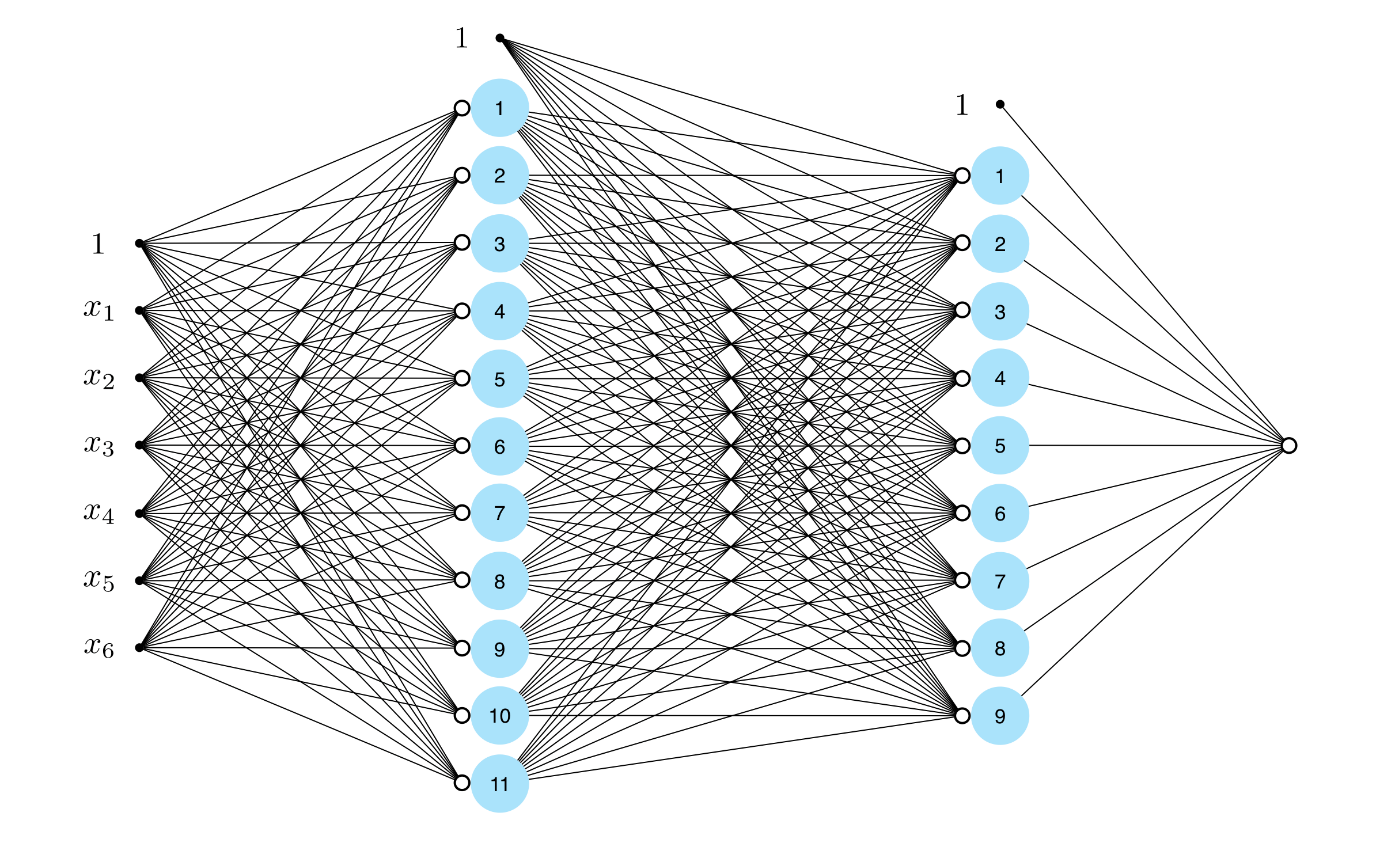
Autonomous 2D Car Simulation



Sean Dougan

Centre: 12532

Candidate: 7074

CONTENTS

[Analysis 5](#_Toc102501368)

[Project Background 5](#_Toc102501369)

[Potential end users 5](#_Toc102501370)

[Descriptions of similar systems 6](#_Toc102501371)

[Interview with end user 7](#_Toc102501372)

[Requirements and Limitations 8](#_Toc102501373)

[Software: 8](#_Toc102501374)

[Libraries: 8](#_Toc102501375)

[Acceptable Limitations: 8](#_Toc102501376)

[Proposed Solution 9](#_Toc102501377)

[Main Objectives 10](#_Toc102501378)

[Design 11](#_Toc102501379)

[Flowcharts 11](#_Toc102501380)

[Main Flowchart 11](#_Toc102501381)

[Hierarchy Charts 12](#_Toc102501382)

[Main Menu 12](#_Toc102501383)

[Map Design 12](#_Toc102501384)

[Data Flow Diagrams 13](#_Toc102501385)

[Level 0 DFD 13](#_Toc102501386)

[Level 1 DFD 13](#_Toc102501387)

[Description of main algorithms 14](#_Toc102501388)

[Map Design 14](#_Toc102501389)

[Neural network algorithm 18](#_Toc102501390)

[File Organisation 20](#_Toc102501391)

[Radars 21](#_Toc102501392)

[Generating map 22](#_Toc102501393)

[Technical Solution 24](#_Toc102501394)

[Table Of Skills 24](#_Toc102501395)

[Structure of program 25](#_Toc102501396)

[Code 26](#_Toc102501397)

[main.py 26](#_Toc102501398)

[MapDesign.py 30](#_Toc102501399)

[Stack.py 34](#_Toc102501400)

[radar.py 35](#_Toc102501401)

[Car.py 37](#_Toc102501402)

[newcar.py 40](#_Toc102501403)

[CT.py 43](#_Toc102501404)

[neural.py 45](#_Toc102501405)

[population.py 47](#_Toc102501406)

[simulation.py 51](#_Toc102501407)

[borderLineGenerator.py 57](#_Toc102501408)

[Testing 61](#_Toc102501409)

[Testing table 61](#_Toc102501410)

[Test evidence 64](#_Toc102501411)

[Test 1 64](#_Toc102501412)

[Test 2 65](#_Toc102501413)

[65](#_Toc102501414)

[Test 3 66](#_Toc102501415)

[Test 4 67](#_Toc102501416)

[Test 5 68](#_Toc102501417)

[Test 6 69](#_Toc102501418)

