

Car Tag: you’re it

Game Design Document



Scott Ewing - B00371006

1–10-2021

Contents

[Game Overview 3](#_Toc89783690)

[Style & Influences 3](#_Toc89783691)

[Core Objectives 4](#_Toc89783692)

[Features Implemented 5](#_Toc89783693)

[Track Laying system 5](#_Toc89783694)

[Checkpoint System 5](#_Toc89783695)

[Respawn System 5](#_Toc89783696)

[Boost 5](#_Toc89783697)

[Obstacles 6](#_Toc89783698)

[Spinning Mallets 6](#_Toc89783699)

[Slamming Mallets 6](#_Toc89783700)

[Falling Crates 6](#_Toc89783701)

[Distance Driven 7](#_Toc89783702)

[Collectables 7](#_Toc89783703)

[Player Mechanics 8](#_Toc89783704)

[Driving 8](#_Toc89783705)

[Runner Specific 8](#_Toc89783706)

[Slow Pad 8](#_Toc89783707)

[Box Pile 8](#_Toc89783708)

[Chaser Specific 8](#_Toc89783709)

[Controls 9](#_Toc89783710)

[Game Pad Control 9](#_Toc89783711)

[Mouse and Keyboard Controls 9](#_Toc89783712)

[Game Balance 10](#_Toc89783713)

[Code Structure 11](#_Toc89783714)

[Design Pattern 11](#_Toc89783715)

[UML Class Diagrams 11](#_Toc89783716)

[Checkpoint System UML 13](#_Toc89783717)

[Assets Used 20](#_Toc89783718)

[Realistic Car Controller 20](#_Toc89783719)

[Dreamteck Splines 20](#_Toc89783720)

[GUI Pack 20](#_Toc89783721)

[Action RPG Effects 21](#_Toc89783722)

[All Sky 21](#_Toc89783723)

[Unity Particle Pack 21](#_Toc89783724)

[References 21](#_Toc89783725)

# Game Overview

*Car Tag: You’re It* is a two-player local multiplayer racing game which will be played split screen. Each player will control a car and will have one of two rolls, the *Runner,* or the *Chaser.* The game takes place in a large arena with no set road or track, but with many ramps, jumps, loops, and other obstacles. In *Car Tag*, the *Runner* begins driving around the arena first, taking whatever path they like. Then after a time delay, the *Chaser* can begin driving but has to follow the same path the *Runner* took. The *Runner’s* objective is to drive a set distance without getting caught while the *Chaser’s* objective is to catch the *Runner* before they reach this distance.

The game is suitable for players of all ages and best for those who want to play a simple, casual game with a friend.

## Style & Influences

The style of the arena area with no set track was influenced by an Android/IOS game called *Jet Stunt’s 2.* The specific level that influenced the arena was called *Free for All*. In this level, the player drives around doing tricks to get the highest number of points in two minutes. They are free to go wherever they like in the arena.

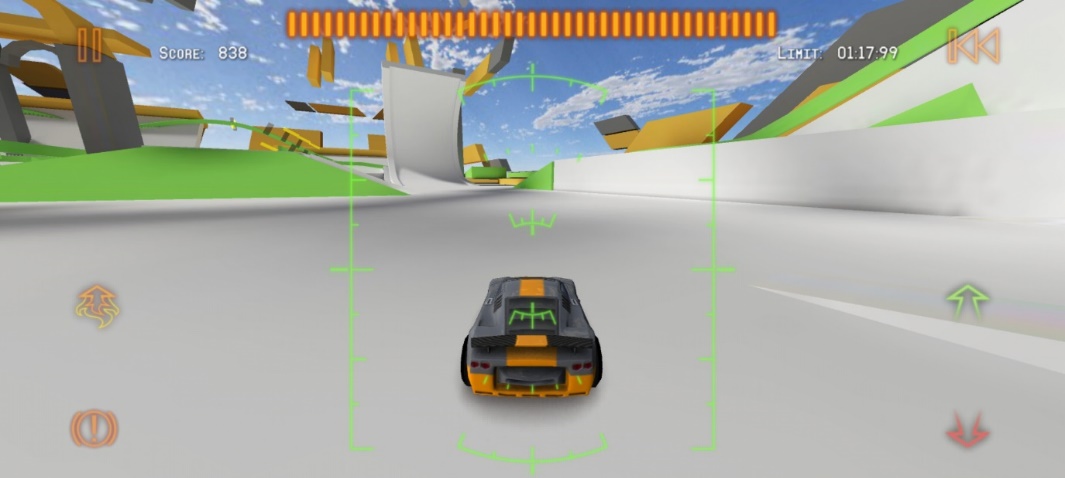


Figure :Jet Stunt's 2 Image 1

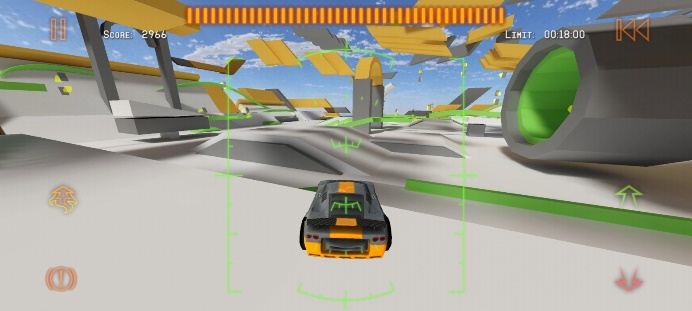


Figure : Jet Stunt's 2 Image 2

# Core Objectives

The winner of the game is the player who wins three rounds first. The flowchart below shows the gameplay loop that takes place during each round.

