1. (Already completed).

2. This piece of code brings the init function into play through the simulation. It keeps the car-green function.png and keeps it within the self.sprite attribute. This way it can load it for future uses on the map when put into play. It is converted to be easier for ‘pygame’ to create and form it. The ‘CAR_SIZE’ attribute uses ‘pygame transformations’ to scale and help it fit effectively on the map which is also stored within self.sprite. Positioning of the car is formed by self.position and the angle is set to 0, for the default position before the sensors are put into play alongside the movements. Most of this is placed into the self.center attribute. Self.radars helps initialise the formation of the radars and sensory part of the car so it can identify the range of it’s movements when progressing down the track. Self.alive helps make sure the car knows it hasn’t ‘crashed’ at the start. Self.distance helsp the car record the distance it progresses by and the time is recorded under self.time. Essentially this part starts to bring the car to life and brings it into it’s surroundings. It brings the agent (the car) into the simulation and teaches the agent of it’s surroundings while bringing it’s characteristics into the game.

3. The draw attribute has two main parameters within it. ‘Self’ is one which implicates the class and it’s job, and screen, which is the canvas and screening, linking to the environment factor of machine learning. This is for the race track. The screen.blit section of the code helps form the imaging or background while taking it from another source to bring it into ‘pygame’. The self.rotated part refers to the fact that the image may have been previously rotated. The screen is used as an argument in draw_radar and makes it functional to the code.

4. Once again, there are two parameters that are the same as in the last piece of code. The first part of the code represents the sensors visually. There is a for loop and the radar is determined through it’s positioning. ‘Pygame’ is used to draw the lines in, in a green format. Then the circle is formatted and placed on the warning points or points of potential curiosity, like the edges of the track that the car must not hit. This function fleshes out the environment and agents interaction at last.

5. This piece of the code refers to the end point of the entity, or the crash function. When self.alive is equal to true, the entity is considered alive or having not crashed. A loop is introduced where corners are thought to be rectangles. These points are presumed the x and y coordinates. There is a link created between border color and sensors. If the pixel is the correct color, the car is rewarded and gets to continue on. The loop is broken when the border_color is incorrect and the car will ‘crash’. Self.alive will then equal false. This brings the crashing theme into play while introducing the reward sector of the game, to live on and continue.

6. This is the true part where the sensors come into play with the role of self.alive. Check_radar can determine finds the distance from one point to another and answers the code’s question of whether the car has crashed or not. Length is originally set to 0. The degree function works with the x and y coordinates to bring realisation to where the car is in terms of position. The rays starting point is made. A while loop is formed so it should keep going until the conditions are met: The x and

y's coordinate is incorrect (which links back to border_colour), or the length goes past the maximum set range (300 units). This information is regularly updated as the car moves forwards, with links to length and direction. The final distance and self.center position is calculated out and the radars take note of where they are so they can attempt to avoid that obstacle in the future, which some cars seem unable to comprehend in the final map. This part of the code further explores the crashing component with the agent and it's environment.

7. This is the code section that focuses on 'updating' the system and all it's attributes so the car can progress forwards with its attributes following it. The speed is initialised and set to 20 as a general rule. Mathematical equations are then used to update the car progress by it's speeds and angle. Trigonometry is used. The distance travelled by the car is then used to improve the speed progressively. The corners and border of the track is also calculated. Collisions are checked to ensure the car is supposed to still be moving forwards and the check-radar function is ensured to be working successfully to work out the car's position through angles and the radars. This mainly links to the progress of the code and the agents eventual learning process.

8. The common parameter, 'self', is brought up again in this code alongside get_data, to calculate and get results for the distance of the sensors. It brings the attribute self.radars and connects it to the radar variable. This can collect and store the information for their position and work. The sensors information is also stored in the return_values function which has a start value of [0, 0, 0, 0, 0]. the enumerate function is then used further for the index, essentially (i) and the actual radar. It gets te distance and divides it. The int() function then makes it into a integer to replace the values for the return_values section. This gains all the information needed to successfully progress and gives this data to the radars after using maths to determine it's position within the track. The reward is continuously created through this one still as the car gets the values converted for it so it can continue forth. The agent also depends on it for it aims to effectively help guide the sensors that will then go on to guide the car in its quest far down the track.

9. 'Self' is used as a parameter again and this is simply to recheck the status of the car and whether it is in fact, 'alive or dead'. This is then forth returned to give the program the fill on whether that specific entity is still continuing or not. This is checking on the agent and it's progress on the environment.

10. A method is made called get_reward, with the same parameter but is is now an argument. There is a commented out part that suggests a change for the way it is rewarded. The reward for the code is gifted by the coordinates of the car and the distance. CAR_SIZE is brought back alongside the self.distance records to be used to influence the car's distance and how this changes the reward sizing for the agent. This has obvious ties to the reward section of machine learning in that it is gifting the agent in terms of it's progress forth on the track and it's being still on the map.

(All these comments were targetted to the code above it, I’m not sure if that is how it was meant to be formatted however.)