[Test 7 70](#_Toc102501419)

[Test 8 71](#_Toc102501420)

[Test 9 72](#_Toc102501421)

[73](#_Toc102501422)

[Test 10 74](#_Toc102501423)

[Test 11 74](#_Toc102501424)

[76](#_Toc102501425)

[Test 12 77](#_Toc102501426)

[77](#_Toc102501427)

[77](#_Toc102501428)

[Test 13 77](#_Toc102501429)

[Test 14 79](#_Toc102501430)

[Test 15 80](#_Toc102501431)

[Evaluation 81](#_Toc102501432)

[Evaluation table 81](#_Toc102501433)

[Final interview with end user 84](#_Toc102501434)

[Bibliography 85](#_Toc102501435)

[Videos that helped me gain understanding of deep learning: 85](#_Toc102501436)

# Analysis

## Project Background

Autonomous cars or self-driving cars are vehicles that can sense their environment and react by moving safely with no human input. These do not require human input and allow them to get from one point to another without having to drive. It would require heavy application of artificial intelligence to program, particularly machine learning. Many car companies such as Waymo are working on developing these cars.

The problem is that these self-driving cars are not widely available yet, as there are still flaws with the idea of a fully autonomous car driving on real streets. Although autopilot is available for some cars, such as Tesla models, they have still not fully developed a car that can fully drive from one place to another by itself.

There are also many ethical issues that arise when a self-driving car is on the road. For example, if the car is presented with an ethical problem, such as choosing the safety of one party over another, decisions have to be made on what to do.

Autonomous cars use neural networks and genetic algorithms to steer around a track, by using sensors as inputs to decide what action should be taken at each moment. My goal was to find a way to use machine learning to make a generation of cars learn how to drive around a user-made track. Simulation is one of the most important factors in developing real self-driving cars, and as stated by Luca Castignani of MSC Software, ‘AI is something that lives and breathes in computers. The right place to build, train and validate this technology is the computer’.

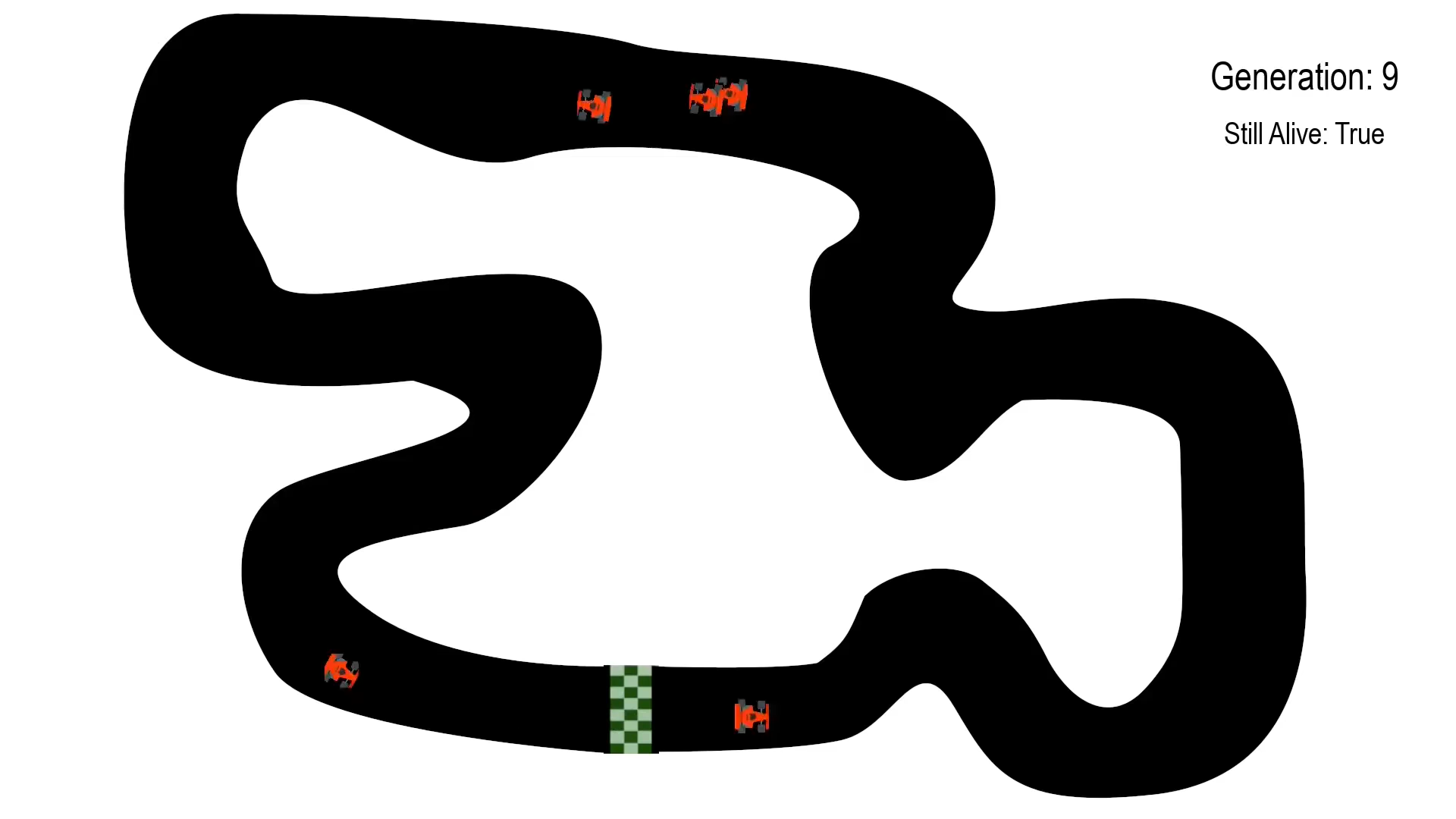
To help demonstrate how neural networks can be used with algorithms to create a self-driving car, I decided to create a 2D simulation to represent this.

## Potential end users

Ashley Dougan, a client, wants a new self-driving car simulator that he can use to study the primary concepts needed to develop a self-driving car, such as the application of neural networks. To do this, he needs to see how the program would work to conduct his research. This project is being made so that he can take it and develop it to work towards creating a self-driving car. In an interview, he described some of the requirements needed for the project.