Assuming Player 1 starts of as the *Runner,* the flow chart shows that when the game starts, Player 1 will be able to begin driving immediately. As Player 1 drives, a road will be generated behind the car. Player 1 can drive anywhere they like around the arena, using the various obstacles to try and make a more difficult path for player 2 to follow.

Figure : Round Gameplay Flow

After a time delay, Player 2 can begin driving as well. They must follow the track that Player 1 left behind and there will be regular checkpoints on the track so Player 2 will not be able to skip sections. To aid Player 2 in catching up they will have a car with faster acceleration, a higher top speed and better handling.

In order to win the round Player 1 (the *Runner*) must drive a set distance without being caught. If Player 1 is **not** caught, then the game will perform a check to see if Player 2 (the *Chaser*) is within a set distance (e.g., 10m) behind Player 1. If Player 2 is too far away the round will end immediately and Player 1 will win.

However, if Player 2 (the *Chaser*) is within this set distance behind Player 1 (the *Runner*) the game will enter an “overtime” mode. The game will repeatedly check if Player 2 has either exceeded this 10m distance behind, in which case the round will end with Player 1 the winner, or until Player 2 catches Player 1. This will result in the players swapping roles and the gameplay loop will begin again with Player 2 (now the *Runner*) able to continue driving and set the track. Meanwhile Player 2 (now the *Chaser*) will stop and have to wait a set time before they can begin Chasing down Player 1

The gameplay loop will always continue until the *Runner* successfully evades the *Chaser,* regardless of whether the runner happens to be Player 1 or Player 2. When a round is won the level will be reset and the next round will start, this time with the opposite Player starting as the *Runner* from who started last round. Once a player wins three rounds, they win the game.

# Features Implemented

## Track Laying system

As mentioned above, while the *Runner* is driving a track will be laid behind them for the *Chaser* to drive on. To generate the mesh for the track I will use an asset from the Unity Store called *Dreamteck Splines.* With *Dreamteck Splines* I can create a spline when the R*unner* begins driving and continuously add additional points on the spline every meter or so that the car travels. I will also be able to generate a mesh for the track which will follow the spline. If the *Runner* goes off a jump, then the mesh will stop generating while the car is in the air and start again when it lands.

## Checkpoint System

As the Chaser is driving nothing will stop them from driving off the track, but in order to make progress in catching up to the *Runner* they must follow the road as it was created thus a checkpoint system will be necessary. In order to dynamically create checkpoints at runtime I can use the points that are being added to the track spline to get the position of the checkpoints. For each point a trigger will be created spanning the width of the track. The triggers must be entered in order and if the player misses any a warning message will be displayed on screen and the part of the track they need to return to will be marked.

## Respawn System

In certain situations, the *Chaser* may come off the track in such a way that they would not be able to get back for a sizable amount of time. For example, if the track goes along-side a ledge and the *Chaser* drove off this ledge, they will then have to drive the long way round to get back to the track. This will most likely result in the chase having no chance of catching the *Runner*. To solve this the *Chaser* will be able to respawn at the last checkpoint they passed through. They will be able to do this any time they like.

## Boost

Both the *Runner* and *Chaser* will have a boost ability which will increase their speed. As they use the boost, a boost meter on the HUD will deplete. For the *Chaser* this boost meter will refill automatically over time, but for the *Runner* it will not. In order to get more boost, the *Runner* must drive through boost pickups which will be located around the arena.

## Obstacles

### Spinning Mallets

These are mallets in the level which will spin continuously. The *Runner* can drive through/round them as they fit trying to make a dificult path for the *Chaser* to follow.

Figure : Spinning Mallets

### Slamming Mallets

These mallets start in a vertical position and then slam down on to the ground, before slowly returning to the vertical position. Just as with the S*pinning Mallets* there will be many of these obstacles located around the arena for the *Runner* to incorporate into their track however they like.

Figure : Slamming Mallets

### Falling Crates

This obstacle is a pile of crates sitting ontop of a platform which is held up by two flimsy legs. The player can drive through these legs causing the platform to collapse and the crates to fall and scatter across the track which will impede the *Chaser.* This obstacle will reset at the start of each new round.

Figure : Falling Crates

## Distance Driven

Each round is won when the *Runner* drives a set distance without getting caught. The distance driven will be displayed on each player’s side of the screen so that they can clearly see how the round is progressing. I have not yet decided whether the distance value will count from zero up to the target distance or vice versa but will try both and see which seams better. Originally, I had planned on having the *Runner* win based on driving for an amount of time rather than distance but decided to change this to improve the gameplay.

