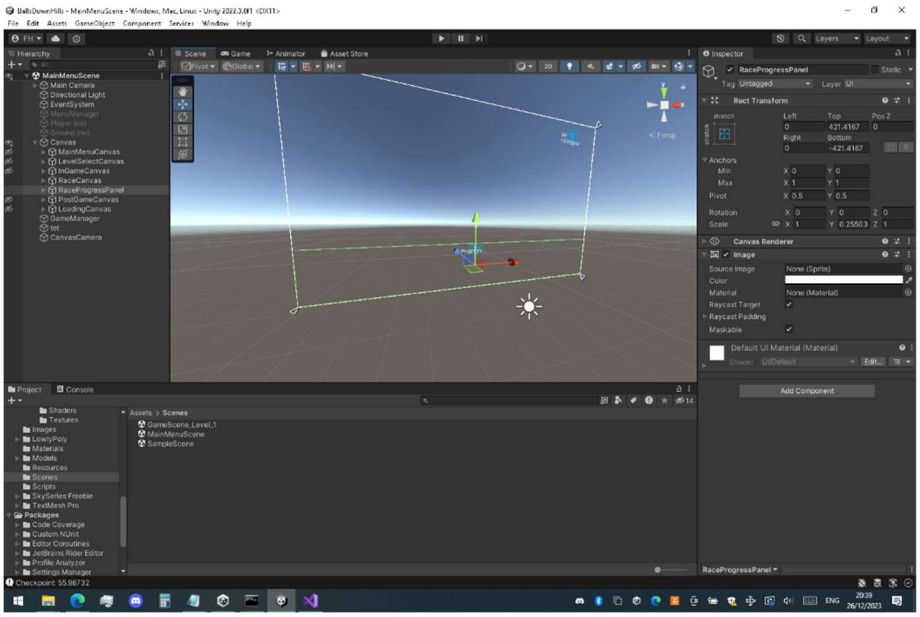
I did a little optimization on the trees on level 2. But fps is still lower on level 2 for me. I then made trees cast and receive no shadows to improve performance since trees are quite far from the player so the player won’t notice. This change is worth the performance boost. I did optimization to the lava material. I removed its height map, that increased fps a lot on level 3.

More screenshots:

