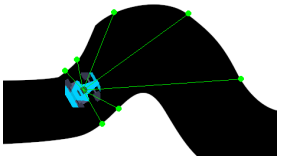
Inquiry topic: Input nodes

Research question: What is the relationship between the amount of input nodes in a machine learning algorithm for cars and their top fitness after 5 generations?

Aim: To determine the effects of changing the number of input nodes on the top fitness of the cars after 5 generations.

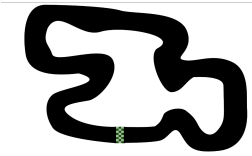
Note: There is only one independent variable in this investigation which is the number of input nodes. It is possible to fine tune the rest of the algorithm to suit a certain number of nodes more, however this investigation does not cover that.

Note: Generations begin counting from 1.

Independent variable: The number of input nodes. The algorithm's default number is 5. Input nodes are the distance between the car and the edge of the track, spaced out from one another like this: 

Dependent variable: Top fitness level after 5 generations

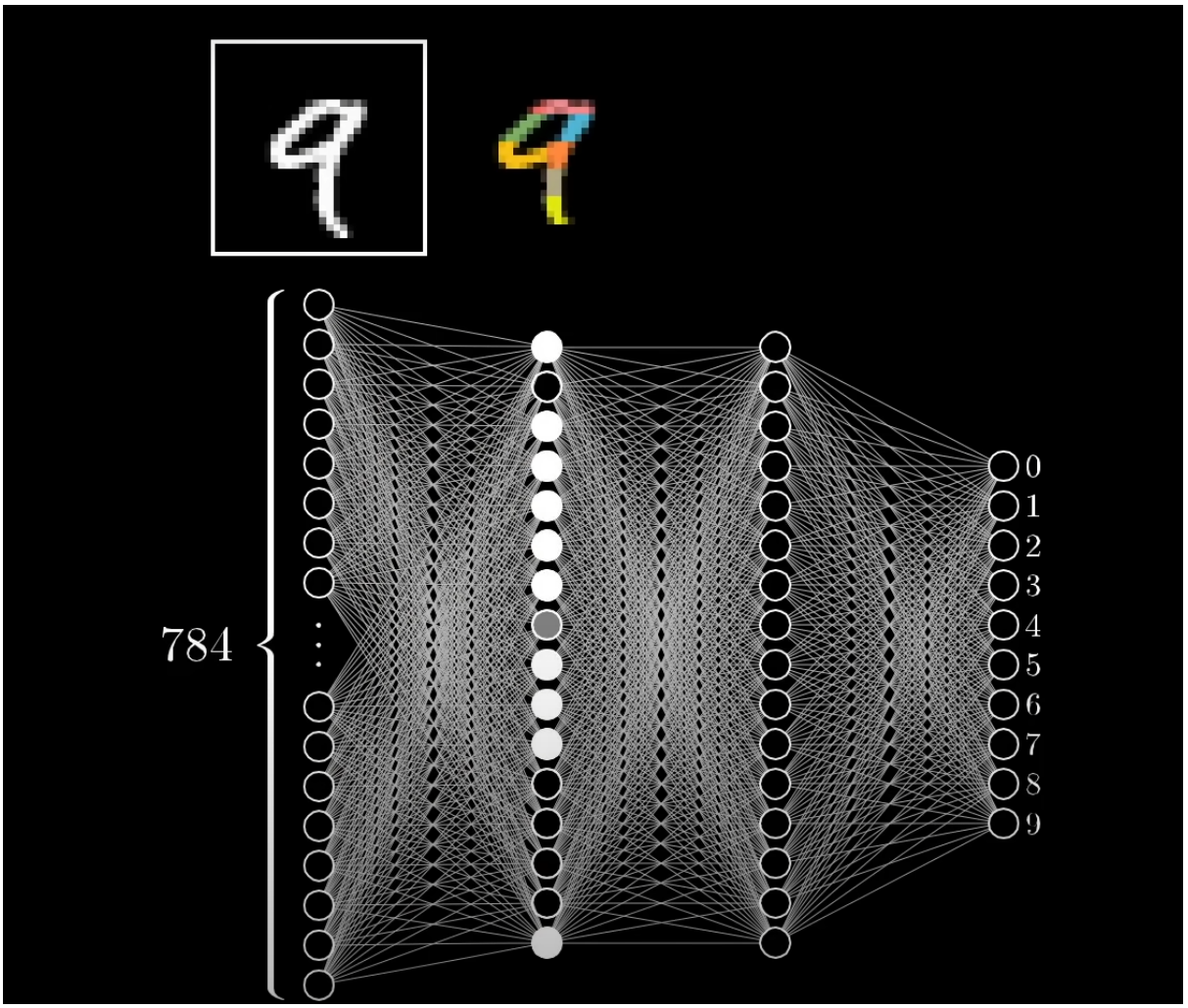
Controlled variables:

- Population (30 cars)
- Map (map2.png) 
- Number of output nodes (4)
- Generations (5)
- The entirety of the rest of the code; only three lines are being modified to change the number of input nodes.