If the runner only had to survive a certain amount of time, then a valid strategy would be to just drive in a series of “S” bends which would massively reduce the advantage the *Chaser* gets by having a faster car and allow the *Runner* to just run out the clock. This would most likely result in the game being repetitive and boring.

But, by requiring the *Runner* to survive for a set distance instead, it gives them an incentive to drive in straighter paths where they can rack up the distance faster, but where the *Chaser* will also be able to get more advantage from their higher top speed. This should give the *Runner* an added challenge of trying to get the balance right between these two styles of driving (“S” bends and straight paths) to try and reach the target distance without getting caught.

## Collectables

The game will contain a few collectables which will prominently be picked up by the *Runner*. There will be a finite number of each collectable in the arena and once they are used, they will not respawn until the start of the next round. These pickups include:

* Boost – Refill the players boost meter
* Distance Reduce – Reduce the distance the *Runner* will be required to drive

# Player Mechanics

Since *Car Tag* has a separate role for each player, there will be certain mechanics that are specific to that role. However, both players will share the same driving mechanics.

## Driving

As this is a driving game the player mechanics will be similar to most other driving games. The player controls a car which can drive forwards, backwards, accelerate and brake. The player can also boost to increase their speed.

## Runner Specific

The *Runner* will have some abilities that they can use to try and slow down the player. These abilities will be one time use.

### Slow Pad

This is a pad that can be left on the track behind the runner. If the *Chaser* runs over this pad then they will be drastically slowed down. This pad will stretch over the center of the track so it will be possible for the *Chaser* to avoid it if they react quickly when it is placed.

### Box Pile

This is a pile of boxes that the *Runner* can place behind them as they drive. These boxes will be physics based so the *Chaser* will be able to drive through them. The boxes will take up the full width of the track so it will be very difficult for the *Chaser* to avoid them.

Figure : Box Pile Ability

## Chaser Specific

The chaser will also be able to spawn a pile of boxes, except this pile will spawn in front the of the runner as they are driving. The boxes will spawn far enough in front that it will be possible for the runner to avoid them but only if their reaction are quick enough.

# Controls

## Game Pad Control

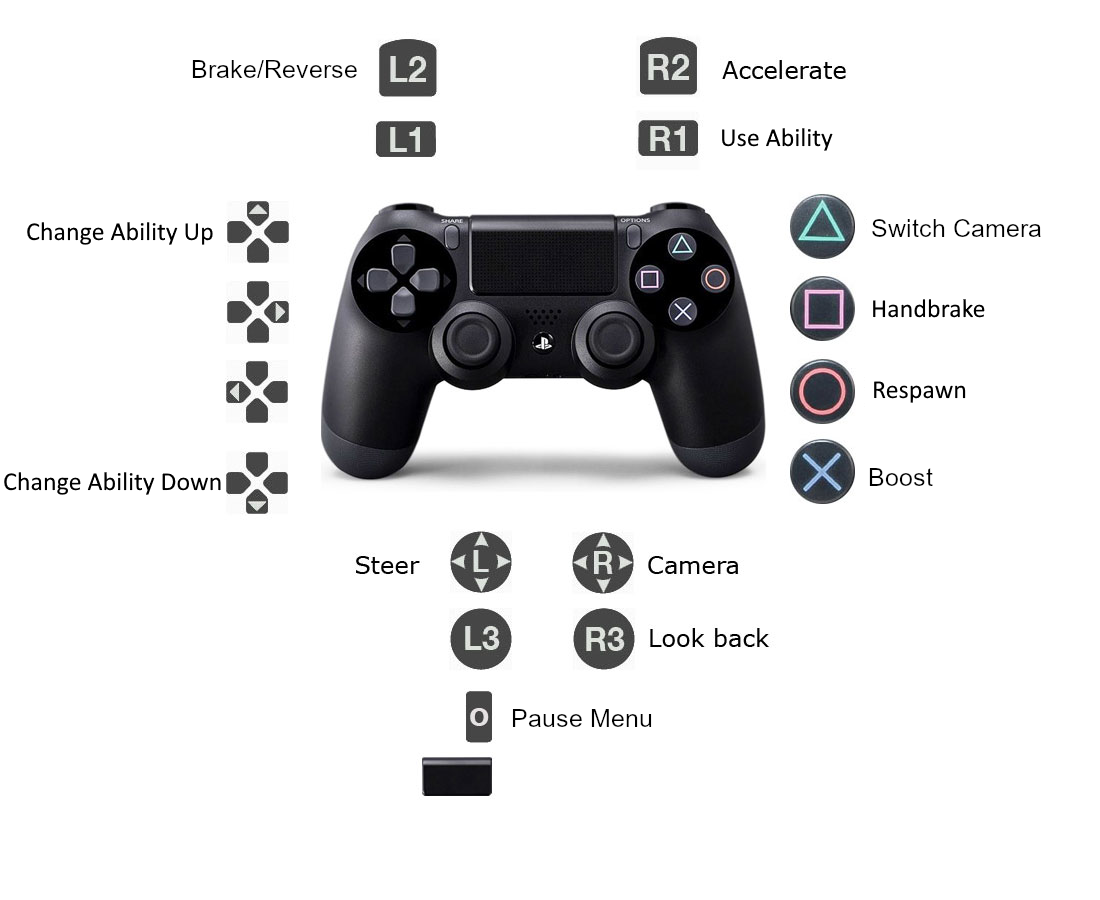


Figure : Gamepad Controls

## Mouse and Keyboard Controls

|  |  |
| --- | --- |
| **Control** | **Input** |
| Throttle | W |
| Brake | S |
| Steer | A/D |
| Handbrake | Space |
| Use Boost | Shift |
| Use Ability | F |
| Change Abilities | Q/E |
| Move Camera | Mouse |
| Switch Camera | C |
| Pause Menu | Esc |