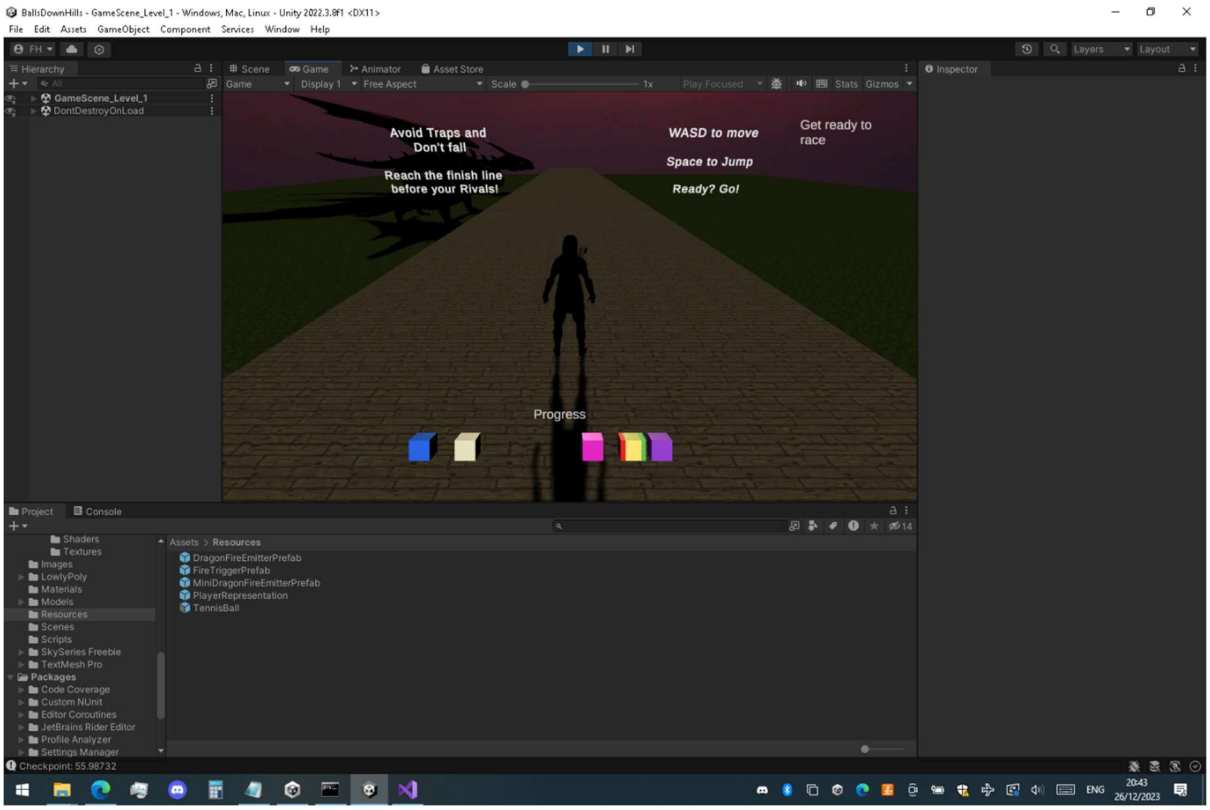
Building the map



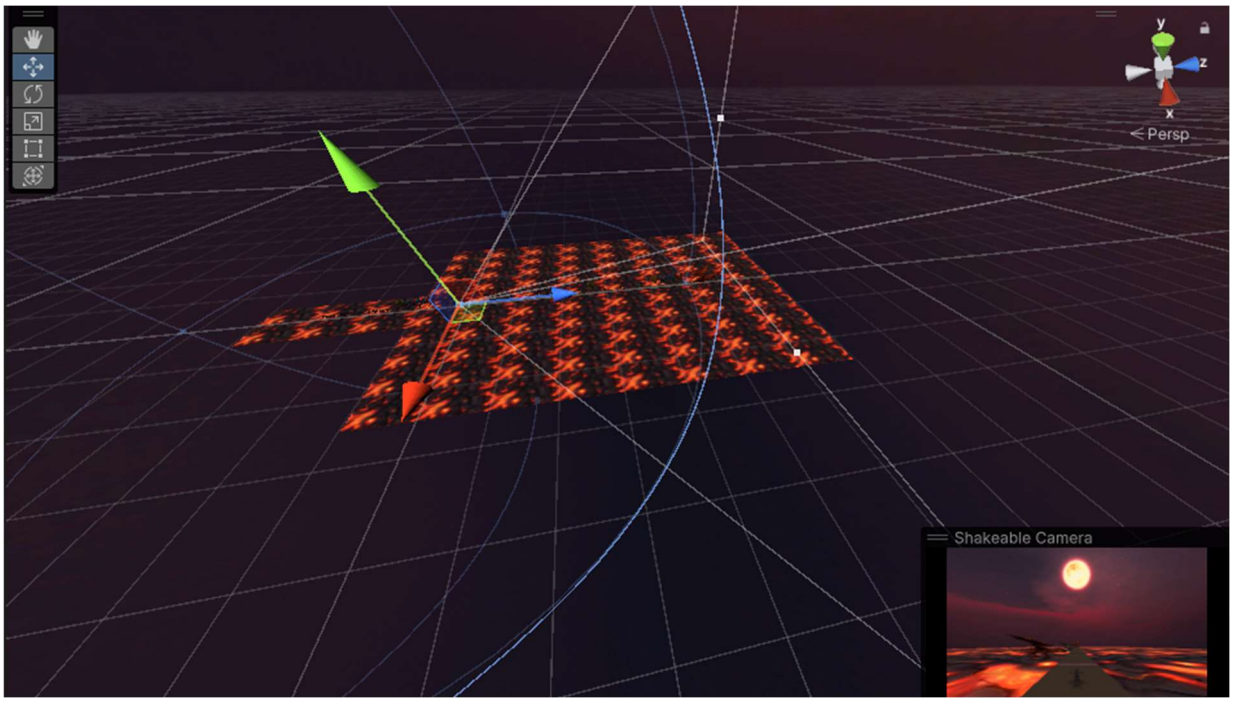
Race Progress Bar that has 3D cubes on a canvas. Screenshot the panel for this.



The moment the Race Progress Bar worked (huge happy moment)



Overview of the level



Scripts

GameManager

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using TMPro;
using System.Text;
using UnityEngine.UI;
using UnityEngine.SceneManagement;
using System;

// Enum for game states
public enum GameState {
    none,
    main_menu,
    level_select,
    in_game,
    race,
    post_game
}

// Struct for each player in the game (AI and players included)
[System.Serializable]
public struct PlayerStruct {
    public GameObject player;
    public GameObject playerRepresentationUI; // Cube used in Race Progress
Bar UI
    public float distanceFromStartLine;

    // The % of the race that has been completed
    // i.e. player is 90% complete the race (based on distance from start to
finish line)
    public float currentRacePercent;
    public bool isAI;
    public float chanceToTouchTrap;
    public float speed;
    public int position; // 1st, 2nd, 3rd, etc...
    public bool canMove; // (Used for AI) Only true when racing. i.e. Not
before race, not once this player reaches the end of the race
    public int playerID;
}

public class GameManager : MonoBehaviour {

    // GUI Components
```

```
    public GameObject mainMenuCanvas, levelSelectCanvas, inGameCanvas,
raceCanvas, raceProgressPanel, postGameCanvas, loadingCanvas;
    public GameObject txtPosition, txtPositionPostGame; // TextMeshProUGUI
    public Button levelSelectPlayButton;
    public int levelSelected; // Level Selected int
    public Button level1;
    public Button level2;
    public Button level3;
    public GameObject selectedLevel;

    // Menu navigation is set in the inspector via onclick methods that use
functions in this class
    // Onclick methods call menu navigation methods seen below

    public static GameManager instance; // Singleton pattern

    public static int TrapLayer;

    public static Vector3 levelStartPoint = new Vector3(5f, 2f, 5f); // Where
the player spawns

    private static WaitForSeconds wait10s = new WaitForSeconds(10f);

    public static GameState gameState = GameState.none;

    // Number of players in the game. This game is singleplayer there is 1
player and 5 AI
    public int numPlayers = 6;

    // Array of materials for the cubes in the Race Progress Bar
    public Material[] playerRepresentationMaterials;

    // Array of cubes for the Race Progress Bar
    public GameObject playerRepresentationPrefab;

    // Value used to position the cubes in the Race Progress Bar
    private float currentRacePercentUIOffset = -12f; // -12 to be off screen
intially

    // Array of PlayerStructs. 1 for each player in the game (AI and players
combined)
    public PlayerStruct[] players;

    // Array used to sort to find position of players e.g. 1st, 2nd, 3rd etc
    public PlayerStruct[] playersSortCurrentRacePercent;

    // centre of start line of the current race
    public Vector3 startLinePoint;
```

```
    // Player one's position in the gameworld
    private Vector3 playerOnePosition;

    // race distance of the current race
    public float raceDistance;

    // Sounds
    // Audio Sources
    public AudioSource audioSource;
    public AudioSource musicAudioSource;

    // Audio Clips
    public AudioClip hitByTrapClip;
    public AudioClip startEndRaceClip;
    public AudioClip menuMusic;
    public AudioClip inGameMusic;

    // Only have this reference to disable/enable the audio listener on the
camera
    public GameObject canvasCamera;


    // Awake is called before the Start method and is guaranteed to be
executed before Start in other scripts
    void Awake()
    {
        instance = this;
    }

    void Start () {
        // Set the loading canvas inactive
        loadingCanvas.gameObject.SetActive(false);

        // Get the TrapLayer
        TrapLayer = LayerMask.NameToLayer("Trap");

        // Set the level selected to 0 (not valid initally)
        levelSelected = 0;

        GoToMainMenu();

        // Add listener to Play button in Level Select screen that calls a
method to start the game when clicked
        levelSelectPlayButton.onClick.AddListener(PlayNewGame);

        // Creating an array of PlayerStructs. 1 for each player in the game
(AI and players combined)
        players = new PlayerStruct[numPlayers];
```

```
    }

    void Update()
    {
        // If the raceProgressPanel is active i.e. We are either: in-game,
race or post-game
        if (raceProgressPanel.activeSelf)
        {
            // Two functions are done in this if statement
            // Function 1. Position Text UI (i.e. 1st, 2nd or 3rd etc)
            // Function 2. RaceProgress UI. (The cubes at the bottom that show
player's progress)
            // Function 1 and 2 are together for efficiency. We only need one
for loop in Update
            // that runs for numPlayers iterations.
            // Function 1 iterates players[] but Function 2 iterates
playersSortCurrentRacePercent[].

