Burnout Valley

Sajid Jahabar Ali

160212899

ECS657U: Multi-platform Game Development

Contents

1. Environments

My game environment is set in a valley, hence the name. This valley is situated in an island that is dedicated to racing and therefore only has a single race track. The terrain includes mountains around the outskirts and has multiple hills and dips in the land which makes the race more challenging and varied as players will have the ability to go at different speeds depending on if the track is headed uphill or downhill. To add to the atmosphere, I have added trees that surround the track so that the player gets a sense of immersion in the environment and is provided with a nice aesthetic. The trees were downloaded from a free online source [2].

Furthermore, the type of terrain varies across the map. For instance, in the mountainous areas on the outskirts of the map, the terrain is rocky and rough. Around the track, the terrain is covered in grassland and trees which adds to the aesthetic and intended environment for the game setting and correlates with the game title. The track itself has a smooth asphalt texture comparable to a real racetrack which needs to be smooth in order to allow for maximum speed during the race. On the edges of the track, the terrain is once again rocky to ensure that the player stays on the track as going on the rocky terrain would slow the player down making it easier for the opponent to win.

The map is a large and a large section of the map is taken up by the racetrack. The racetrack has multiple sharp bends and turns meaning that the player has to pick their speed wisely as going too fast could result in a crash. This could be catastrophic to the result of the race as accelerating back up to speed from being stationary after a crash takes time, meaning the AI would have gotten a significant lead. Therefore, the bends and turns not only offer a sense of realism, since most race tracks have bends and turns in them, but also grant the player the ability to make key decisions with their speed and test their reaction times when approaching corners.

The racetrack has a racing flag which both serves as the races start point and end point. This shows the player the sole aim of the game which is to reach the finish line (i.e. return to the starting position by doing a full lap of the track) before the opponent in order to win the game. Both cars start at the same point on the line to make the race fair and both cars are on the track at the same time, so the player can see if they need to speed up to catch the AI and overtake him to win the race.

My game is designed to be a short casual game so there is only one level and one map. The player races on this map against the AI and wins if they beat the AI to the finish line or loses if the AI reaches the finish line first.

2. Gameplay features

The main controls for my game are the arrow keys. The arrow keys allow the player to control the car. The car is a prefab from the Unity Standard Assets package [1]. The controls include accelerating with the up-arrow key, turning left with the left-arrow key, turning right with the right-arrow key, and braking with the down-arrow key. The player must use these controls to drive his car around the track in order to win the race maintaining and controlling his speed with the up and down arrow keys. Going too fast would cause the player to crash at a corner and going too slow would cause them to be overtaken by the AI and lose the race.

A secondary control for my game is the ability to control camera angles. This is not essential to playing or winning the game but provides the player with more viewing options for a more comfortable playing experience. This control is bound to the ‘c’ key and allows the player to change between 3 camera angles.

The first camera angle is the default camera angle which provides a medium-distance angle which shows the players car and a relatively small area around it from an angled down position from behind the car. This provides a nice balanced view where the player can focus on his car, the upcoming turn, and the opponent if the opponent is right next to him.

The second camera angle offered is a wide camera angle which, again, shows an angled down view from behind the car but at a greater distance from the car allowing the player to see more of his surroundings. This is useful for the player if the player wants a better idea of the turns, hills or dips coming ahead and grants the player a better idea of the position of his opponent which can help the player decide if they need to speed up or can slow down and drive more cautiously if they have a lead, so they can reduce the risk of crashing.

The third and final camera angle is a driver perspective camera angle which positions the camera on the windscreen of the car. This camera angle is also another standard camera angle offered in many racing games and especially simulation racing games. This camera angle provides a more realistic view of what racers see in a real race. This camera angle may also provide the player with an advantage as it provides a better view of the track meaning the player can see bumps and dips on the track more clearly allowing the user to adjust their speed accordingly.

A mini-map is present in my game which allows the player to see upcoming turns as well as the position of his car and the opponent's car in order to plan ahead and adjust their speed and the route they want to take around the track. The mini-map is displayed in the bottom left of the HUD.

The player can also have the time they took to complete the race, assuming they win, recorded and saved in a text field in the top right of the screen as part of the HUD. This adds another element to the game so that even if the player can consistently beat the AI, they are enticed to continue playing so that they can beat their previous time by constantly improving and finding the best route to take and the correct speeds to take during the race. This was done by adding a 3D object at the finish line so that if the player collides with it before the AI they will win the game and their time will be saved and shown in the top right corner the next time they play. If they lose, however, their time will not be recorded so it is a requirement to win in order to save lap times. The 3D object at the finish line has its mesh turned off so that it does not look like a wall or a dead-end.