# Game Balance

In an a-symmetric two player game like this it would be very easy for the game to be completely unbalanced. In the case of *Car Tag* case the game would be unbalanced if it was either much easier for the *Runner* to evade the *Chaser*, or too easy for the *Chaser* to catch the *Runner.* While developing the game, if either of these scenarios become true there are a number of ways in which the game could be modified to increase rebalance the game.

* The time delay before *Chaser* can begin driving can be changed
* The top speed, acceleration and handling values can be changed
* Boost refill time for the *Chaser* can be increased or decreased
* The distance the *Runner* is required to reach can be changed
* The quantity of pickups in the arena can be modified

By fine tuning these parameters it should be possible to balance the game in a way which will result in the winner being the most skilful player rather than the player with the easier role.

# Code Structure

## Design Pattern

The game only contains one design pattern, the “Singleton Pattern”. Currently only the *GameManager* class is a singleton. This allows me to access the *GameManager* from any class in the project without having to find or assign it first. The Singleton pattern works by creating a public static variable of the same type as the class itself.

  
In the *Awake* method this variable is assigned to the current instance of the class.

The singleton pattern requires that only one instance of the class exists in the scene. In my game, instead of directly making the *GameManager* a singleton, I made it inherit from a *MonoSingleton* class which turns any class inheriting from it into a singleton.

##### MonoSingleton.cs



## UML Class Diagrams

The full UML class diagram was a little big to display on a single page, so I’ve attached an image of the diagram below.



##### GameManager

For this project I decided to try and keep logic out of the *GameManager.* I have no idea if that was a good idea or not, but basically what happens is the *GameManager* will be told that something has happened, and it will then inform the other manager classes one by one. For example, below is a method on the *GameManager* which get called from the *PlayerCollision* class when the chaser has caught up with runner and has triggered the player roles to be swapped.



Structuring the *GameManager* this way has allowed it to remain quite short and understandable.