            // Function 1. Position Text UI (i.e. 1st, 2nd or 3rd etc)
            // Get the position of each player
            // Sort a copy of the PlayerStruct array on the attribute:
currentRacePercent in descending order.
            playersSortCurrentRacePercent = players;
            Array.Sort(playersSortCurrentRacePercent, (x, y) =>
y.currentRacePercent.CompareTo(x.currentRacePercent));


            // Function 2. RaceProgress UI. (The cubes at the bottom that show
player's progress)
            // Draw player representations (cubes) in RaceProgress UI area
            // Get a reference point in 3D space to position cubes on canvas
            Vector3 refPoint =
raceProgressPanel.GetComponent<RectTransform>().position;

            // Update all player representation (cube) positions
            for (int i = 0; i < numPlayers; i++)
            {
                // Update non-AI player's currentRacePercent
                if (!players[i].isAI)
                {
                    // Get the player's distance from the start line
                    players[i].distanceFromStartLine =
players[i].player.transform.position.z - startLinePoint.z;
                    // Turn that into a percent
                    players[i].currentRacePercent =
(players[i].distanceFromStartLine / raceDistance) * 100f;

                }
```

```
                // Update AI player's currentRacePercent
                if (players[i].isAI && players[i].canMove)
                {
                    // Update AI's race percent according to speed and race
distance
                    // Race distance is included so that the rate of course
completion is the same for any length of map
                    players[i].currentRacePercent += players[i].speed *
Time.deltaTime * (raceDistance/3000);

                    // When the AI reaches the end of the race, they stop
moving
                    if (players[i].currentRacePercent >= 100)
                    {
                        players[i].canMove = false;
                    }

                    // Apply race percent penalty when the AI hits a trap
(random chance)
                    if (UnityEngine.Random.value <
players[i].chanceToTouchTrap)
                    {
                        // Race distance is included so that the rate of
course completion is the same for any length of map
                        players[i].currentRacePercent -= 20f * (raceDistance /
3000);
                        players[i].currentRacePercent =
Mathf.Clamp(players[i].currentRacePercent, 0f, 100f);
                    }
                }

                // Convert race percent into a value from -6 to 6 for position
on the Race Progress Bar UI
                currentRacePercentUIOffset =
Map(players[i].currentRacePercent, 0f, 100f, -6f, 6f);
                currentRacePercentUIOffset =
Mathf.Clamp(currentRacePercentUIOffset, -6f, 6f);

                // Note on the vector space:
                // from refPoint.x-6 -> refPoint.x+6 is 0->100 % race
completion.
                // refPoint.y more negative values are closer to the camera.

                // Set the position of the cube
                players[i].playerRepresentationUI.transform.position = new
Vector3(refPoint.x + currentRacePercentUIOffset, refPoint.y + 0.5f, refPoint.z
- 1f + (((float)i)/50f));
```

```
                // The "i" in this line of code offsets the cubes a little so
they aren't all fully on top of each other


                // Function 1. TEXT UI (i.e. 1st, 2nd or 3rd etc)
                // Get the non-AI player position
                // The only non-AI player is always playerID = 0
                if (playersSortCurrentRacePercent[i].playerID == 0)
                {
                    // Update the Player position on the UI (i.e. 1st, 2nd or
3rd etc)
                    SetPosition((i + 1), false);
                }
            }
        }
    }

    void initialisePlayerStructArray()
    {
        // Destroy game objects if there are any already there
        if (players.Length != 0)
        {
            DestroyPlayerRespresentationObjects();
        }

        for (int i = 0; i < numPlayers; i++)
        {
            players[i].player = null;  // AI don't have one. Player assigns
it's own later
            GameObject playerRepresentation =
Instantiate(playerRepresentationPrefab); // Instantiate cube for Race Progress
Bar
            playerRepresentation.transform.GetChild(0).GetComponent<Renderer>(
).sharedMaterial = playerRepresentationMaterials[i]; // Material for Cube
            DontDestroyOnLoad(playerRepresentation);
            players[i].playerRepresentationUI = playerRepresentation;  // The
Cube in the UI that represents the player/AI
            players[i].distanceFromStartLine = 0f;
            players[i].currentRacePercent = 0f;
            players[i].isAI = true;
            players[i].chanceToTouchTrap = UnityEngine.Random.Range(0.005f,
0.0065f);
            players[i].speed = UnityEngine.Random.Range(12f, 16f);
            players[i].position = numPlayers; // Everyone is last to begin
with
            players[i].canMove = false;
            players[i].playerID = i; // This is necessary because a copy of
this array will be sorted to get positions 1st, 2nd etc
        }
```

```
    }

    // Destroy any player representation objects from the last game
    public void DestroyPlayerRespresentationObjects()
    {
        for (int i = 0; i < numPlayers; i++)
        {
            Destroy(players[i].playerRepresentationUI); // Destroy the Cube in
the UI that represents the player/AI
        }
    }

    // Method that sets all player's canMove (only the AI utilize the canMove
variable)
    public void SetAICanMove(bool cM)
    {
        for (int i = 0; i < numPlayers; i++)
        {
            players[i].canMove = cM;
        }
    }