## Descriptions of similar systems

*Figure 1*



Description: This python program is a car simulation where cars learn to drive around a track without crashing using the NEAT python module (NeuroEvolution of Augmenting Topologies), an algorithm which creates its own neural networks. As a car dies, its fitness is saved, and when all cars have crashed, a new generation is created which uses previous cars as parents. The track is created using GIMP, a GNU Image Manipulation Program, which can work like Microsoft Paint. The program displays the number of generations and whether the cars are still alive.

Analysis:

|  |  |
| --- | --- |
| **PROS** | **CONS** |
| The algorithm can optionally show green distance radars to show the inputs being put into the first layer of the neural network. | As the track is being drawn with only colours, it has essentially one dimension and only senses colour. This would be hard to implement in 3d using this method, as a real car would need to be able to sense borders. |
| Due to the NEAT python module, the cars improve each generation, and become faster after each run. | The program does not show how many cars are alive. |
| On some maps, it takes a very long time to complete the track. | Due to the nature of my proposed program, I cannot use the NEAT python module, so I must generate my own neural network. |
|  | There is no option to add or remove obstacles. |

## Interview with end user

Key: Q = Question ; A = Answer

Q: What are your requirements with this project?

**A: I need you to create a self-driving car simulation that features generations of cars that use neural inputs to determine actions for the autonomous cars.**

Analysis of response: I will need to create a neural network and use a generic algorithm to create the simulation. To promote realism, I will use a car png file rather than a simple shape like a circle or triangle. To run the simulation, I will need to use the Pygame module.

Q: What extra features would you like to be included?

**A: An option for the user to create a map, and to add or remove obstacles will be needed, as well as displaying stats for research.**

Analysis of response: I will need to use the Tkinter module to allow the user to create a simple map, and add options to add or remove obstacles. To remove actions, I may need a stack algorithm for the Last in First out approach. To display a menu for this, I will also need to use the Tkinter module.

Q: How will you make the cars get better at going around the track each time?

**A: I will save the best car for each generation and use it to create each generation after. This will ensure that as the generation number increments, the cars are more likely to achieve a higher fitness.**

Analysis of response: I will need to create car populations that mutate from the best car from the previous generation during each new generation created. I plan to achieve this with my population class in my program.

## Requirements and Limitations

### Software:

The software I will use for this program is the Python programming language in Microsoft Visual Code. Importing code from GitHub to VS is much easier than using Python IDLE, so it would be convenient to pass my code onto end users using this as well.

### Libraries:

Pygame – The Pygame library will be used to make the car simulation; it makes it easier to display shapes and make animations, as well as make the simulation more aesthetically pleasing for end users. Moreover, the Pygame functions are essential to making this project work.

Tkinter – The Tkinter library will be used for most of the GUI; The main menu will be coded using Tkinter functions, especially buttons as well as options for creating the user’s map.

Math – The Math module will be used for sine and cosine operations revolving angles for car turns. Although I could create classes for these without the Math module, it is much easier to use it and save time, especially as it is part of the Python Standard Library.

### Acceptable Limitations:

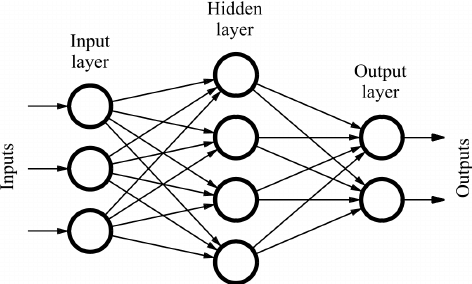
There are limitations which affect the usability of this program. Firstly, a 3D simulation would be more useful for autonomous car researchers, as it is closer to realistic. However, due to limits of time, and my aim to use as little external python modules or programs as possible, this would not be possible under such a short time frame.

## Proposed Solution

Neural Networks and Deep Learning

A neural network algorithm emulates the processes of a human brain. It features single neurons which are connected to each other, the first layer taking inputs and the final layer processing outputs. In this case, my neural network will use the values of the car radars (which calculate distances to borders) as inputs, then, through what we call a feed forward neural network, will go through the hidden layer and finally the output layer. The output neuron that has the greatest value will be my final output, and the algorithm will react based on this.

In our project the outputs will be the angle change and acceleration



Map Creation

As I will be using the Tkinter module, I will need to create two classes for map creation. One that defines how a user can interact with buttons, for example clicking a square to make it part of the track, or selecting a starting track. The other will define operations such as Undo, Clear All, and Save Map. To make the undo functionality I will create a module that depicts the operations of a Stack, which is an abstract data type. With this module it will be easy to pop operations, so I can undo actions this way, due to a stack’s First In Last Out nature.

The map will be saved by compressing it to RLE, then saving it in a file location under the pickle extension. More on this will be touched in the design section of the report.

## Main Objectives

1. Designing a display menu
   1. Display option to design a map
   2. Display option to add/remove obstacles from maps
   3. Display option to run simulation
   4. Quit program
2. Creating a user-designed track
   1. Allow user to design shape of map with a Tkinter GUI
   2. Allow user to place obstacles
   3. Allow user to undo/redo actions using a stack module
   4. Allow user to select start and end locations
3. Allowing user to create a map
   1. Display entry box for map size
   2. Should display a map creator window, with the correct size and an option to select blocks
   3. Must prompt user to select a starting block
   4. User should be able to save map and choose the location and file name
   5. Map should save to a folder called ‘Maps’
4. Creating a neural network
   1. Network should be able to detect wall boundaries with 5 sensors
   2. Distance from wall edges will be considered as inputs to each starting neuron
   3. Outer layers of neural network should be commands to turn left, turn right, slow down or speed up
   4. Outputs of input neurons will determine which command will be executed
5. Cars should run around the chosen track as quick as possible without crashing
   1. Cars that move the furthest gain a higher fitness
   2. Cars that move the fastest gain a higher fitness
   3. Higher fitness cars regenerate in the next generation
6. Allowing user to run car simulation
   1. Neural networks should trigger commands turn left, right, speed up or slow down
   2. Algorithm senses when the edge of a track is hit by a car
      1. This car will die and be deleted
   3. Algorithm selects cars with the highest fitness and regenerates them when cars die
   4. Algorithm runs until a car makes it through the track
7. Display map and stats interface
   1. Decompress map from PKL file and display in pygame window
   2. Display cars left for each generation
   3. Display generation number
8. Allowing user to quit
   1. Exits the program when option is selected.

# Design

## Flowcharts

### Main Flowchart

Diagram

Description automatically generated

## Hierarchy Charts

These hierarchy charts are shown for some classes. They depict how functions are used in conjunction with one another, and which functions are called by each other.