In addition to the 3D object at the finish line. There is a second 3D object at the midway point which also has its mesh turned off. This is to prevent the user from cheating and abandoning the track to take a shortcut to the finish line. The 3D object at the finish line will be disabled until the car has collided with the 3D object at the midway point first. Since the mesh is not visible to the player, this forces the player to go around the whole track to ensure that they do not miss the trigger that activates the finish line. This midway point activates the finish line using scripts. The script which enables the finish line trigger is the CheckpointTrigger script. It does this by using the OnTriggerEnter method which sets the lapCompleteTrigger (which is the trigger at the finish line) to active when car collides with the object at the midway point.

```
public class CheckpointTrigger : MonoBehaviour
{
    public GameObject lapCompleteTrigger;
    public GameObject checkpointTrigger;

    void OnTriggerEnter(){
        lapCompleteTrigger.SetActive(true);
        checkpointTrigger.SetActive(false);

    }
}
```

Figure 1: CheckpointTrigger Script

The player also has the option of colliding with the AI and blocking the AI by driving in front of it. This adds a sense of realism to the game as real races have collisions and crashes between cars which can be used to get ahead of the opponent. Real races also have cars blocking other cars on the track by driving directly in front of them to prevent them getting past. This blocking can allow the car that is in front to maintain their lead even if they need to slow down due to an upcoming turn bad road/track conditions.

The cars can also collide with the rocky edges of the track which is used to simulate a crash in the game and this is what forces the player to maintain a steady and reasonable speed rather than allowing the player to go at full speed which would make the game too easy and not fun to play. The collisions with the edges of the track can cause the cars speed to decrease significantly or even cause the car to come to a complete stop forcing the player to reverse out of the rocky edges and accelerate back to high speeds again which grants the AI an opportunity to overtake the player.

In addition to the race against the AI, players also have the option to improve on their race time. This is possible because the players best race time is recorded in the top right of the screen along with the time they are taking for the current race. This keeps the player interested in the game even if they can consistently beat the AI each time they play. They can use the best time feature to constantly improve on their driving by picking the correct speeds and route to take during the race in order to bring their race time lower. Having the current race time as well as the best race time side by side makes it so that the player can see how close they are to beating their previous record and therefore motivating them to keep playing when they are close to beating it. The lap time is recorded and displayed to the screen using the LapTimeManager and FinishingLap scripts. The LapTimeManager creates the timer that is running when the game is being played and the FinishingLap script uses that timer to see how long the player took to complete the race. The LapTimeManager calculates the duration that the race has currently been going on for by using the Time.deltaTime variable to retrieve the time between frames and summing that variable at every frame and storing the sum in a variable called rawTime. The FinishingLap script then accesses the rawTime variable when the finishing line trigger has been set off in order to see how long the player took to complete the race. This is only calculated when the player wins the race and not when he loses. The best time is then replaced if the time taken for the player to complete the current race is shorter than the current best time. The best time is then saved and loaded for the next game using the PlayerPrefs. The script which loads the best time is the LoadSavedTimes script which uses the Start method retrieve the best times from the PlayerPrefs and display them in the best time display in the HUD.

```
void Update()
{
    milliPassed += Time.deltaTime * 10;
    rawTime += Time.deltaTime;
    milliDisplay = milliPassed.ToString("F0");
    millibox.GetComponent<Text>().text = "" + milliDisplay;

    if (milliPassed >= 10) {
  milliPassed = 0;
  secondsPassed += 1;
}
```