    // Function to map a value from one range to another
    float Map(float value, float fromMin, float fromMax, float toMin, float
toMax)
    {
        return (value - fromMin) / (fromMax - fromMin) * (toMax - toMin) +
toMin;
    }

    // Function to set game state
    public static void SetGameState(GameState state) {
        gameState = state;

        if (state == GameState.main_menu) {
            instance.mainMenuCanvas.SetActive (true);
            instance.levelSelectCanvas.SetActive(false);
            instance.inGameCanvas.SetActive (false);
            instance.raceCanvas.SetActive(false);
            instance.raceProgressPanel.SetActive(false);
            instance.postGameCanvas.SetActive(false);
            instance.PlayMenuMusic(); // Switch to menu music
        }
        else if (state == GameState.level_select)
        {
            instance.mainMenuCanvas.SetActive(false);
            instance.levelSelectCanvas.SetActive(true);
            instance.inGameCanvas.SetActive(false);
```

```
            instance.raceCanvas.SetActive(false);
            instance.raceProgressPanel.SetActive(false);
            instance.postGameCanvas.SetActive(false);
        }
        else if (state == GameState.in_game) {
            instance.mainMenuCanvas.SetActive(false);
            instance.levelSelectCanvas.SetActive(false);
            instance.inGameCanvas.SetActive(true);
            instance.raceCanvas.SetActive(false);
            instance.raceProgressPanel.SetActive(true); // Can see progress
panel in game
            instance.postGameCanvas.SetActive(false);
            instance.canvasCamera.GetComponent<AudioListener>().enabled =
false; // Disable canvas audio listener while in game
            instance.musicAudioSource.Stop();
        }
        else if (state == GameState.race)
        {
            instance.mainMenuCanvas.SetActive(false);
            instance.levelSelectCanvas.SetActive(false);
            instance.inGameCanvas.SetActive(false);
            instance.raceCanvas.SetActive(true);
            instance.raceProgressPanel.SetActive(true); // Can see progress
panel during race
            instance.postGameCanvas.SetActive(false);
        }
        else if (state == GameState.post_game) {
            instance.mainMenuCanvas.SetActive(false);
            instance.levelSelectCanvas.SetActive(false);
            instance.inGameCanvas.SetActive(false);
            instance.raceCanvas.SetActive(false);
            instance.raceProgressPanel.SetActive(true); // Can see progress
panel post game
            instance.postGameCanvas.SetActive(true);
        }
    }

    // Set the position text on GUI
    // @param position integer
    // @param is this for post game or not? bool
    // generates and sets text to the post game Text or the race Text
    public static void SetPosition(int pos, bool isPostGame) {
        // Select suffix
        string suffix = "default";
        if (pos == 1)
        {
            suffix = "st";
        }
```

```
        else if (pos == 2)
        {
            suffix = "nd";
        }
        else if (pos == 3)
        {
            suffix = "rd";
        }
        else if (pos > 3)
        {
            suffix = "th";
        }

        // Optimise String concatenation using a string builder
        string pos_ui_text = "You are\n";
        string pos_ui_text_postGame = "You finished\n";
        StringBuilder sb = new StringBuilder();

        // Build string
        // Use a different string depending on postGame bool
        if (isPostGame)
        {
            sb.Append(pos_ui_text_postGame);
        }
        else
        {
            sb.Append(pos_ui_text);
        }

        sb.Append(pos);
        sb.Append(suffix);
        sb.Append("\n/ ");
        sb.Append(instance.numPlayers);
        sb.Append(" Players");

        if (isPostGame)
        {
            // Update post game position Text
            instance.txtPositionPostGame.GetComponent<TextMeshProUGUI>().text
= $"{sb.ToString()}";
        }
        else
        {
            // Update race position Text
            instance.txtPosition.GetComponent<TextMeshProUGUI>().text =
$"{sb.ToString()}";
        }
    }
```

```
    // Method that finds the player position and updates the post game UI with
the position (used at end of race)
    public void UpdatePositionUIPostGame()
    {
        for (int i = 0; i < numPlayers; i++)
        {
            // Get the player position
            // The only player is always playerID = 0
            if (playersSortCurrentRacePercent[i].playerID == 0)
            {
                // Update the Player position on the UI (i.e. 1st, 2nd or 3rd
etc)
                SetPosition((i + 1), true);
                return;
            }
        }
    }

    // Menu navigation methods

    // Start the game with the currently selected level
    public void PlayNewGame()
    {
        // Reset the player struct array
        initialisePlayerStructArray();

        // Currently only 3 playable levels
        if (levelSelected > 0 && levelSelected < 4)
        {
            // Show fake loading screen for 10 seconds
            StartCoroutine(ShowFakeLoadingScreenAndLoadScene());
        }
    }

    private IEnumerator ShowFakeLoadingScreenAndLoadScene()
    {
        // Show fake loading screen
        loadingCanvas.gameObject.SetActive(true);
        loadingCanvas.GetComponent<Canvas>().sortingOrder = 1;

        // Wait for the loading screen duration
        yield return wait10s;

        // Set loading screen inactive
        loadingCanvas.gameObject.SetActive(false);

        // Set Game state to in game after loading screen
```

```
        SetGameState(GameState.in_game);

        // Load the selected level's scene
        string sceneToLoad = "GameScene_Level_" + levelSelected;
        SceneManager.LoadScene(sceneToLoad); //
SceneManager.LoadScene("GameScene_Level_1");
    }

    // Close the game
    public void QuitGame()
    {
        Application.Quit();
    }

    // Switch to main menu
    public void GoToMainMenu()
    {
        SetGameState(GameState.main_menu);
    }

    // Switch to level select
    public void GoToLevelSelect()
    {
        SetGameState(GameState.level_select);
    }

    // Set game state to race
    public void GoToRace()
    {
        SetGameState(GameState.race);
    }

    // Set game state to post game
    public void GoToPostGame()
    {
        SetGameState(GameState.post_game);
    }