### Main Menu

### Map Design

## Data Flow Diagrams

### Level 0 DFD

Diagram

Description automatically generated

### Level 1 DFD

Diagram

Description automatically generated

## Description of main algorithms

### Map Design

The user has the option to:

1. Pick a start tile (required)
2. Undo an action
3. Clear all
4. Save map

**Tile Types**

To design my python algorithm for the map design, I needed to create two classes: a userButtons class for the tiles and a main map creation class. In the first class, the tile colours and types were saved.

Text

Description automatically generated

The pseudocode above shows an example of how the tiles are assigned. For example, the track tile, represented by ‘T,’ has the colour which is stored in trackColour. This is replicated for all three tile types: track, start and wall.

**Creating a map**

Before starting the map creation, we must prompt the user to enter the number of rows and columns they want to create. We can then use our userButtons class to change tiles to walls or tracks.

Text

Description automatically generated

By using a nested for loop, we can successfully create the map with our desired size.

**Saving the map**

Perhaps the most important function between both classes is the map saving function. This falls under the MapCreation class, which uses Tkinter in order to display the map and allow the user to interact.

Text

Description automatically generated

Algorithm to create RLE string

Gives each tile an ID

**User interface design**

*Figure 2*

Graphical user interface, text, application

Description automatically generated

Figure 2 shows what the menu will look like for the user to enter the number of rows and columns. The design is pulled from Tkinter, and a green start button is used to make it more obvious how to progress with the simulation for usability.

*Figure 3*

Chart

Description automatically generated

Figure 3 shows what the map design interface looks like before any tiles are added. It has four options as described on page 13.

*Figure 4*

Red = walls, blue = track

Chart

Description automatically generated

Figure 4 shows what the interface looks like when the user has selected tiles to be part of the track. Note that a start tile has not been selected yet.

*Figure 5*

A picture containing text, electronics, screenshot, display

Description automatically generated

Figure 5 shows a start tile. Once a start file has been selected, the map should be ready to be saved.

*Figure 6*

RLE used here

Text

Description automatically generated with medium confidence

Figure 6 shows what the pickle file looks like for the map. It is clear that there is run length encoding used in order to store the positions of tiles and walls.

### Neural network algorithm

This algorithm focuses on the creation of the Feed Forward Neural Network

**Creating the network size**

Firstly, the size of the neural network needs to be set.

Text

Description automatically generated

This means that the input layer will have four inputs, and there will be two hiddel layers followed by an output layer which has two outputs.

**Randomising weights**

We then need to randomise our weights.

Text

Description automatically generated

This gives a random number for each weight.

**Traversing through the network**

Text

Description automatically generated

Applies sigmoid function here

Uses matrix multiplication on the weights

This uses the sigmoid function (Figure 6) to get our outputs.

*Figure 7*

Transformed version of hyperbolic tan function

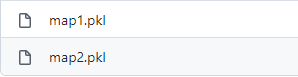
The function gives us values between 0 and 1, which is useful for the neural network as we can set numerical conditions to change the angle or speed of the car easily.

### File Organisation

*Figure 8*



*Figure 9*



*Figure 10*



Figure 8 above shows two folders which are created in main.py. They store the maps and weights in pickle files (.pkl) so that they can be retrieved in the menu shows in figure 10. These maps and weights are retrieved using an algorithm, and it should show the files to the user, as seen in Figure 9.



This subroutine allows the user to open the weights file and retrieve the pickle file as intended. It uses the module ‘os’ in order to do this, but I replaced the command with ‘GETDIRECTORY’ as we are using pseudocode here.

### Radars

Each car will have 5 green beams extended from it. They will get updated every frame, rotating and calculating the distance to walls based on the car itself.

**Updating beams**

Text

Description automatically generated

This subroutine takes the starting vector, rotation and lines as parameters. It calculates the anhgle change by adding the angle to the rotation, then applying the MOD function with 360 in order to ensure that the angle is between 0 and 360.

**Showing beams**

Text

Description automatically generated

This simple subroutine draws the beams onto the pygame screen.

### Generating map

In my simulation class, there will be a subroutine to generate the map. When the map is saved, the points are saved as starting ID, wall or track to make it easier to decompress.

Text

Description automatically generated

Although this subroutine references the use of the Track and Wall classes, they are self explanatory so I decided not to detail them here.

Both classes are child classes inheriting the parent class ‘Tile’.

Text

Description automatically generated

This class has attributes for the tile size and tile ID, which are important for generating the map.

# Technical Solution

## Table Of Skills

|  |  |
| --- | --- |
| **Skill** | **Page reference** |
| Simple OOP models | *main.py (pg 22)* |
| Stack operations | *MapDesign.py (pg 28)*  *Stack.py (pg 30)* |
| Reading/writing to and from files | *Main.py ( pg 24-25)*  *MapDesign.py (pg 29)* |
| Binary search | *borderLineGenerator.py (pg 48)* |
| Complex mathematical calculations, including finding line intersects, gradients, y intercepts, applying functions to inputs | *CT.py (pg 35-36)*  *neural.py (pg 37-38)* |
| Advanced matrix operations | *neural.py (pg 37-38)* |
| Recursive operations | *borderLineGenerator.py (pg 48-50)* |
| Merge sort | *borderLineGenerator.py (pg 48-50)* |
| Complex OOP models | *simulation.py (pg 42-46)* |
| Run length encoding | *MapDesign.py (pg 29)* |
| Neural network algorithm | *neural.py (pg 37-38)* |
| Complex user algorithms | *simulation.py (pg 42-46)* –  generateMap (method)  animationLoop (method)  *population.py (pg 39-41)* –  mutate (method)  createNextGeneration (method)  *Car.py (pg 33-34)* –  update (method) |
| Exception handling |  |

## Structure of program

A screenshot of a computer

Description automatically generated with medium confidenceClasses:

* Radar class in radar.py
* borderLineGenerator class in borderlinegenerator.py
* Car class in car.py
* Coordinate toolkit class in CT.py
* MapCreator and UserButtons classes in MapDesign.py
* Population class in population.py
* Simulation class in simulation.py
* Stack class in Stack.py

## Code

### main.py

#Program source code by Cathline Sean Dougan

#Project start 23/07/21

#Project end TBA

#Autonomous car simulation using neural networks and pygame

import sys

import os #to find files

import tkinter as tk #for display

from tkinter import messagebox

import tkinter.font as tkFont

from tkinter.filedialog import askopenfilename

import pickle #to save and load map

from typing import Type #for saving map

import MapDesign

import simulation as sim

import threading #Trying to solve tkinter 'Not Responding' problem

#Early August update: 03/08/2021

#Design menu created. Imports imported.