Figure 2: Update method in LapTimeManager Script

The AI itself follows a fixed path that it takes around the track. This path has been made to follow a general route that is not the fastest but at the same time, not the slowest path that can be taken either. This is to ensure that the game is not too easy and not too difficult for the player and provides a difficulty that is reasonable and keeps the player interested and wanting to play again until they beat the AI. The AI can be nudged off the path if the players collides with it but then returns back to the path that has been set for it once it has recovered from the collision. The AI follows a fixed path using waypoints and a target which are implemented using the OpponentTracker script. The opponent's car follows a target object when the game begins. I controlled where this target object is positioned around the track by matching the position of the target object to specific waypoints that are positioned around the track. The target changes its positions to the different waypoints by acting as a trigger with a counter to keep track of which waypoint it is currently on. Each time the AI car collides with the target, the target changes its position to the next waypoint.

```
void Update () {
    if (tracker == 0) {
        Target.transform.position = Waypoint1.transform.position;
    }
    if (tracker == 1) {
        Target.transform.position = Waypoint2.transform.position;
    }
    if (tracker == 2) {
        Target.transform.position = Waypoint3.transform.position;
    }
    if (tracker == 3) {
        Target.transform.position = Waypoint4.transform.position;
    }
    if (tracker == 4) {
        Target.transform.position = Waypoint5.transform.position;
    }
    if (tracker == 5) {
        Target.transform.position = Waypoint6.transform.position;
```

Figure 3: Update method in OpponentTracker Script

3. Non-playable features

The game begins with a menu that has a start and quit button on it. Upon hitting the start button, the player will be presented with the main game scene which begins with a countdown timer for the start of the race. The player cannot move their car during this countdown but can hold down the controls so that they will be immediately executed when the timer reaches 0. This allows the player to have the best start to the game, granting them a better chance of winning. In order to ensure that the cars only begin moving after the countdown is finished, I used C# scripts to disable the player car controller and AI car controller at the beginning of the race and only enable it after the countdown is finished. I did this using the CarStart and Countdown scripts which use the Start method to set the CarController and CarAIControl to "active" only when the countdown timer has reached 0. In addition to creating the scripts, to ensure that the cars are not able to move at the beginning of the game, I had to deactivate the CarController and CarAIController in the unity inspector.

```
void Start () {
    CarControl.GetComponent<CarController>().enabled = true;
    OpponentCar.GetComponent<CarAIControl>().enabled = true;

}
```

Figure 4: Start method in CarStart Script

The player also has their best race time saved so that it can be used as a high score system. The best time is loaded into the start of every game and is shown at the top right of the screen and if the best time is beaten by the player, it gets overwritten so that the new best time is displayed.

In addition to this, the player is presented with a win/lose screen when the game ends depending on if the player reaches the finish line before the AI or not. If the player wins by reaching the finish line before the AI, they are presented with a green screen that says, "YOU WON". However, if the AI reaches the finish line before the player, the player is presented with a red screen that says, "YOU LOST".

4. Audio-visual

My game does not include any background music as it is a simulation game that attempts to imitate real racing. This, therefore, means that there is no music so that the player can have an advantage as they will be able to hear if the opponent is about to overtaking them since they will be able to hear the opponents car engine.

Since this is a simulation game, the cars have sound effects for accelerating and drifting. When driving the car, the player will be able to hear the engine of the car as well as the engine of the opponent car if they are close by. Furthermore, the player will be able to hear the screeching of the tires when there is a hard turn being made.

The car also has particle effects such as smoke that can be seen coming from the tires during drifting and acceleration which adds realism to the game. Skid marks are also left on the track whenever the car is drifting which recreates what would happen on a real track during a race.

As my game is set outside in a valley, the main light source for the game is the directional light. This light source is meant to represent the sun as my game takes place during the day in sunny weather conditions. Another light source in my game are the light sources for the car's headlights and brake

lights. The brake lights can be helpful to the player as they can show when the opponent is slowing down. The player can use this information to adapt their speed so that they have the best chance of overtaking the opponent.

There are 4 cameras in my game. 3 of the cameras are used to provide different driving perspectives for the player for when they are playing the game and can be changed with the ‘c’ key. These 3 camera angles are positioned behind the car and are facing forward relative to the car and include a driver-perspective camera, normal-camera, and a wide-angle camera. These cameras follow the player's car during the race so that the player's camera is always in view. I did this by making the a camera a child of the car object. The cameras also have a script attached to them which stabilizes the camera during bumps so that the camera does not get too distorted during the race which can confuse and disorient the player. The script which does this is called CameraStabilizer and works by calling the Update method which accesses the cars Euler angles and uses those Euler angles to change the transform of the car so that only the Euler angles for Y are affected and the Euler angles for X and Z are stable. The ability to switch between these cameras is done with another C# script.

```
void Update()
{
  carX = car.transform.eulerAngles.x;
  carY = car.transform.eulerAngles.y;
  carZ = car.transform.eulerAngles.z;
  transform.eulerAngles = new Vector3(carX - carX, carY, carZ-carZ);
}
```