##### Road System UML

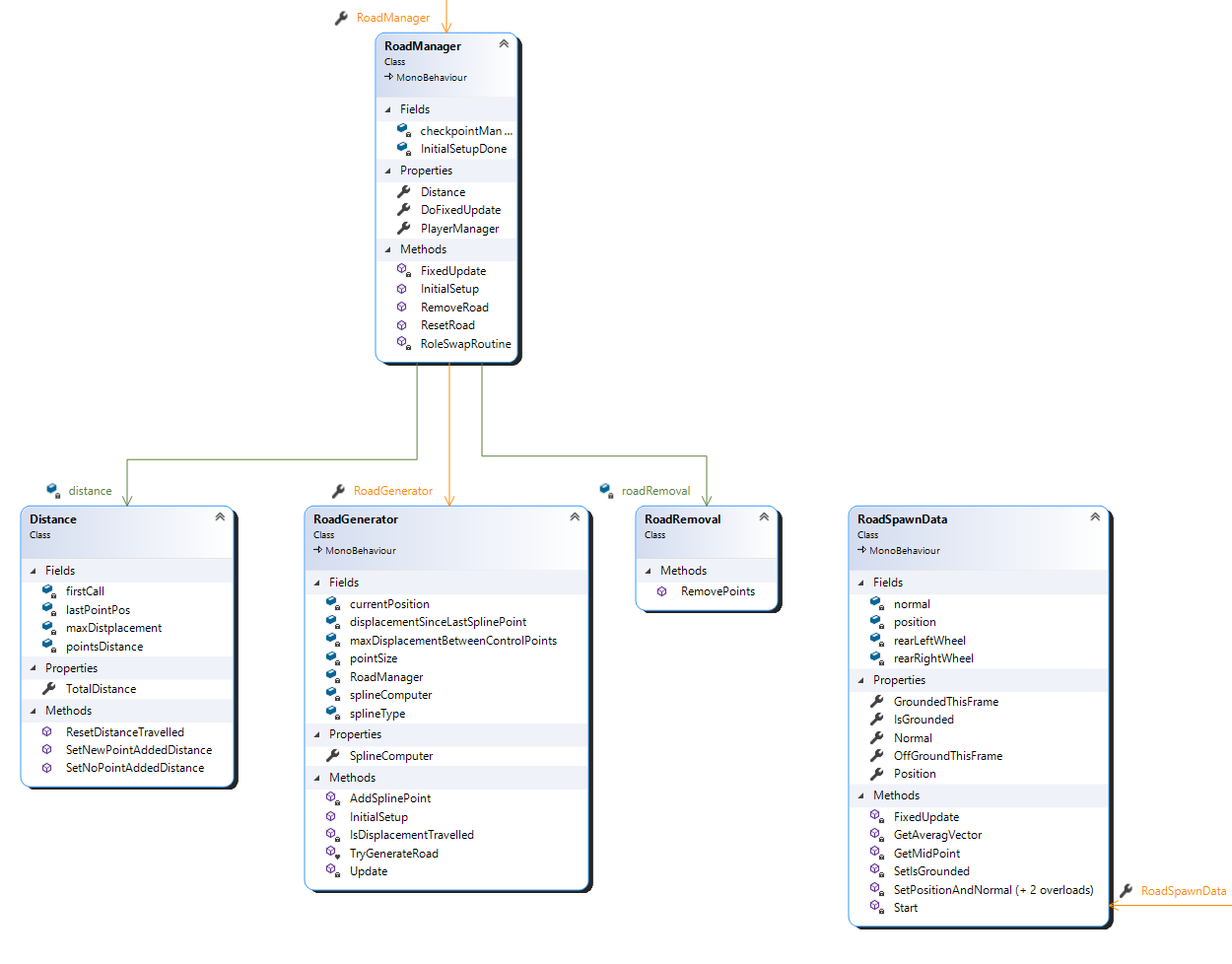


Figure : Road System UML Class Diagram

Every fixed update the *RoadManager* will tell the *RoadGenerator* to try and place a new point on the road using [Dreamteck Splines](#_Dreamteck_Splines). The point can be placed assuming that the car is on the ground and that the car’s displacement from the last point is large enough. Once a point is added, the *RoadManager* will tell the *CheckpointManager* to try and place a checkpoint.

The road system is responsible for keeping track of the distance the runner has travelled which is used by the *RoundManager* to determine when the round has been won. It also removes section of road as the chaser is driving through checkpoints.