#Quit function completed

#Map Design function started

EMPTY\_STRING = ''

class Application(tk.Frame):

    def \_\_init\_\_(self, master = None):

        tk.Frame.\_\_init\_\_(self, master)

        self.master = master

        self.pack()

        self.MapDict = None

        self.loadedWeights = None

        self.fontStyle = tkFont.Font(family="Lucida Grande", size=20)

        self.displayMenu()

        global EMPTY\_STRING

    def displayMenu(self):

        menuTitle = tk.Label(self, text = "Welcome to Sean's Autonomous Car Simulation",

            foreground = "black",

            background = "white",

            font = self.fontStyle

        ).pack(side = 'top')

        self.designMap = tk.Button(self, text = 'DESIGN A MAP', font = self.fontStyle, bg = 'black', fg = 'white', command = self.getMapinfo).pack(side = 'top')

        self.runSimulation = tk.Button(self, text = 'RUN SIMULATION', font = self.fontStyle, bg = 'black', fg = 'white', command = self.getSiminfo).pack(side = 'top')

        self.obstacles = tk.Button(self, text = 'ADD/REMOVE OBSTACLES', font = self.fontStyle, bg = 'black', fg = 'white', command = self.getObstacles).pack(side = 'top')

        self.quit = tk.Button(self, text = 'QUIT', font = self.fontStyle, bg = 'black', fg = 'white', command = self.master.destroy).pack(side = 'bottom')

    def getMapinfo(self):

        sizeWindow = tk.Tk()

        sizeWindow.title("Grid dimensions")

        rowMessage = tk.Label(sizeWindow, text = 'Enter the number of rows')

        self.rowInput = tk.Entry(sizeWindow, bd = 5)

        columnMessage = tk.Label(sizeWindow, text = 'Enter the number of columns')

        self.columnInput = tk.Entry(sizeWindow, bd = 5)

        allez = tk.Button(sizeWindow, text = 'START', bg = 'green', command = self.createMap)

        rowMessage.grid(row = 0, column = 0)

        self.rowInput.grid(row = 0, column = 1) #ctrl C

        columnMessage.grid(row = 1, column = 0)

        self.columnInput.grid(row = 1, column = 1)

        allez.grid(row = 2, column = 1)

        sizeWindow.mainloop()

    def createMap(self):

        root = tk.Tk()

        try:

            MapDesign.MapCreation(int(self.rowInput.get()),int(self.columnInput.get()), master = root) #Grabs values and creates a window

        except ValueError:

            root.withdraw() #Doesn't show the window

            messagebox.showinfo("Entry error", "Please retry entering values") #messagebox shows message for error

    def getSiminfo(self):

        menuWindow = tk.Tk()

        showMutation  = tk.Label(menuWindow, text = 'mutation level = ').pack(side = 'top')

        self.mutationEntry = tk.Entry(menuWindow, bd = 5)

        self.mutationEntry.pack(side = 'top')

        showCarNumber = tk.Label(menuWindow, text = 'cars per generation = ').pack(side = 'top')

        self.carEntry = tk.Entry(menuWindow, bd = 5)

        self.carEntry.pack(side = 'top')

        selectMap = tk.Button(menuWindow, text = 'SELECT MAP', font = self.fontStyle, bg = 'black', fg = 'white', command = self.getMapfile).pack(side = 'bottom')

        selectWeights = tk.Button(menuWindow, text = 'SELECT WEIGHTS FILE', font = self.fontStyle, bg = 'black', fg = 'white', command = self.loadWeights).pack(side = 'bottom')

        startButton = tk.Button(menuWindow, text = 'START', command = self.runSim).pack(side = 'bottom')

    def getObstacles(self):

        pass

    def runSim(self):

        mutation = self.mutationEntry.get()

        carNum = self.carEntry.get()

        if self.MapDict is not None: #Ensures a map is selected

            try:

                if mutation == "":

                    raise TypeError

                mutation = int(mutation)

                #print(mutation)

                if mutation < 0 or mutation > 100:

                    raise ValueError

                else:

                    newSimulation = sim.Simulation(self.MapDict, mutation, self.loadedWeights, carNum)

            except ValueError:

                messagebox.showinfo("ValueError", "Mutation entry invalid.")

                self.mutationEntry.delete(0,len(self.mutationEntry.get())) #Clears entry box

            except TypeError:

                messagebox.showinfo("","Invalid mutation level, Enter again")

            try:

                if carNum == "":

                    raise TypeError

                carNum = int(carNum)

                newSimulation = sim.Simulation(self.MapDict, mutation, self.loadedWeights, carNum)

            except TypeError:

                messagebox.showinfo("", "Enter number of cars per generation again")

        else:

            messagebox.showinfo("Map not selected","Please select a map before continuing!")

    def loadWeights(self):

        file = askopenfilename(initialdir= os.getcwd() + "\\weights", filetypes=(("PKL File", "\*.pkl"),("All Files", "\*.\*")), title = "Choose file")

        try:

            with open(file, 'rb') as pkl\_file:

                self.loadWeights = pickle.load(pkl\_file)

        except:

            messagebox.showinfo("", "File not retrieved, try again")

    def getMapfile(self):

        file = askopenfilename(filetypes=(("PKL Files", "\*.pkl"),))

        try:

            with open(file, 'rb') as pkl\_file:

                self.MapDict = pickle.load(pkl\_file)

        except:

            messagebox.showinfo("Error", "File not found, try again")

if \_\_name\_\_ == "\_\_main\_\_":

    cwd = os.getcwd()

    try:

        os.makedirs(cwd + '/weights') #makes a weights folder

    except:

        pass

    try:

        os.makedirs(cwd + '/maps') #makes a map folder

    except:

        pass

    root = tk.Tk()

    app = Application(master = root)

    app.mainloop()