    // Select level stores the choice made as an integer and updates the level
selected image's sources in Level Select GUI
    public void SelectLevel(int level)
    {
        levelSelected = level;

        switch(levelSelected)
        {
            case 1:
```

```
                selectedLevel.GetComponent<Image>().sprite =
level1.GetComponent<Image>().sprite;
                break;
            case 2:
                selectedLevel.GetComponent<Image>().sprite =
level2.GetComponent<Image>().sprite;
                break;
            case 3:
                selectedLevel.GetComponent<Image>().sprite =
level3.GetComponent<Image>().sprite;
                break;
            default:
                break;
        }
    }

    // Assign in game Audio Source
    // This overwrites the audio sources of the main menu scene
    // In this version of the game the player never returns to the main menu
scene after clicking play the first time
    // So the audio sources of the level can be used when the main menu's GUI
appears on screen after Play Again is clicked
    public void SetInGameAudioSources(AudioSource a, AudioSource aMusic)
    {
        audioSource = a;
        musicAudioSource = aMusic;

        PlayInGameMusic(); // Switch to in game music
    }

    // Play the audio clip of the player being hit by a trap
    public void PlayHitByTrapClip()
    {
        // Assign the audio clip to the AudioSource component
        audioSource.clip = hitByTrapClip;

        // Play the sound
        PlaySound(audioSource);
    }

    // Play the audio clip for the start of a race
    public void PlayStartEndRaceClip()
    {
        // Assign the audio clip to the AudioSource component
        audioSource.clip = startEndRaceClip;

        // Play the sound
        PlaySound(audioSource);
```

```
    }

    // Play the audio clip for menu music
    public void PlayMenuMusic()
    {
        // Assign the audio clip to the AudioSource component
        musicAudioSource.clip = menuMusic;

        // Play the sound
        PlaySound(musicAudioSource);
    }

    // Play the audio clip for in game music
    public void PlayInGameMusic()
    {
        // Assign the audio clip to the AudioSource component
        musicAudioSource.clip = inGameMusic;

        // Play the sound
        PlaySound(musicAudioSource);
    }

    // Method to play a sound as long as there is an audio clip in the audio
source
    void PlaySound(AudioSource aS)
    {
        // Check if an audio clip is assigned
        if (aS.clip != null)
        {
            // Play the audio clip
            aS.Play();
        }
        else
        {
            // Show an error
            Debug.LogError("No audio clip assigned. Please assign an AudioClip
in the Unity Editor.");
        }
    }
}
```

SmoothCameraMovement

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

public class SmoothCameraMovement : MonoBehaviour
{
    // A reference to the player assigned in inspector
    public GameObject player;

    // Offset between camera and player set to (0, 2, 5) in inspector
    public Vector3 offset;

    // Time it will take to reach the target
    // The lower the number the quicker the camera moves to the player
    public float smoothTime = 0.3f;

    private Vector3 velocity = Vector3.zero;

    // A reference to the main camera
    private GameObject mainCamera;

    void Start()
    {
        // Get main camera reference
        mainCamera = GameObject.Find("Main Camera");
    }


    void FixedUpdate()
    {
        // Get the terget position that we want the camera to move to. i.e.
player position + offset
        Vector3 cameraTargetPos = player.transform.position + offset;

        // Smoothly move the Main Camera towards the target position over time
        mainCamera.transform.position =
Vector3.SmoothDamp(mainCamera.transform.position, cameraTargetPos, ref
velocity, smoothTime);

        // Update camera's rotation to face them player
        transform.LookAt(player.transform);
    }
}
```

Player

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


// Attached to player GameObject (a capsule)
public class Player : MonoBehaviour {
    private Rigidbody rigid;

    // The model is the child of the Player GameObject. This child contains
the player's model
    public GameObject modelParent;
    private Transform modelTransform;

    public LayerMask jumpableLayer;

    private float sphereRadius = 0.5f;
    private float distanceFromPlayerCentreToFeet = 1f;
    public float jumpForce = 10000f;
    public float movementForce = 100000f;
    public float customGravity = -19.62f;

    private int raycastHitCount = 0;
    private RaycastHit[] hits = new RaycastHit[1];
    private bool isGrounded = false;

    private float coyoteTime = 0.1f; // Duration of coyote time
    private float coyoteTimer = 0f;
    private bool hasJumped = true;
    // Pre-jump buffer time
    // A prejump is when you press the jump button before touching the ground
    // As long the player pressed jump within a certain amount of time before
touching the ground,
    // the game will automatically jump for the player when the Player touches
the ground
    private float preJumpBuffer = 0.15f;
    private float preJumpTimer = 0f;
    // Prejump is not allowed immediately after jumping (prevent automatic
jump the frame after you jump)
    // Without this boolean you get a double jump every time you jump
    private bool preJumpAllowed = false;

    private Animator playerAnimator;

    public GameObject raceManager;

    public GameObject player;
```

```
    private GameObject mainCamera;

    private Camera childCamera;

    void Start () {
        // Get the Player capsule's rigidbody
        rigid = GetComponent<Rigidbody>();

        // Get the model's transform
        modelTransform = modelParent.transform;

        // Get the animator
        playerAnimator = modelTransform.GetComponent<Animator>();

        if (modelTransform == null)
        {
            Debug.LogError("Model transform not found!");
        }

        // Set custom gravity
        Physics.gravity = new Vector3(0, customGravity, 0);

        raceManager = GameObject.Find("RaceManager");

        player = GameObject.Find("Player");

        mainCamera = GameObject.Find("Main Camera");

        childCamera = mainCamera.GetComponentInChildren<Camera>();

        // Updating players struct in GameManager
        // Assign player gameobject
        GameManager.instance.players[0].player = player;
        // Set isAI to false
        GameManager.instance.players[0].isAI = false;
    }

    void Update () {
        // Check if the player is grounded
        // Use NonAlloc version for efficiency. Cuts down on new and delete
operations
        raycastHitCount = Physics.SphereCastNonAlloc(this.transform.position,
sphereRadius, Vector3.down, hits, distanceFromPlayerCentreToFeet -
sphereRadius, jumpableLayer);

        // Update isGrounded boolean based on result of raycast
        isGrounded = raycastHitCount > 0;
```