### Checkpoint System UML

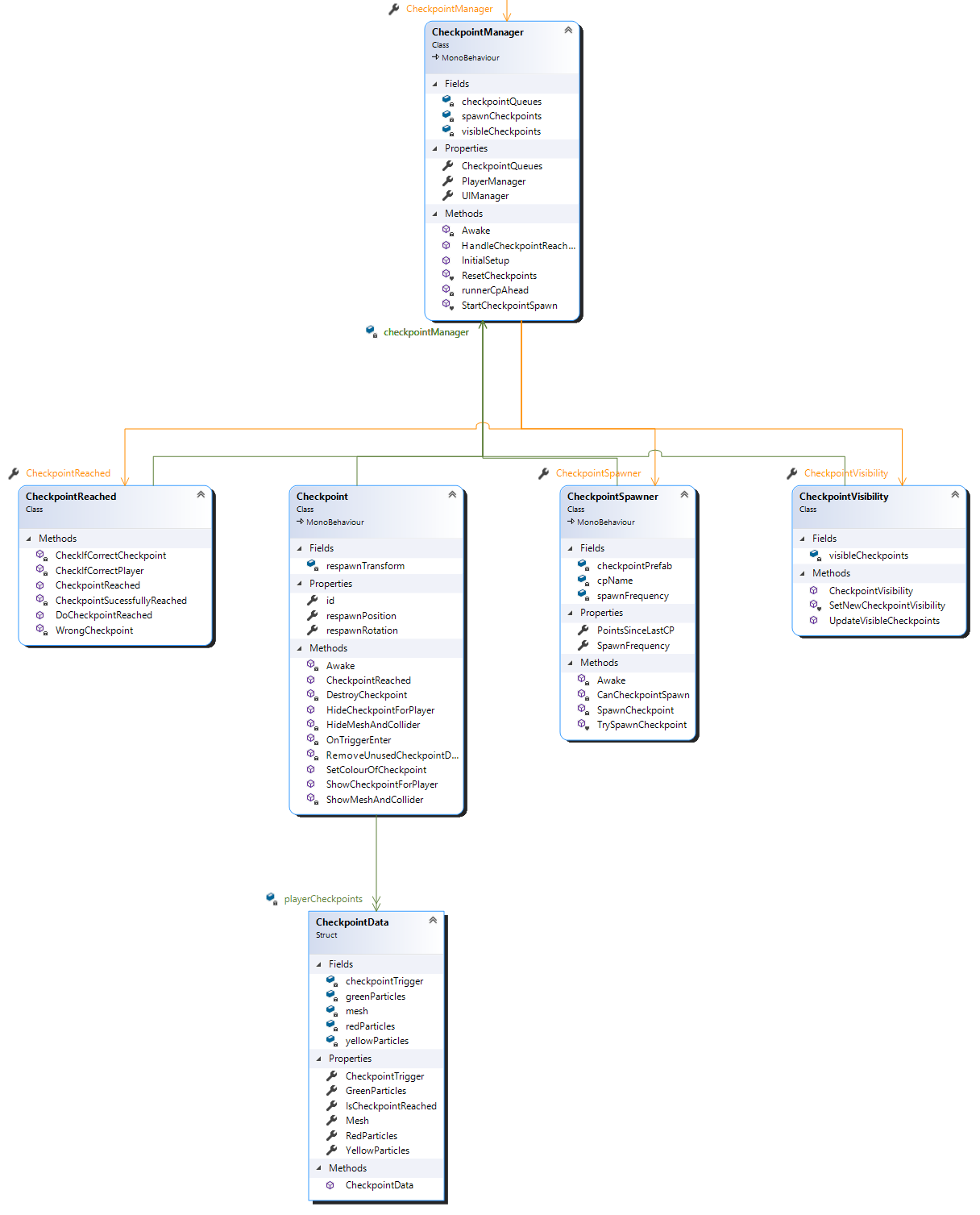


Figure : Checkpoint System UML Class Diagram

Just like all other parts of the game, the Checkpoint System was written to work completely independent of the number of players in the game. Each player in the game has their own queue of *Checkpoints* which allows the progress of each player to be tracked independently. The purposes of the scripts in the Checkpoint system are:

* *CheckpointManager –* Sets up the checkpoint queues for the number of players. Acts as a go between for script inside and outside of the Checkpoint System.
* *CheckpointSpawner –* Is called from the RoadManager every time a point is placed on the spline. Will spawn a checkpoint every set number of points.
* *CheckpointVisibility –* Controls which checkpoints are visible as only a few checkpoints should be visible. Different checkpoints will be visible to the separate players since the player may not be on the same checkpoint. This is done by having the camera attached to the player only able to see certain layers.
* *CheckpointReached –* Used whenever a player collides with a checkpoint and determines if the player collided with the correct checkpoint.
* *Checkpoint –* This is the script which is attached to the Checkpoint Game object in the scene. Detects when a player has collided with it and informs C*heckpointReached.* If the checkpoint should be visible it is responsible for enabling the correct particle effect. Each checkpoint has three particle effects for each player which correspond to the order of the checkpoints. Each set of three particles are only visible to one player.

##### Player UML

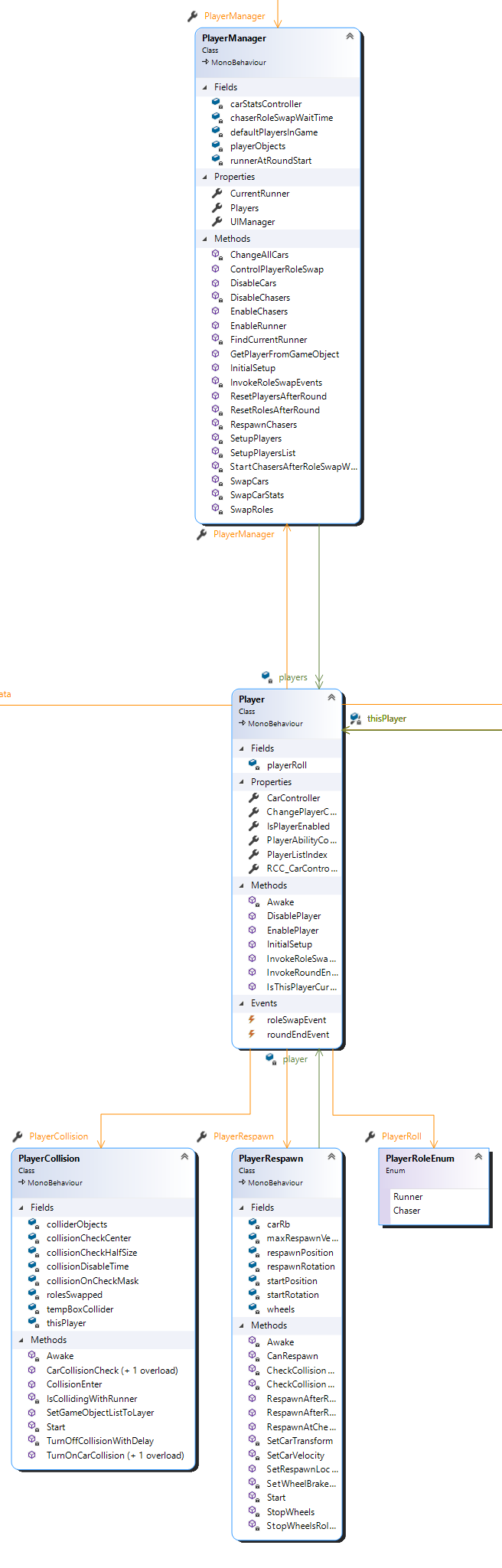
The *PlayerManager* is used when changes must be made to multiple players at once e.g., to reset all the players after the round ends.

Figure :Player UML Class Diagram

There is one *Player* script on each player. It contains a reference to a number of other scripts which are attached to the Player game object including, *PlayerUIController, PlayerAbilityController, PlayerCollision.* Each of these classes will also have a reference to the player which means the are able to communicate with each other by going through the central *Player* class.