Figure 5: Update method in CameraStabilizer Script

This script I used to change between camera angles is called CameraSettings and it uses the update method to detect the keypress of the ‘c’ key. To do this, I added an input key into the Unity InputManager and assigned that key to the letter ‘c’. I then referred to that input that was added to the InputManager to my C# script to detect when the key was pressed and coded a change in the camera angle for when the key was pressed by calling another method called ModeChange. This method sets the next camera angle to be active and sets the current camera angle which was active to be inactive. It was important that I activated the next camera angle before I deactivated the current one to prevent the screen from being blank for a short period of time during gameplay which can be distracting to the player.

```
void Update () {
    if (Input.GetButtonDown ("Viewmode")) {
        if (CamMode == 2) {
            CamMode = 0;
        } else {
            CamMode += 1;
        }
        StartCoroutine (ModeChange ());
    }
}
```

Figure 6: Update method in CameraSettings Script

The 4th camera is used for the mini-map. Similarly, to the other cameras, this camera also follows the player's car but differs in that it is positioned at a birds-eye view. This means the camera is above the car at a far distance, with the camera facing directly downwards. This camera is standard amongst many racing games and can be used by the player in order to keep track of their position and opponent's position in the race. The player can also use this mini-map to prepare their speed for upcoming corners and check how far away they are from the finish line.

I have used animations in my game to show the countdown before the race, as shown in Figure 7. This animation adds a nice aesthetic by animating the numbers during the countdown to rotate from an upright position to a flat position which makes them look like they disappear whilst the next number comes up. I created this animation using the unity animation timeline, as shown in Figure 8. This is accompanied by sound effects for the countdown which makes "beep" noises during the countdown numbers and another similar noise that has a different pitch when the countdown has finished which is used to signify the start of the race. The countdown is controlled by another C# script which is attached to the text object which has the countdown.

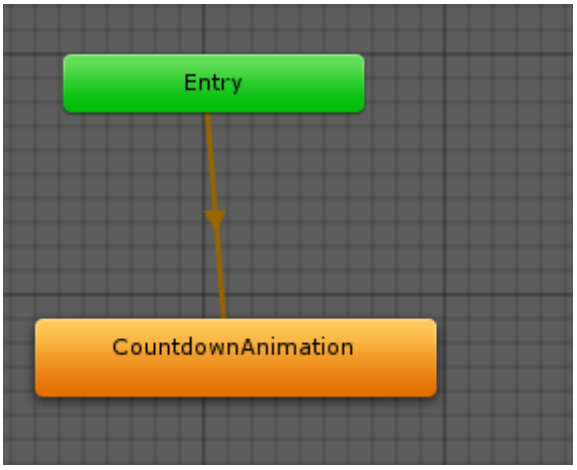

Figure 7: Animation state diagram

Figure 8: Animation timeline

I have used multiple materials for my game including 3 different materials for the terrain which I obtained from online sources [2]. For the mountains and edges of the track, I have used a rocky material. For the track itself, I have used a smooth asphalt material and for the areas around the track, I have used a grass material to in order to create a valley scenery. Furthermore, I have used a checkerboard material for the finish flag, as shown in Figure 9, as that is a standard pattern for all finish flags and will, therefore, make it clear to the player where they should be heading when playing the game. Another material which I have used in my game is a metal material which is used for the poles that hold up the finish flag.