```
        // Update isJumping parameter for player's jumping animations
        playerAnimator.SetBool("IsJumping", !isGrounded);

        // Update coyote timer
        if (isGrounded)
        {
            // Movement force on ground
            movementForce = 100000f;

            // Pre-jump automatic jump
            if ((preJumpTimer > 0f) && preJumpAllowed)
            {
                // Set velocity to 0 before jump to ensure all jumps are equal
                rigid.velocity = new Vector3(rigid.velocity.x, 0f,
rigid.velocity.z);
                // Jump
                rigid.AddForce(Vector3.up * jumpForce);
                // Cannot prejump until jump button is pressed again (prevent
double jump)
                preJumpAllowed = false;

                // Set the jump flag when jumping
                hasJumped = true;
                preJumpTimer = 0f;
            }

            coyoteTimer = coyoteTime;

            // Reset the jump flag when grounded
            hasJumped = false;
        }
        else
        {
            // Movement force in the air
            movementForce = 10000f;

            // Decrement timers while not grounded for jump mechanics
            coyoteTimer -= Time.deltaTime;
            preJumpTimer -= Time.deltaTime;
        }

        // Check for spacebar key or controller button (A on Xbox controller)
during coyote time
        if (Input.GetKeyDown(KeyCode.Space) || Input.GetButtonDown("Submit"))
        {
            // Start the pre-jump buffer timer
            preJumpTimer = preJumpBuffer;
```

```
            preJumpAllowed = true;

            if ((coyoteTimer > 0f) && !hasJumped)
            {
                // Set velocity to 0 before jump to ensure all jumps are equal
                rigid.velocity = new Vector3(rigid.velocity.x, 0f,
rigid.velocity.z);
                // Jump
                rigid.AddForce(Vector3.up * jumpForce);
                // Cannot prejump until jump button is pressed again (prevent
double jump)
                preJumpAllowed = false;
                // Set the jump flag when jumping
                hasJumped = true;
            }
        }
    }

    void FixedUpdate() {
        // Move the player with respect to the camera
        float h = Input.GetAxis("Horizontal");
        float v = Input.GetAxis("Vertical");

        Vector3 camRight = Camera.main.transform.right;
        camRight.y = 0f;
        camRight = camRight.normalized;
        Vector3 camForward = Camera.main.transform.forward;
        camForward.y = 0f;
        camForward = camForward.normalized;

        // Move Player ignoring contorller deadzones
        if (Mathf.Abs(h) > 0.05f || Mathf.Abs(v) > 0.05f)
        {
            Vector3 moveDirection = h * camRight + v * camForward;
            rigid.AddForce(moveDirection * Time.fixedDeltaTime *
movementForce);

            // Rotate the model to face the movement direction
            if (modelTransform != null)
            {
                modelTransform.forward = moveDirection.normalized;
            }

            // Update animator parameter for running
            playerAnimator.SetFloat("Speed", 1f);
        }
        else
        {
```

```
            // Update animator parameter for idle
            playerAnimator.SetFloat("Speed", 0f);
        }
    }

    void OnTriggerEnter(Collider collider) {
        // Hit by trap
        if (collider.gameObject.layer == GameManager.TrapLayer) {
            // Set the player's velocity to 0
            rigid.velocity = Vector3.zero;

            // Play a sound
            GameManager.instance.PlayHitByTrapClip();

            // Get last checkpoint
            float lastCheckpointZ =
raceManager.GetComponent<RaceManager>().LastCheckpoint(player.transform.positi
on.z);
            // Teleport player to last checkpoint
            player.transform.position = new Vector3(0f, 1.5f,
lastCheckpointZ);
            // Bring the camera too
            childCamera.transform.position = new
Vector3(childCamera.transform.position.x, childCamera.transform.position.y,
lastCheckpointZ - 3);
            // Shake the camera
            CameraShake.Shake(0.06f, 0.1f);
        }
    }
}
```

DragonTrapManager

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

public class DragonTrapManager : MonoBehaviour {
    // Array of the dragon trap objects
    private GameObject[] dragonTraps;

    private static WaitForSeconds wait500ms = new WaitForSeconds(0.5f);
    private static WaitForSeconds wait4s = new WaitForSeconds(4f);
    private static WaitForSeconds wait1s = new WaitForSeconds(1f);

    void Start()
    {
        // Get all of the dragon traps and put them in an array
        dragonTraps = GameObject.FindGameObjectsWithTag("Dragon Trap");
        // Sort the array by their z co-ordinate
        Array.Sort(dragonTraps, (x, y) =>
y.transform.position.z.CompareTo(x.transform.position.z));
    }

    // Method that calls a coroutine that starts all of the coroutines of
dragon firing cycles
    public void ActivateDragonTraps()
    {
        StartCoroutine(ActivateAllDragonTraps());
    }

    // Coroutine that starts all of the coroutines of dragon firing cycles
    IEnumerator ActivateAllDragonTraps()
    {
        // For every dragon trap
        foreach (var dragon in dragonTraps)
        {
            // Every 0.5 seconds...
            yield return wait500ms;

            // Start the firing cycle of a dragon trap
            StartCoroutine(DragonTrapCoroutine(dragon));
        }
    }

    // Coroutines of a dragon's firing cycle
    IEnumerator DragonTrapCoroutine(GameObject dragon)
    {
        // Get the position of the Dragon Trap
```

```
        Vector3 dPos = dragon.transform.position;

        // Spawn a box trigger with the tag "Trap"
        GameObject trigger = Instantiate(Resources.Load("FireTriggerPrefab",
typeof(GameObject))) as GameObject;

        // Set the trigger inactive
        trigger.SetActive(false);

        float startTime = 0f;

        Vector3 pos = new Vector3(-5.4f, 2, dPos.z);
        Vector3 triggerPos = new Vector3(-25f, 2, dPos.z);

        // Spawn a particle emitter
        GameObject particleEmitter =
Instantiate(Resources.Load("DragonFireEmitterPrefab", typeof(GameObject))) as
GameObject;
        particleEmitter.transform.position = pos;

        // Spawn a smaller particle emitter
        GameObject miniParticleEmitter =
Instantiate(Resources.Load("MiniDragonFireEmitterPrefab", typeof(GameObject)))
as GameObject;
        miniParticleEmitter.transform.position = pos;

        // Start the fireblast
        while (true)
        {
            // Warn the player the fire is coming with the smaller particle
emitter
            miniParticleEmitter.GetComponent<ParticleSystem>().Play();

            // Delay before the actual fire blast
            yield return wait1s;

            // Set the trigger position
            trigger.transform.position = triggerPos;

            // Start particle emitter and set trigger active
            particleEmitter.GetComponent<ParticleSystem>().Play();
            trigger.SetActive(true);

            startTime = Time.time;

            // Duration of fireblast
            while (Time.time - startTime < 2f)
            {
```

```
                // Move the trigger around the speed of the fire particles
until it has surpassed the width of the track
                if(trigger.transform.position.x < -10f)
                {
                    // Move the trigger
                    trigger.transform.Translate(Vector3.down * 15f *
Time.deltaTime);
                }

                yield return null;
            }

            // Stop the smaller particle emitter
            miniParticleEmitter.GetComponent<ParticleSystem>().Stop(true,
ParticleSystemStopBehavior.StopEmittingAndClear);

            // Set trigger inactive and stop particle emitter
            trigger.SetActive(false);
            particleEmitter.GetComponent<ParticleSystem>().Stop(true,
ParticleSystemStopBehavior.StopEmittingAndClear);

            // Delay before the next fire blast
            yield return wait4s;
        }
    }
}
```