My reasons for setting it up like this is that I thought it would make more sense for scripts on the player to access each other via another script on the same player rather than having the scripts access each other via their respective manager classes. Especially since I would like to try and make the game online multiplayer compatible at some point and my understanding is that the manager classes would run server side. So, if the classes on the player were to try and access each other via the managers there would presumably be more latency that if the classes accessed each other through the *Player* class.

##### Ability System

Currently the game contains three abilities, not including the boost which is separate from the Ability System and can be used regardless of which ability is selected. These Abilities are:

Figure : Ability System UML Class Diagram

* *BoxSpawnAbility*
* *RocketAbility*
* *SlowTimeAbility*

The Ability System has been programmed so that it should be easy to add additional abilities in later. Each new ability can inherit from the *Ability* class or if it if the ability will need to spawn an object in the scene it can inherit from the *SpawnableAbility* class. The Ability System will support abilities which are only available when the player is a specific role or abilities which will work regardless of the role.

* *AbilityManager –* This contains a list of *PlayerAbilityController*’s and is used when changes need to be made for all players at once, for example each time the chaser catches the runner the roles will swap, and number of uses left for each ability must be reset.
* *PlayerAbilityController –* This controls aspects of the Ability System for a single player. There are one of these scripts attached to each of the player game objects in the scene. It is responsible for keeping track of the ability which is currently selected. It will be told to try and activate the ability by the *PlayerInputHandler* and will determine if the ability can be started.

##### UI System

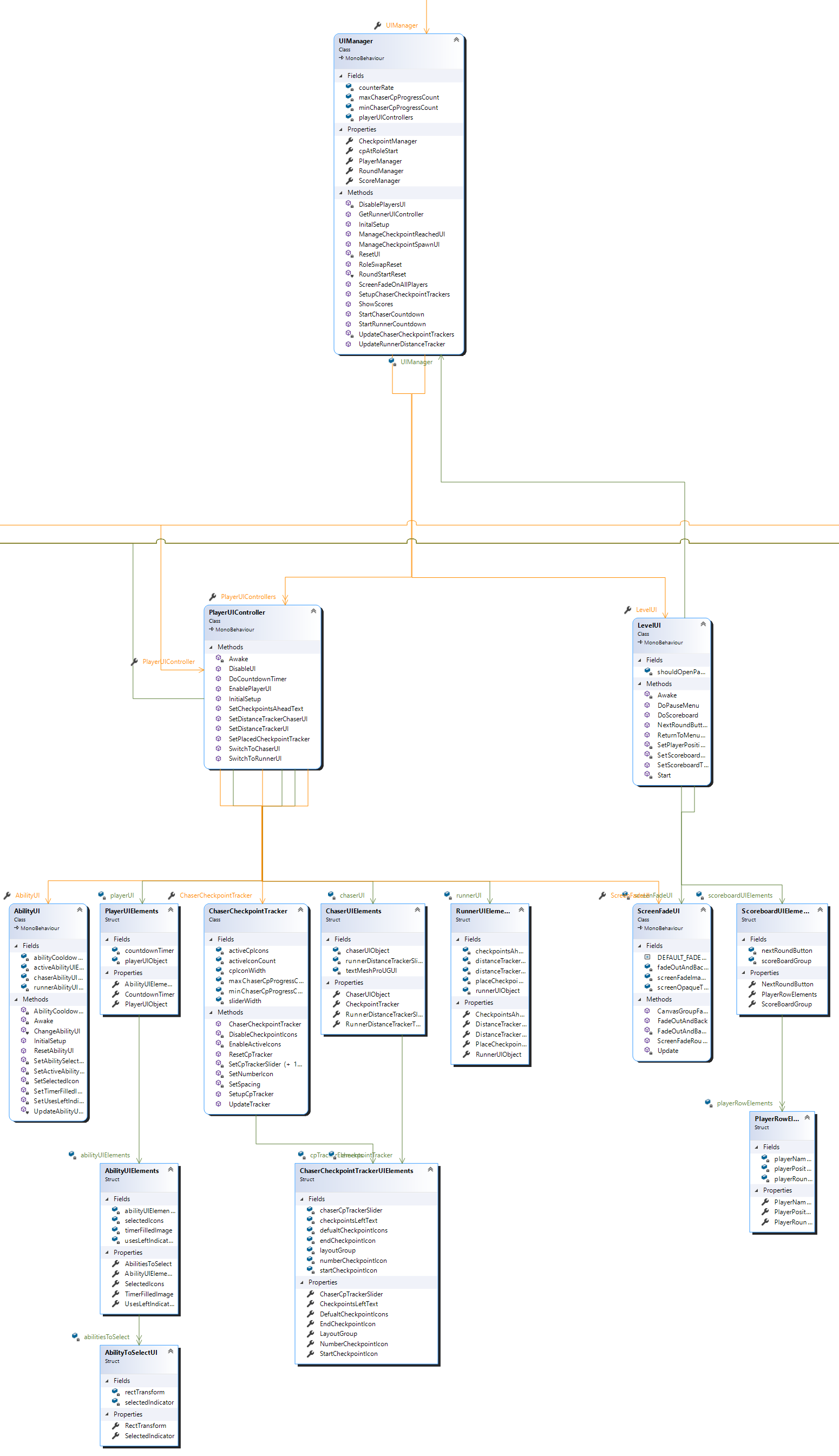


Figure :UI System UML Class Diagram

The UI System follows a similar structure as the Ability System in that the *UIManager* controls changes which must be made to all players’ UI at once and the *PlayerUIController* controls changes which are made to a single player’s UI. The three main features of the UI System are the:

The UI that the player will change as they swap from one role to the other.