### MapDesign.py

import tkinter as tk

import tkinter.font as tkFont

from tkinter.filedialog import asksaveasfile

from tkinter import messagebox

import pickle

import os

from Stack import Stack

class userButtons:

    trackColour = (0,255,255)

    wallColour = (255,0,0)

    buttonCount = 0

    startup = False

    currentStack = None

    def \_\_init\_\_(self, r, c, win):

        self.Button = tk.Button(win, bg = userButtons.fromRGB(userButtons.wallColour), command = self.changeColour, width = 4, height = 2)

        self.Button.grid(row=r,column=c)

        self.buttonType = 'W'

        self.id = userButtons.buttonCount

        userButtons.buttonCount += 1

    def changeColour(self):

        userButtons.currentStack.push([self.buttonType + str(self.id)]) #This pushes the info onto the stack (LiFo)

        if userButtons.startup:

            self.pickStart()

            userButtons.startup = False

        else:

            if self.buttonType == 'W':

                self.makeTrack()

            elif self.buttonType == 'T' or self.buttonType == 'S':

                self.makeWall()

    def makeTrack(self):

        self.buttonType = 'T'

        self.Button.config(text='', bg=userButtons.fromRGB(userButtons.trackColour))

    def pickStart(self):

        self.buttonType = 'S'

        self.Button.config(text="S", bg=userButtons.fromRGB(userButtons.trackColour))

    def makeWall(self):

        self.buttonType = 'W'

        self.Button.config(text="", bg=userButtons.fromRGB(userButtons.wallColour))

    def fromRGB(rgb):

        return "#%02x%02x%02x" % rgb

class MapCreation(tk.Frame):

    def \_\_init\_\_(self, rows, columns, master = None):

        tk.Frame.\_\_init\_\_(self, master)

        self.master = master

        self.master.title("MAP CREATOR")

        self.pack()

        self.buttons = []

        self.rows = rows

        self.columns = columns

        self.createWidgets()

    def createWidgets(self):

        mapFrame = tk.Frame(self.master)

        mapFrame.pack(side = 'top')

        #Frame for the whole map

        optionsFrame = tk.Frame(self.master)

        optionsFrame.pack(side = 'bottom')

        #Creates a frame ready for options to be added

        userButtons.currentStack = Stack(50) #Allows last 5 changes to be undone

        for r in range(self.rows):

            for c in range(self.columns):

                self.buttons.append(userButtons(r,c,mapFrame))

        #creates the map creator in the correct size

        startButton = tk.Button(optionsFrame, text = 'PICK START TILE', command = self.startButton).grid(row = 0, column = 0)

        undoButton = tk.Button(optionsFrame, text = 'UNDO', command = self.undo).grid(row = 0, column = 1)

        clearButton = tk.Button(optionsFrame, text = 'CLEAR ALL', command = self.clearMap).grid(row = 1 , column = 0)

        saveButton = tk.Button(optionsFrame, text = 'SAVE', command = self.saveMap).grid(row = 1 , column = 1)

    def startButton(self):

        buttonDeya = False

        for i in self.buttons:

            if i.buttonType == 'S':

                messagebox.showinfo("","Try removing the current start button before adding a new one")

                buttonDeya = True

        if not buttonDeya:

            messagebox.showinfo("","Click on a tile to make it the new starting tile")

            userButtons.startup = True

    def undo(self):

        try:

            returnVal = userButtons.currentStack.pop()

            for x in returnVal:

                tileType = x[0] #This will return the button type

                tileID = int(x[1:]) #This shows the ID

                if tileType == 'W':

                    self.buttons[tileID].makeWall()

                elif tileType == 'T':

                    self.buttons[tileID].makeTrack()

                elif tileType == 'W':

                    self.buttons[tileID].pickStart()

        except TypeError:

            messagebox.showinfo("Cannot undo","You can't undo anymore! ")

    def clearMap(self):

        status = []

        for i in self.buttons:

            status.append(i.buttonType + str(i.id))

            i.buttonType = 'W'

            i.Button.config(text = '', bg = userButtons.fromRGB(userButtons.wallColour))

        userButtons.currentStack.push(status)

    def saveMap(self):

        startingID = -1

        trackCount = 0

        for i in self.buttons:

            if i.buttonType == 'T':

                trackCount += 1

            elif i.buttonType == 'S':

                startingID = trackCount

                break

        if startingID == -1:

            messagebox.showinfo("","Add a start track before you save!")

        else:

            file = asksaveasfile(initialdir= os.getcwd() + '\\maps', mode='wb', defaultextension=".pkl")

            if file is None:

                return

            mapRLE =''

            counter = 0

            currentType = self.buttons[0].buttonType

            for i in self.buttons:

                if i.buttonType == currentType:

                    counter += 1

                else:

                    mapRLE += str(counter) + currentType

                    if currentType == 'W':

                        currentType = 'T'

                    else:

                        currentType = 'W'

                    counter = 1

            mapRLE += str(counter) + currentType

            mapDictionary = {'rows':self.rows, 'columns':self.columns, 'data':mapRLE, 'startID':startingID}

            pickle.dump(mapDictionary, file) #saves file

### Stack.py

#Technique: stack operations

class Stack:

    #Just a typical Stack module. This is needed for map creation so the user can have an 'undo' option.

    def \_\_init\_\_(self, maxSize):

        self.top = -1#points to top of stack

        self.contents = []

        for i in range(maxSize):#initialise with placeholders

            self.contents.append(0)

        self.maxSize = maxSize

    def push(self, item):

        if self.isFull():

            temp = self.contents

            for i in range(len(self.contents) - 1):

                self.contents[i] = temp[i + 1]

                self.contents[self.top] = item

        else:

            self.top += 1

            self.contents[self.top] = item

    def pop(self):

        if self.isEmpty():

            pass

        else:

            item = self.contents[self.top]

            self.top -= 1

            return item

    def isEmpty(self):

        if self.top == -1:

            return True

        else:

            return False

    def isFull(self):

        if self.top == self.maxSize - 1:

            return True

        else:

            return False

### radar.py

#This program is used to create the radar beams

import pygame

from pygame.math import Vector2

import math

from CT import CT #imports coordinate toolkit from previous python file

class radars:

    colour = (0, 255, 0) # green beams (by choice)

    def \_\_init\_\_(self, start, angle):

        self.start = start #returns starting position

        self.length = 5

        self.end = (start[0] + 3, start[0] + 4) #so the end starts with a length

        self.angle = angle

        self.startPoint = pygame.math.Vector2(start[0], start[1]) #creates vector for start point

        self.endPoint = pygame.maths.Vector2(start[0] + 50, start[1])

    def update(self, startVector, rotation, lines):

        self.startPoint = startVector

        if rotation > 0:

            self.angle = (self.angle + rotation) % 360 #adds rotation to angle, then mods with 360 so it's between 0 and 360

        else:

            self.angle = (self.angle + rotation) % -360 #if rotation is negative it needs to mod with -360 for the same reason

        self.rotateLine()

        self.calculateBorderIntersection(lines)

        self.length = self.startPoint.distance\_to(self.current) #distance\_to (from pygame) calculates distance

    def show(self, SCR):

        try:

            pygame.draw.aaline(SCR, radars.colour, self.startPoint, self.current, 1) #blits radars onto screen

        except:

            print('Must update then show beam to initialise length')

    def rotateLine(self):

