RentRace State and Roadmap

In its current state, RentRace consists of 2 Levels, a title screen and a tutorial screen.

**Animation:**

In this project, animation of sprites was done frame-by-frame using custom written sprite swapping code. This code is called SpriteAnimation.cs. you attach it to an object with a sprite renderer and then load in the sprites in Unity. In that code, you can set the ideal framerate for the project. It would probably be good to make the rate a public variable that can be set on a instance-by-instance basis.

**Player Movement:**

Player movement is controlled by the player swiping the in direction they want to go. Node prefabs should be placed on every corner of valid areas of the map. This allows the player to make more accurate moves and also queue actions if the player swipes too early.

**Pathing:**

Pathing is achieved using A\* pathing package

**Spawns:**

In the code, any enemy type is called a “Mugger”.  
Muggers are any Mob/Entity that pursue the player.

Muggers use A\* pathing to pursue the player, and can also chose to head in a random direction every now and then to try and flank the player.

Muggers are activated by a mugger spawner object which also manages their behaviour.

Coins are created at load-time by a coinspawner object which creates many copies of a single coin placed in the scene. These coins are dynamically assigned a location by randomly selecting a coordinate and then checking if the point intersects with other collision boxes. If so, then it selects another point randomly until a valid spot is found.

**Win and Loss:**

Win and loss tracking is done by an object called the WinController.  
the WinController listens out for certain conditions to be met so that It can activate the notification windows. When those windows are activated, deltaTime is made equal to 0 to immobilise all entities.

**Text Displays:**

Text displays pull their information using a script called PullScore.cs.

In the case of the “-40” text that appears when a player hits an enemy, this text is activated from the player in ScoreManagerAndInteraction.cs, the text object itself then runs a script that waits for 2 seconds to pass and then turns itself off again.

**Map:**

The map in this project is actually an image that I have overlayed collision boxes onto. This approach is good for rapid development and accutate hitboxes, but not really ideal for expansion. Ideally the maze could be rebuild using individual objects, but for the sake of rapid development, I opted for the quicker method.