#### Runner UI



Figure : Runner UI

###### Top Left – Checkpoint Spawn

A bar which fills up while the player drives, a checkpoint will be placed when its full. The runner can use it to try and place checkpoints in the middle of a turn forcing the chaser to go through the full turn.

The number to the left of the bar shows the runner how many checkpoints behind the leading chaser is.

###### Bottom Left – Ability UI

This is the Ability UI. The icon in the green circle signifies the currently selected ability. The other circles above this are the abilities available the player. This will be different depending on the player’s role. When an ability is used the green circle will turn grey and will gradually turn green again. This indicates the time the player must wait before using another ability.

Outside the green circle is a yellow circle with 4 segments. The number of segments which are lit up yellow show the player how many uses left they have of the currently selected ability. Different abilities can have a different number of uses left

###### Top Middle – Distance Tracker

This displays the distance the runner has to travel to win the level. The bar fills up as the runner is driving.

###### Bottom Right – Speedometer and Boost

Speedometer shows current speed of car. The Runner is unable to use the boost ability so the boost UI really shouldn’t be on the Runner UI screen.

#### Chaser UI



Figure : Chaser UI

###### Top Left – Runner Distance Tracker

The same as the Runner distance tracker bar on the runner UI but a smaller version

###### Bottom Left – Ability UI

This works the same as the ability UI on the runner, the only different being that the chaser has an extra ability that they can use

###### Bottom Right – Speedometer and Boost

Speedometer shows current speed of car. The boost UI show how much boost the chaser has left.

###### Top Middle – Checkpoint Tracker

This number in the circle shows how many checkpoints the chaser still has to go through. Every time the runner spawns a new checkpoint this circle will move to the next yellow marker on the left. When the chaser goes through a checkpoint the circle will move to the next yellow icon on the right. When the circle. This allows the chaser to visualize if they are losing progress as well as if they are gaining on the runner.

After the round has started and the runner has had their head start. The checkpoint tracker UI will be adjusted depending on the number of checkpoints the runner was able to place during their head start. For example, if the runner placed six checkpoints then the UI will look like this:



Figure : Chaser Checkpoint Tracker Six Checkpoints Left

If the runner placed three checkpoints the UI will look like this.



Figure : Chaser Checkpoint Tracker Three Checkpoints Left

# Assets Used

This section will list the assets in the game which were not created by me.

## Realistic Car Controller

Asset Link: <https://assetstore.unity.com/packages/tools/physics/realistic-car-controller-16296>  
Developer: <https://www.bonecrackergames.com/>

This asset is used for both cars in the game as well as for the camera controls. Unfortunately, the new Unity input system was not supported by the asset, so I had to modify the code to ignore the input code that was in the asset and write my own to use the new input system.

The other elements of the asset which were used in the game include a Nitro ability as well the Nitro and Speedometer UI. Each of these had to me modified to work with my existing ability and UI code.

##### City Scene

The city scene used in the game was also from this asset

## Dreamteck Splines

Unity Store Link: <https://assetstore.unity.com/packages/tools/utilities/dreamteck-splines-61926>  
Developer: <https://assetstore.unity.com/publishers/21142>

Dream text splines allowed me to create a path behind the car as it drove along. I first I wanted to the car to leave a road behind as it drove, but while this did work it meant that a lot of new mesh was constantly being created which resulted in low framerate once the road got to a certain length. So instead, I decided to create the path behind the player with the Line Renderer that came with Dreamteck splines as appose to the Mesh Renderer. This fixed the framerate issues at the cost of looking less fancy.

## GUI Pack

Unity Store Link: <https://assetstore.unity.com/packages/2d/gui/gui-pro-kit-fantasy-rpg-170168#content>  
Developer: <https://layerlabgames.com/>

The was used for the UI in the game with the exception of the *Nitro* and *Speedometer* UI.

## Action RPG Effects

Unity Store Link: <https://assetstore.unity.com/packages/vfx/particles/action-rpg-fx-38222>  
Developer: <https://assetstore.unity.com/publishers/8569>

Used for the checkpoint and rocket particle effects.

## All Sky

Unity Store Link: <https://assetstore.unity.com/packages/2d/textures-materials/sky/allsky-free-10-sky-skybox-set-146014>  
Developer: <http://www.richardwhitelock.com/>

The Skybox in the game

## Unity Particle Pack

Unity Store Link: <https://assetstore.unity.com/packages/essentials/tutorial-projects/unity-particle-pack-127325>  
Developer: <https://assetstore.unity.com/publishers/1>

Explosion for player rocket.

# References

Control’s image - <http://www.gamecontrols.net/ps4/mafia-iii/>