Background Information

What are input nodes?
Input nodes are one of the layers in a neural network, which consists of an input layer, hidden layers, and then an output layer. Within the hidden layer each node has a specific weighting which determines which output the neural network gives.

In a Video by 3Blue1Brown he illustrates these nodes by showing the layers as columns of nodes:



On the left are the input nodes. There are 784 of them in this example because there are 784 pixels in the image. The input itself is a number between 0 and 1 of how bright a pixel is.

On the very right are the output nodes which are numbered 0 through 9 because in this example it is trying to identify handwritten numbers between 0 and 9. Each one of these gets assigned a value of 0 to 1 which is dependent on how the hidden nodes in between interpret the input nodes.

In the current case of AI cars, the input nodes are the distances between the wall and the car for each radar, for example with 5 radars there are five input nodes. The output nodes are the four actions the car can do: speed up, slow down, rotate left, and rotate right. By changing the number of input nodes in this example, the hidden nodes will be trained to interpret them differently and therefore have different outputs to certain situations such as a particular corner.

What is fitness?

The dependent variable in this investigation is the fitness of the best performing car after five generations. In this case, fitness is the measure of how much distance the car covers before either all the cars die off or the time limit ends for the particular generation.

Method

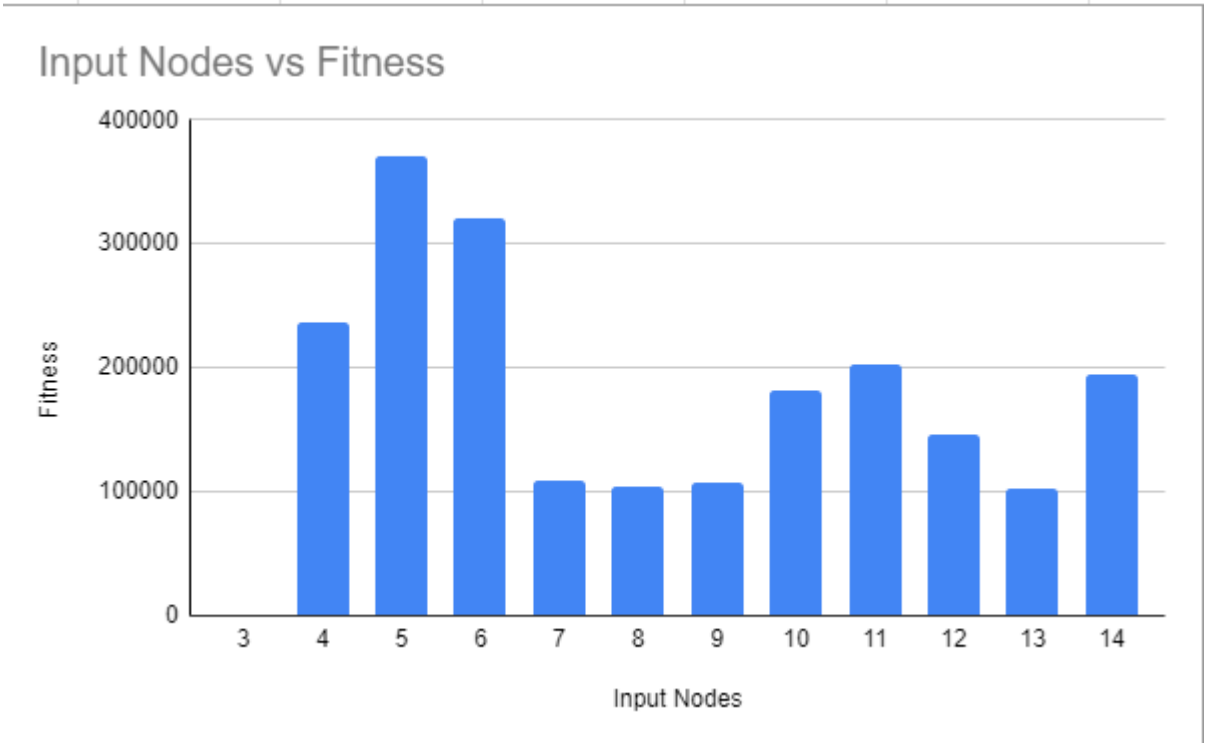
For each number of nodes I am experimenting with, I change the three lines of code required to make this change which are shown in the table “Radar configurations for each number of nodes”