RaceManager

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

// This class is to manage the race. Starts when a level is loaded
// Note this class enforces a constraint that all levels need to have the same
width path (but any length)
// and be built along the positive z direction.
public class RaceManager : MonoBehaviour {
    private float raceTimer;
    private GameObject startLineTrigger, finishLineTrigger;
    public GameObject dragonTrapManager;

    private List<float> checkpoints = new List<float>();

    // Total length of the level
    private float raceDistance;

    private int numCheckpoints;

    private Vector3 startLinePoint;

    private float distanceBetweenCheckpoints = 20f;

    // Used for finding suitable checkpoints
    private float backstep = 0f;

    // Used for finding the last checkpoint
    private float prevCheckpoint = 0f;

    // Sounds
    public AudioSource audioSource;

    public AudioSource musicAudioSource;

    void Start()
    {
        startLineTrigger = GameObject.Find("Start Line Trigger");
        finishLineTrigger = GameObject.Find("Finish Line Trigger");

        // Distance is start to finish
        raceDistance = Mathf.Abs(finishLineTrigger.transform.position.z -
startLineTrigger.transform.position.z);

        // Checkpoint every 20 units
        numCheckpoints = (int) Mathf.Floor(raceDistance /
distanceBetweenCheckpoints);
```

```
        // Checkpoints are measured from the start line
        startLinePoint = new Vector3(0f, 0f,
startLineTrigger.transform.position.z);

        // Populate list of checkpoints
        RaycastHit hit;
        for (int i = 1; i <= numCheckpoints; i++)
        {
            // Check that a potential checkpoint location has the path below
it (not above a hole)
            while (!Physics.Raycast(new Vector3(0f, 2f,
(startLineTrigger.transform.position.z + i * distanceBetweenCheckpoints -
backstep)), Vector3.down, out hit, 1.6f))
            {
                // Not suitable checkpoint go back some units and try again
                backstep += 5;

                // Breaking from situation where a checkpoint is not possible
                if (backstep > 1000)
                {
                    break;
                }
            }
            // Suitable checkpoint location found, add it to the list of
checkpoints
            checkpoints.Add(startLinePoint.z + i * distanceBetweenCheckpoints
- backstep);
            backstep = 0f;

            // Sounds
            GameManager.instance.SetInGameAudioSources(audioSource,
musicAudioSource);
        }

        // Update startLinePoint in GameManager
        GameManager.instance.startLinePoint = startLinePoint;

        // Update raceDistance in GameManager
        GameManager.instance.raceDistance = raceDistance;
    }

    // Method to start the race
    public void StartRace()
    {
        // Change game state when the race starts
        GameManager.instance.GoToRace();
```

```
        // Check if the start line trigger gameobject was found
        if (startLineTrigger != null)
        {
            // Set the start Line Trigger to inactive
            startLineTrigger.SetActive(false);
        }
        else
        {
            // Show an error
            Debug.LogError("Start Line Trigger not found in the hierarchy.");
        }

        // Play sound
        GameManager.instance.PlayStartEndRaceClip();

        EnableDragonTraps();

        // Make it so that all AI starts moving (in the race progress bar at
the bottom)
        GameManager.instance.SetAICanMove(true);
    }

    // Call the method to activate all dragon traps
    private void EnableDragonTraps()
    {
        dragonTrapManager.GetComponent<DragonTrapManager>().ActivateDragonTrap
s();
    }

    // Method for when the player reaches the end of the race
    public void EndRace()
    {
        // Check if the finish line trigger gameobject was found
        if (finishLineTrigger != null)
        {
            // Set the Finish Line Trigger to inactive
            finishLineTrigger.SetActive(false);
        }
        else
        {
            // Show an error
            Debug.LogError("Finish Line Trigger not found in the hierarchy.");
        }

        // Play a sound
        GameManager.instance.PlayStartEndRaceClip();

        // Update the position i.e. 1st, 2nd etc, for post game text
```

```
        GameManager.instance.UpdatePositionUIPostGame();

        // Change GameState and therefore UI
        GameManager.instance.GoToPostGame();
    }

    // Get the z co-ordinate of the last checkpoint a player passes
    // @param pz the current z co-ordinate of the player that wants their last
checkpoint
    // @return the z co-ordinate of the last checkpoint behind the the player
    public float LastCheckpoint(float pz)
    {
        // If the player respawns before the first checkpoint then they will
go to the start line
        prevCheckpoint = startLinePoint.z;