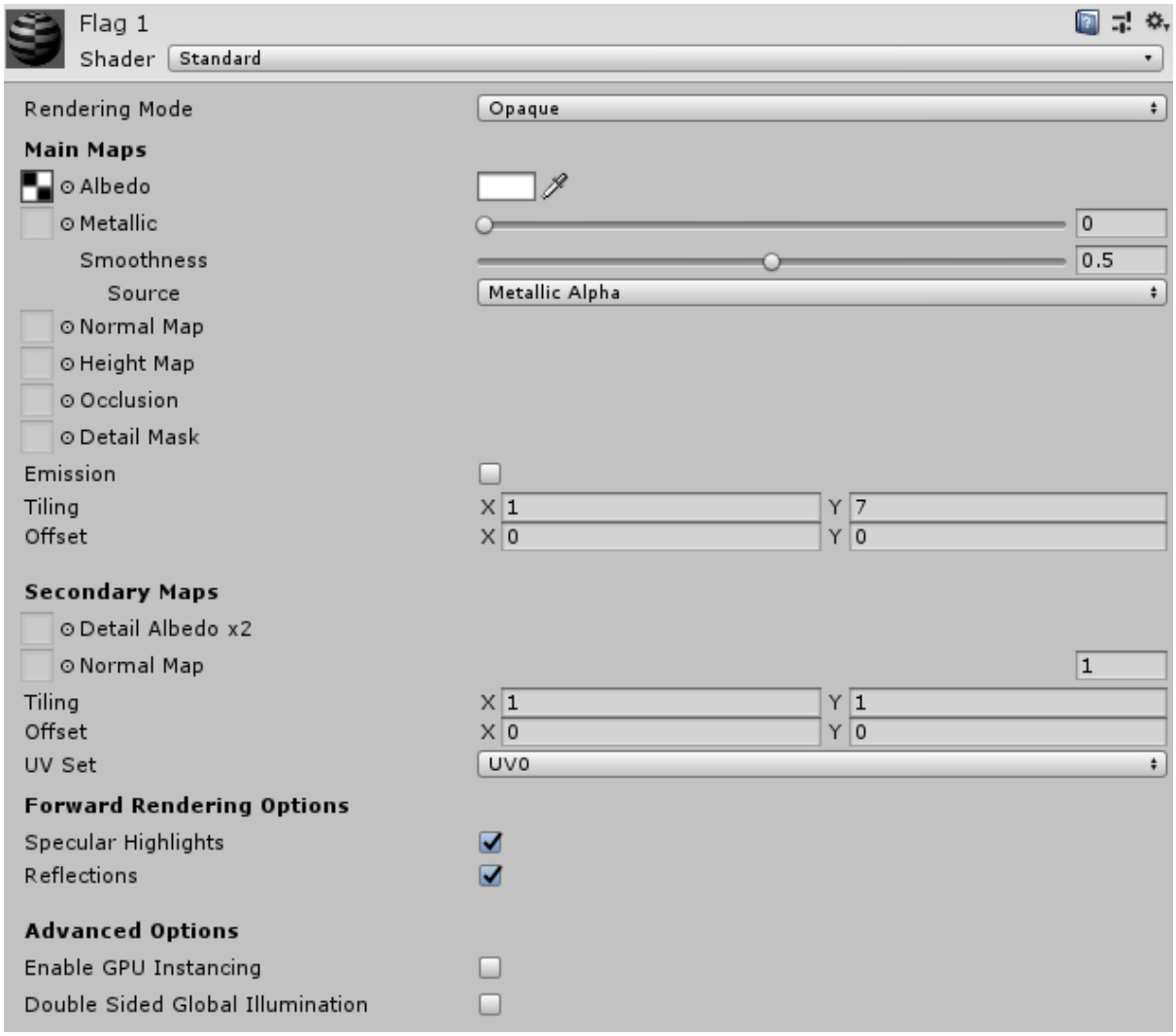

Figure 9: Flag material

Both the players car and the opponent's car also have materials applied to them and the colour for the materials on the car body are different so that it is easier for the player to identify which car is which when looking at the mini-map as being able to distinguish between the 2 cars can be difficult when they are close together if they have the same colour.

5. Game Quality

My game provides an enjoyable experience as it is has a good balance of difficulty that makes the game not too easy but at the same time difficult enough to keep the player interested and prevents them from getting bored. This was done by ensuring the AI that they play against is a good opponent and can beat the player if they do not try to be tactical with their speed and the route they take around the track. This meets the concept idea of "racing against a computer AI" very well as the game grants a competitive aspect on top of the goal of beating the AI since players are presented with their best race time meaning they can play the game multiple times with the intention of beating their best time.

The game, however, could do with more features such as the ability to unlock items and the addition of cut scenes between races to keep the player engaged. Furthermore, I can add more race maps that the player can race on to increase the variety of different tracks that are available to the player.

In addition to this, the car sensitivity can be adjusted to make the car easier to control during harsh corners in the race and I could adapt the AI to perform better or worse in the race depending on the current position of the player in the race in order to make the game more challenging.

6. Acknowledgements

[1]"Standard Assets - Asset Store", *Assetstore.unity.com*, 2019. [Online]. Available: https://assetstore.unity.com/packages/essentials/asset-packs/standard-assets-32351. [Accessed: 2019].

[2]"Azure Series", *Jimmy Vegas Unity Tutorials*, 2019. [Online]. Available: https://jvunity.weebly.com/azure-series.html. [Accessed: 2019].

7. Gameplay Video

https://www.youtube.com/watch?v=isiAuFmscwM