Results

Table of results

Nodes	Test 1 fitness	Test 2 fitness	Test 3 fitness	Average fitness
3	675.1	371.1	424.7	490
4	176118.7	55080.7	480400.0	237200
5	290155.7	342619.0	480400.0	371058
6	480400.0	3653.9	480400.0	321485
7	14119.9	22661.9	289358.7	108714
8	289039.3	12351.1	10255.7	103882
9	13265.3	289081.7	18138.7	106829
10	294431.7	7056.7	242607.7	181365
11	289039.3	289039.3	32370.1	203483
12	136891.7	10502.0	290386.5	145927
13	15401.2	289438.3	746.7	101862
14	4813.7	289358.7	290792.2	194988

Graph of Input Nodes vs Fitness



This graph clearly shows that having five nodes performed better than the other options, while having more than six meant quite a large dropoff in performance.

A qualitative explanation for this is that the populations were overfitted to certain corners, where the cars could capably turn left but turning right was an issue. The risk of overfitting is one of the negative impacts of having too many input nodes.

With only three input nodes it is clear that there was not enough input for the populations to figure out any patterns with input, output, and fitness.

Conclusion and direction of further study

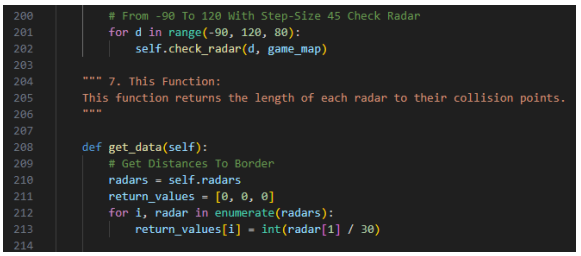
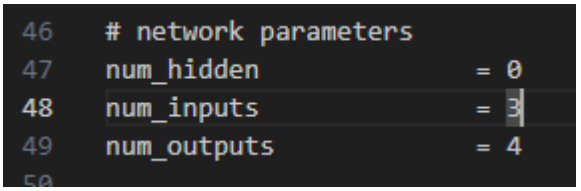
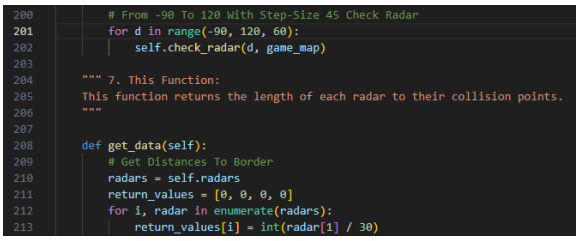
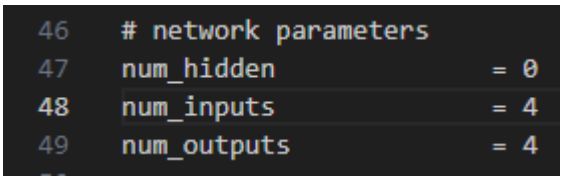
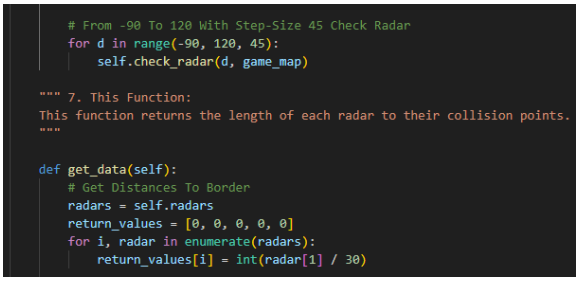
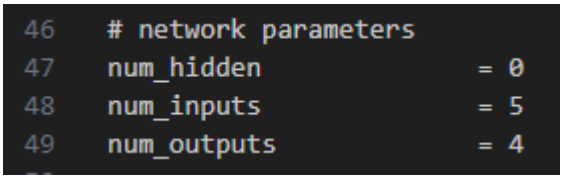
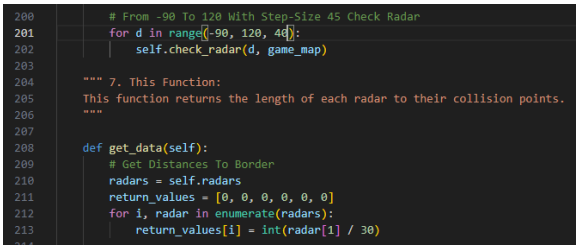
The results concluded that it is important to choose the right amount of input nodes for a certain algorithm because it can make a large impact on how well it performs. By having too many or too little input nodes can raise the issue of overfitting or the algorithm not recognising anything that it can do to raise its fitness and evolve.