        // Check that there are checkpoints and the list is not null
        if(checkpoints.Count != 0 || checkpoints != null)
        {
            // Find the biggest checkpoint that is behind the player
            foreach (float checkpoint in checkpoints)
            {
                if (checkpoint > pz)
                {
                    // Return last checkpoint
                    return prevCheckpoint;
                }

                prevCheckpoint = checkpoint;
            }
        }
        // If there are no checkpoints that meet this criteria, return start
line
        return startLinePoint.z;
    }
}
```

CameraShake

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

public class CameraShake : MonoBehaviour {
    private static float shakeDecay = 0f;
    private static float shakeIntensity = 0f;

    void LateUpdate (){
        if (shakeIntensity > 0f){
            transform.localPosition = Random.insideUnitSphere * shakeIntensity
* 10f;
            transform.localRotation = Quaternion.Euler(new Vector3(
                Random.Range (-shakeIntensity,shakeIntensity) * 10f,
                Random.Range (-shakeIntensity,shakeIntensity) * 10f,
                Random.Range (-shakeIntensity,shakeIntensity) * 10f));

            shakeIntensity -= shakeDecay*Time.deltaTime;
        }
    }

    public static void Shake(float intensity, float decay) {
        if (intensity>shakeIntensity)
            shakeIntensity = intensity;
        shakeDecay = decay;
        Debug.Log("Shook");
    }
}
```

Persistent

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

// Makes the object that this is attached to persist when changing scenes
public class Persistent : MonoBehaviour {
    void Awake()
    {
        DontDestroyOnLoad(this.gameObject);
    }
}
```

StartLineTrigger

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

// Script that calls a method in the RaceManager when the Player has reached
the start line
public class StartLineTrigger : MonoBehaviour {
    public GameObject RaceManager;

    private void OnTriggerEnter(Collider other)
    {
        if (other.CompareTag("Player"))
        {
            // Tell the RaceManager that the Player has reached the start line
            RaceManager.GetComponent<RaceManager>().StartRace();
        }
    }
}
```

FinishLineTrigger

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

// Script that calls a method in the RaceManager when the Player has reached
the finish line
public class FinishLineTrigger : MonoBehaviour {
    public GameObject RaceManager;

    private void OnTriggerEnter(Collider other)
    {
        // Check if the entering collider is the player
        if (other.CompareTag("Player"))
        {
            // Tell the RaceManager that the Player has reached the finish
line
            RaceManager.GetComponent<RaceManager>().EndRace();
        }
    }
}
```

SpeedLimiter

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

// Script that limits the maximum speed that the GameObject that this is
attached to can move at
public class SpeedLimiter : MonoBehaviour {
    public float maxSpeed = 1000f;

    void FixedUpdate()
    {
        // Limit this GameObject's maximum velocity to the maxSpeed
        if (this.GetComponent<Rigidbody>().velocity.magnitude > maxSpeed)
        {
            this.GetComponent<Rigidbody>().velocity =
Vector3.ClampMagnitude(this.GetComponent<Rigidbody>().velocity, maxSpeed);
        }
    }
}
```

Credit

Images:

Level 1's Icon
https://www.freepik.com/free-vector/river-with-wooden-bridge-mountain-valley_27368332.htm#query=game%20concept%20art%20wood%20road%20grass&position=20&from_view=search&track=ais
Image by upklyak on Freepik

Level 2's Icon
https://www.freepik.com/free-vector/fairy-tale-scene-with-dragon-medieval-knight-fight-fantasy-landscape-magic-forest-with-mythology-monster-warrior-with-spear-night-vector-cartoon-illustration_63597044.htm#query=game%20concept%20art&position=2&from_view=keyword&track=ais
Image by upklyak on Freepik

Level 3's Icon
https://www.freepik.com/free-vector/lava-hell-background-cave-view-game-illustration_40892116.htm#query=game%20concept%20art&position=46&from_view=keyword&track=ais#position=46&query=game%20concept%20art
Image by upklyak on Freepik

Main menu background
https://www.freepik.com/free-vector/flying-rock-islands-night-sky-cartoon-landscape_31755745.htm#query=game%20concept%20art&position=45&from_view=keyword&track=ais#position=45&query=game%20concept%20art
Image by upklyak on Freepik

Loading screen background
https://www.freepik.com/free-vector/watercolor-adventure-background_16390428.htm#page=2&query=game%20concept%20art&position=0&from_view=keyword&track=ais
Image by freepik on Freepik

Textures:

Grass Texture
https://assetstore.unity.com/packages/2d/textures-materials/floors/hand-painted-grass-texture-78552
By LowlyPoly

Wood Path Texture
https://assetstore.unity.com/packages/2d/textures-materials/wood/stylized-wood-texture-153499
By LowlyPoly

Lava Texture
https://assetstore.unity.com/packages/2d/textures-materials/floors/lava-rocks-texture-1-hand-painted-seamless-tileable-232710
By Texture Me!

Models:

Player Character Model

Akai E Espiritu at https://www.mixamo.com/

Dragon models
https://assetstore.unity.com/packages/3d/characters/creatures/dragon-for-boss-monster-hp-79398
By Dungeon Mason

Animations:

- Standing Jump Running To Run Forward

- Run

- Standing Run Forward

- Warrior Idle

at https://www.mixamo.com/

Skyboxes:

Level 1, 2 and 3's skyboxes
https://assetstore.unity.com/packages/2d/textures-materials/sky/fantasy-skybox-free-18353
By Render Knight

Other:

Sam Redfern's code to move the player with respect to the camera from Player script in MazeShooter

Sam Redfern's CameraShake script from MazeShooter

Fire Particle Emitter
https://www.youtube.com/watch?v=5Mw6NpSEb2o&ab_channel=Sirhaian
By Sirhaian

Tree
https://assetstore.unity.com/packages/3d/environments/fantasy/fantasy-forest-environment-free-demo-35361
By TriForge Assets

All game sounds and music from Canva
https://www.canva.com/
Is copyright free for non-commercial use. No author listed

I hope you enjoyed playing my game!