One strength of this experiment was that it was easy to measure the effects of changing the independent variable in both a qualitative and quantitative way, which made interpreting the data easier as I could see both visual qualitative evidence and quantitative evidence from the program of overfitting occurring.

One weakness of the test was that the results were quite varied within each tested number of nodes. A potential solution to this is to run several tests simultaneously to get more data and to also make the process more efficient.

For further investigation it would be beneficial to repeat this test with different neural networks for different problems to find patterns with overfitting when the number of input nodes is not suited to the program.

Radar configurations for each number of nodes

3	 
4	 
5	 
6	

```
        # From -90 To 120 With Step-Size 45 Check Radar
        for d in range(-90, 120, 80):
            self.check_radar(d, game_map)

    """ 7. This Function:
    This function returns the length of each radar to their collision points.
    """

    def get_data(self):
        # Get Distances To Border
        radars = self.radars
        return_values = [0, 0, 0]
        for i, radar in enumerate(radars):
            return_values[i] = int(radar[1] / 30)
```

```
# network parameters
num_hidden                      = 0
num_inputs                      = 3
num_outputs                     = 4
```

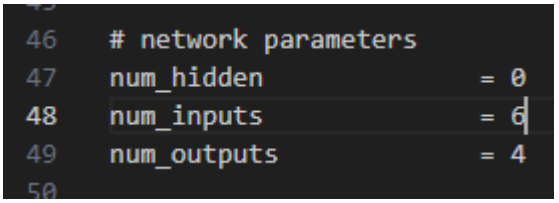
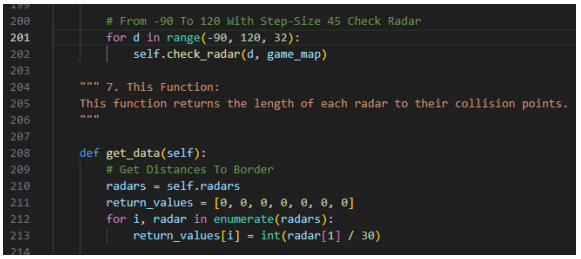
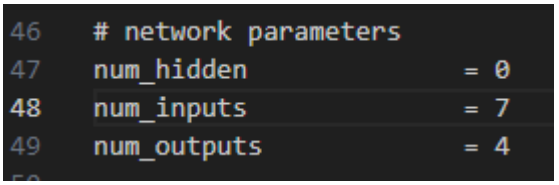
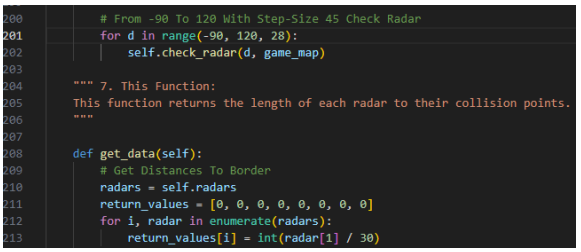
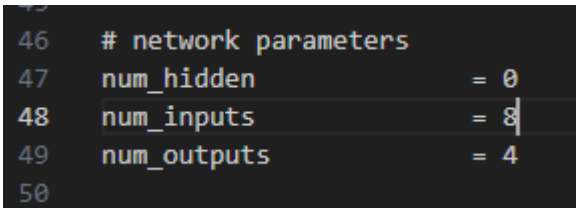
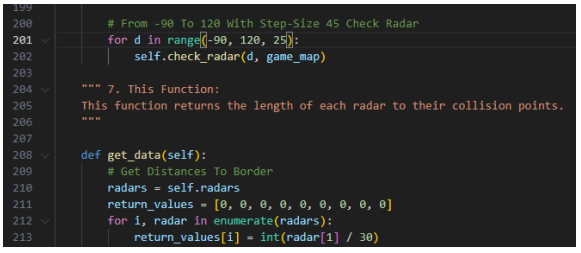
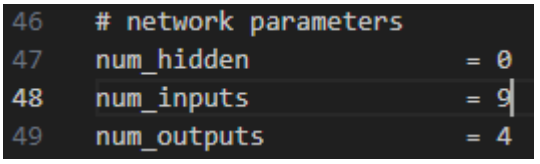
```
        # From -90 To 120 With Step-Size 45 Check Radar
        for d in range(-90, 120, 60):
            self.check_radar(d, game_map)

    """ 7. This Function:
    This function returns the length of each radar to their collision points.
    """

    def get_data(self):
        # Get Distances To Border
        radars = self.radars
        return_values = [0, 0, 0, 0]
        for i, radar in enumerate(radars):
            return_values[i] = int(radar[1] / 30)
```

```
# network parameters
num_hidden                      = 0
num_inputs                      = 4
num_outputs                     = 4
```

```
    # From -90 To 120 With Step-Size 45 Check Radar
    for d in range(-90, 120, 45):
        self.check_radar(d, game_map)

"""" 7. This Function:
This function returns the length of each radar to their collision points.
"""

def get_data(self):
    # Get Distances To Border
    radars = self.radars
    return_values = [0, 0, 0, 0, 0]
    for i, radar in enumerate(radars):
        return_values[i] = int(radar[1] / 30)
```

```
# network parameters
num_hidden                      = 0
num_inputs                      = 5
num_outputs                     = 4
```

```
        # From -90 To 120 With Step-Size 45 Check Radar
        for d in range(-90, 120, 40):
            self.check_radar(d, game_map)

    """ 7. This Function:
    This function returns the length of each radar to their collision points.
    """

    def get_data(self):
        # Get Distances To Border
        radars = self.radars
        return_values = [0, 0, 0, 0, 0, 0]
        for i, radar in enumerate(radars):
            return_values[i] = int(radar[1] / 30)
```

	 46 # network parameters 47 num_hidden = 0 48 num_inputs = 6 49 num_outputs = 4 50
7	 200 # From -90 To 120 With Step-Size 45 Check Radar 201 for d in range(-90, 120, 32): 202 self.check_radar(d, game_map) 203 204 """ 7. This Function: 205 This function returns the length of each radar to their collision points. 206 """ 207 208 def get_data(self): 209 # Get Distances To Border 210 radars = self.radars 211 return_values = [0, 0, 0, 0, 0, 0, 0] 212 for i, radar in enumerate(radars): 213 return_values[i] = int(radar[1] / 30) 214  46 # network parameters 47 num_hidden = 0 48 num_inputs = 7 49 num_outputs = 4 50
8	 200 # From -90 To 120 With Step-Size 45 Check Radar 201 for d in range(-90, 120, 28): 202 self.check_radar(d, game_map) 203 204 """ 7. This Function: 205 This function returns the length of each radar to their collision points. 206 """ 207 208 def get_data(self): 209 # Get Distances To Border 210 radars = self.radars 211 return_values = [0, 0, 0, 0, 0, 0, 0, 0, 0] 212 for i, radar in enumerate(radars): 213 return_values[i] = int(radar[1] / 30)  46 # network parameters 47 num_hidden = 0 48 num_inputs = 8 49 num_outputs = 4 50
9	 200 # From -90 To 120 With Step-Size 45 Check Radar 201 for d in range(-90, 120, 25): 202 self.check_radar(d, game_map) 203 204 """ 7. This Function: 205 This function returns the length of each radar to their collision points. 206 """ 207 208 def get_data(self): 209 # Get Distances To Border 210 radars = self.radars 211 return_values = [0, 0, 0, 0, 0, 0, 0, 0, 0] 212 for i, radar in enumerate(radars): 213 return_values[i] = int(radar[1] / 30)  46 # network parameters 47 num_hidden = 0 48 num_inputs = 9 49 num_outputs = 4

10

```
        # From -90 To 120 With Step-Size 45 Check Radar
        for d in range(-90, 120, 23):
            self.check_radar(d, game_map)

    """ 7. This Function:
    This function returns the length of each radar to their collision points.
    """

    def get_data(self):
        # Get Distances To Border
        radars = self.radars
        return_values = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
        for i, radar in enumerate(radars):
            return_values[i] = int(radar[1] / 30)
```

```
# network parameters
num_hidden              = 0
num_inputs              = 10
num_outputs             = 4
```

11

```
        # From -90 To 120 With Step-Size 45 Check Radar
        for d in range(-90, 120, 20):
            self.check_radar(d, game_map)

    """ 7. This Function:
    This function returns the length of each radar to their collision points.
    """

    def get_data(self):
        # Get Distances To Border
        radars = self.radars
        return_values = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
        for i, radar in enumerate(radars):
            return_values[i] = int(radar[1] / 30)
```

```
# network parameters
num_hidden              = 0
num_inputs              = 11
num_outputs             = 4
```

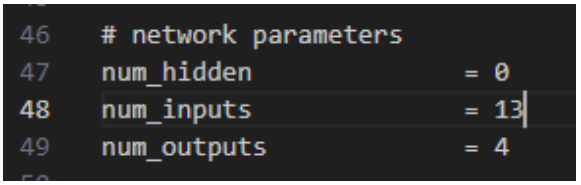
12

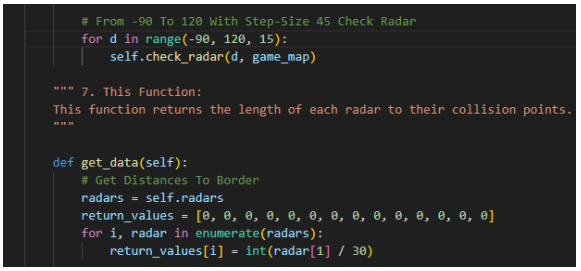
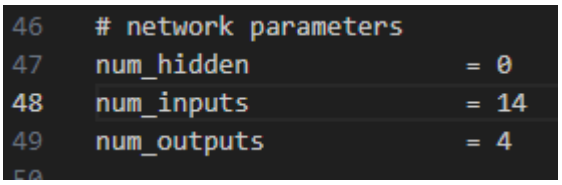
```
        # From -90 To 120 With Step-Size 45 Check Radar
        for d in range(-90, 120, 19):
            self.check_radar(d, game_map)

    """ 7. This Function:
    This function returns the length of each radar to their collision points.
    """

    def get_data(self):
        # Get Distances To Border
        radars = self.radars
        return_values = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
        for i, radar in enumerate(radars):
            return_values[i] = int(radar[1] / 30)
```

```
# network parameters
num_hidden              = 0
num_inputs              = 12
num_outputs             = 4
```

13

```
        # From -90 To 120 With Step-Size 45 Check Radar
        for d in range(-90, 120, 17):
            self.check_radar(d, game_map)

    """ 7. This Function:
    This function returns the length of each radar to their collision points.
    """

    def get_data(self):
        # Get Distances To Border
        radars = self.radars
        return_values = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
        for i, radar in enumerate(radars):
            return_values[i] = int(radar[1] / 30)
```

	 46 # network parameters 47 num_hidden = 0 48 num_inputs = 13 49 num_outputs = 4

14	 # From -90 To 120 With Step-Size 45 Check Radar
	for d in range(-90, 120, 15):
	self.check_radar(d, game_map)
	""" 7. This Function:
	This function returns the length of each radar to their collision points.
	"""
	def get_data(self):
	# Get Distances To Border
	radars = self.radars
	return_values = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
	for i, radar in enumerate(radars):
	return_values[i] = int(radar[1] / 30)
	  46 # network parameters
	47 num_hidden = 0
	48 num_inputs = 14
	49 num_outputs = 4

References

3Blue1Brown. (2017, October 6). *But what is a neural network? | Chapter 1, Deep learning* [Video]. YouTube. https://www.youtube.com/watch?v=aircAruvnKk&list=RDCMUCYO_jab_esuFRV4b17AJtAw&index=2&ab_channel=3Blue1Brown

3Blue1Brown. (2017, October 17). *Gradient descent, how neural networks learn | Chapter 2, Deep learning* [Video]. YouTube. https://www.youtube.com/watch?v=IHZwWFHWa-w&list=RDCMUCYO_jab_esuFRV4b17AJtAw&index=2&ab_channel=3Blue1Brown

3Blue1Brown. (2017, November 4). *What is backpropagation really doing? | Chapter 3, Deep learning* [Video]. YouTube. https://www.youtube.com/watch?v=Ilg3gGewQ5U&list=RDCMUCYO_jab_esuFRV4b17AJtAw&index=4&ab_channel=3Blue1Brown

NeuralNine. (2021, February 18). *Self-Driving AI Car Simulation in Python* [Video]. YouTube. https://www.youtube.com/watch?v=Cy155O5R1Oo&ab_channel=NeuralNine