        #rotates radars by specified angle

        self.current = self.startPoint + self.endPoint.rotate(self.angle)

    def calculateBorderIntersection(self, lines):

        horizontalLines = lines[0] #x values on cartesian plane

        verticalLines = lines[1] #y values on cartesian plane

        smallestDistance = 1

        smallestIntersect = None

        for line in horizontalLines:

            intersect = CT.getIntersectBetweenLineSegments((self.startPoint, self.current), ((line[0], line[2]), (line[1], line[2]))) #finds point of intersection using CT class

            if intersect is not None:

                distance = self.startPoint.distance\_to(pygame.math.Vector2(intersect[0], intersect[1])) #finds distance between the intersect and the start point

                if distance < smallestDistance:

                    smallestDistance = distance

                    smallestIntersect = intersect

        for line in verticalLines:

            #repeats the same thing but for vertical lines

            intersect = CT.getIntersectBetweenLineSegments((self.startPoint, self.current), ((line[2], line[0]), (line[2], line[1])))

            if intersect is not None:

                distance = self.startPoint.distance\_to(pygame.math.Vector2(intersect[0], intersect[1]))

                if distance < smallestDistance:

                    smallestDistance = distance

                    smallestIntersect = intersect

        try:

            self.current = pygame.math.Vector2(smallestIntersect[0], smallestIntersect[1]) #Vector2 function defines a vector for the intersect (direction and magnitude)

        except:

            pass

        #if there is no smallest intersect it will pass rather than stop the program.

### Car.py

#For arrow car model

from numpy.lib.function\_base import angle

from neural import Neural

import pygame

from pygame.math import Vector2

from radar import radars

class Car:

    #static class variables here

    beamAngles = [-45, 0, 45] #angles for the car beams

    \_\_SIZE = None

    beamCarOffset = 10

    idCounter = 0

    def \_\_init\_\_(self, frontP):

        self.id = Car.idCounter

        Car.idCounter += 1

        self.brain = Neural() # Instantiates neural network to car

        self.framesAlive = 0

        self.fitness = 0

        self.dead = False #Boolean to check whether car is dead

        self.bestOfPrevGen = False #Will be True if car selected has the highest fitness

        self.collidedCheckPoints = []

        self.sprite = pygame.image.load(r"car.png").convert()

        self.sprite = pygame.transform.scale(self.sprite, (Car.\_\_SIZE, Car.\_\_SIZE))

        self.vel = Vector2(1,0)

        self.nextPoint = frontP + self.vel

        self.frontPoint = self.nextPoint#make front of triangle/car

        self.leftPoint = self.frontPoint + Vector2(-Car.\_\_SIZE,0).rotate(30)

        self.rightPoint = self.frontPoint + Vector2(-Car.\_\_SIZE,0).rotate(-30)

        self.carCenter = ((self.frontPoint+self.leftPoint+self.rightPoint)/3)

        self.angle = 0

        #create beams

        self.beams = []

        for a in Car.beamAngles:

            beamOrigin = self.frontPoint - Vector2(Car.beamCarOffset, 0).rotate(self.angle)

            self.beams.append(radars(beamOrigin, a))

        self.edges = []

    def update(self, borderLines):

        self.framesAlive += 1

        brainInput = [b.length/50 for b in self.beams]

        brainInput.append(self.vel.length() / (Car.\_\_SIZE/2))

        (angleChange, acceleration) = self.brain.calculateOutput(brainInput) #Uses FF Neural Network to calculate the change in angle and acceleration

        acceleration += 1

        if (Vector2(self.vel)).length() >= Car.\_\_SIZE/2 and acceleration > 1:

            acceleration = 0

        self.frontPoint = self.nextPoint

        if angleChange > 0:

            self.angle = (self.angle + angleChange) % 360 #Mods with 360 so the answer is between  0 and 360

        else:

            self.angle = (self.angle + angleChange) % -360

        self.leftPoint = self.frontPoint + Vector2(-Car.\_\_SIZE, 0).rotate(30 + self.angle)

        self.rightPoint = self.frontPoint + Vector2(-Car.\_\_SIZE, 0).rotate(-30 + self.angle)

        self.carCenter = ((self.frontPoint+self.leftPoint+self.rightPoint)/3)

        self.nextPoint = self.frontPoint + Vector2(self.vel \* acceleration).rotate(self.angle)

        self.edges = [[self.leftPoint, self.frontPoint], [self.frontPoint, self.rightPoint], [self.rightPoint, self.leftPoint]]

        for b in self.beams:

            beamOrigin = self.frontPoint - Vector2(Car.beamCarOffset, 0).rotate(self.angle)

            b.update(beamOrigin, angleChange, borderLines)

    #draws car

    def show(self, screen):

        carCol = (255, 0, 0)

        if self.bestOfPrevGen:

            carCol = (0, 255, 0)

        pygame.draw.polygon(screen, carCol, (self.leftPoint, self.frontPoint, self.rightPoint, (self.leftPoint+self.rightPoint+self.frontPoint)/3))

        #screen.blit(self.sprite, self.carCenter)

        pygame.draw.circle(screen, (255,255,255), (int(self.frontPoint[0]), int(self.frontPoint[1])), 2)

        pygame.draw.aaline(screen, (0, 0, 100), self.frontPoint, self.nextPoint, 1)

        for b in self.beams:

            b.show(screen)

    def getSize():

        return Car.\_\_SIZE

    def setSize(size):

        Car.\_\_SIZE = size

### newcar.py

#For car sprite

import math

from numpy.lib.function\_base import angle

from neural import Neural

import pygame

from pygame.math import Vector2

from radar import radars

import math

class Car:

    #static class variables here

    beamAngles = [-45, 0, 45] #angles for the car beams

    \_\_SIZE = None

    beamCarOffset = 10

    idCounter = 0

    def \_\_init\_\_(self, frontP):

        self.id = Car.idCounter

        Car.idCounter += 1

        self.brain = Neural() # Instantiates neural network to car

        self.framesAlive = 0

        self.fitness = 0

        self.dead = False #Boolean to check whether car is dead

        self.bestOfPrevGen = False #Will be True if car selected has the highest fitness

        self.collidedCheckPoints = []

        self.sprite = pygame.image.load(r"car.png").convert()

        self.sprite = pygame.transform.scale(self.sprite, (Car.\_\_SIZE, Car.\_\_SIZE))

        self.vel = Vector2(1,0)

        self.nextPoint = frontP + self.vel

        self.frontPoint = self.nextPoint#make front of triangle/car

        self.leftPoint = self.frontPoint + Vector2(-Car.\_\_SIZE,0).rotate(30)

        self.rightPoint = self.frontPoint + Vector2(-Car.\_\_SIZE,0).rotate(-30)

        self.carCenter = (((self.frontPoint[0]+self.leftPoint[0]+self.rightPoint[0])/3),((self.frontPoint[1]+self.leftPoint[1]+self.rightPoint[1])/3))

        self.angle = 0

        #create beams

        self.beams = []

        for a in Car.beamAngles:

            beamOrigin = self.carCenter - Vector2(Car.beamCarOffset, 0).rotate(self.angle)

            self.beams.append(radars(beamOrigin, a))

        self.edges = []

    def update(self, borderLines):

        self.framesAlive += 1

        brainInput = [b.length/50 for b in self.beams]

        brainInput.append(self.vel.length() / (Car.\_\_SIZE/2))

        (angleChange, acceleration) = self.brain.calculateOutput(brainInput) #Uses FF Neural Network to calculate the change in angle and acceleration

        acceleration += 1

        if (Vector2(self.vel)).length() >= Car.\_\_SIZE/2 and acceleration > 1:

            acceleration = 0

        self.frontPoint = self.nextPoint

        if angleChange > 0:

            self.angle = (self.angle + angleChange) % 360 #Mods with 360 so the answer is between  0 and 360

        else:

            self.angle = (self.angle + angleChange) % -360

        self.leftPoint = self.frontPoint + Vector2(-Car.\_\_SIZE, 0).rotate(30 + self.angle)

        self.rightPoint = self.frontPoint + Vector2(-Car.\_\_SIZE, 0).rotate(-30 + self.angle)

        #self.sprite = self.rotate\_center(self.sprite, self.angle % 360)

        self.carCenter = ((self.frontPoint+self.leftPoint+self.rightPoint)/3)

        self.nextPoint = self.frontPoint + Vector2(self.vel \* acceleration).rotate(self.angle)

        self.sprite = self.rotate\_center(self.sprite, self.angle % 360)

        '''length = 0.5 \* Car.\_\_SIZE

        left\_top = [self.carCenter[0] + math.cos(math.radians(360 - (self.angle + 30))) \* length, self.carCenter[1] + math.sin(math.radians(360 - (self.angle + 30))) \* length]

        right\_top = [self.carCenter[0] + math.cos(math.radians(360 - (self.angle + 150))) \* length, self.carCenter[1] + math.sin(math.radians(360 - (self.angle + 150))) \* length]

        left\_bottom = [self.carCenter[0] + math.cos(math.radians(360 - (self.angle + 210))) \* length, self.carCenter[1] + math.sin(math.radians(360 - (self.angle + 210))) \* length]

        right\_bottom = [self.carCenter[0] + math.cos(math.radians(360 - (self.angle + 330))) \* length, self.carCenter[1] + math.sin(math.radians(360 - (self.angle + 330))) \* length]

        self.edges = [left\_top, right\_top, left\_bottom, right\_bottom]'''

        self.edges = [[self.leftPoint, self.frontPoint], [self.frontPoint, self.rightPoint], [self.rightPoint, self.leftPoint]]

        for b in self.beams:

            beamOrigin = self.frontPoint - Vector2(Car.beamCarOffset, 0).rotate(self.angle)

            b.update(beamOrigin, angleChange, borderLines)

    #draws car

    def show(self, screen):

        carCol = (255, 0, 0)

        if self.bestOfPrevGen:

            carCol = (0, 255, 0)

        #pygame.draw.polygon(screen, carCol, (self.leftPoint, self.frontPoint, self.rightPoint))

        #pygame.draw.polygon(screen, carCol, (self.frontPoint, ((self.rightPoint+self.leftPoint)/2)), 15)

        screen.blit(self.sprite, self.carCenter)

        pygame.draw.circle(screen, (255,255,255), (int(self.frontPoint[0]), int(self.frontPoint[1])), 2)

        pygame.draw.aaline(screen, (0, 0, 100), self.frontPoint, self.nextPoint, 1)

        for b in self.beams:

            b.show(screen)

    def rotate\_center(self, image, angle):

        # Rotate The Rectangle

        rectangle = image.get\_rect()

        rotated\_image = pygame.transform.rotate(image, angle/60)

        rotated\_rectangle = rectangle.copy()

        rotated\_rectangle.center = rotated\_image.get\_rect().center

        rotated\_image = rotated\_image.subsurface(rotated\_rectangle).copy()

        return rotated\_image

    def getSize():

        return Car.\_\_SIZE

    def setSize(size):

        Car.\_\_SIZE = size

### CT.py

#Coordinate Toolkit used to navigate cars

#Uses a range of mathematical calculations including calculating gradients, equations of lines, points of intersections between lines

class CT:

    def calcGradient(start, end): #Uses start and end of line (both Cartestian) as parameters to calculate gradient

        if start[0] == end[0]: #if line is vertical (x's are same) it will not have gradient (as it's infinity)

            return None

        else:

            m = (end[1] - start[1])/(end[0] - start[0]) #(y2-y1)/(x2-x1) to calculate the gradiant 'm'

            return m #returns m as the gradient

    def calcYintercept(point, m): #uses a point on the line to calculate the y intercept

        return point[1] - (point[0] \* m) #rearraged y-y1=m(x-x1) formula

    def getIntersect(s1,e1,s2,e2):

        m1 = CT.calcGradient((s1),(e1)) #gets gradient for first line

        m2 = CT.calcGradient((s2),(e2)) #gets gradient for second line

        if m1 != m2: #if not parallel

            if m1 is not None and m2 is not None:#if neither are vertical

                c1 = CT.calcYintercept(s1, m1) #calculates y intercept of both lines

                c2 = CT.calcYintercept(s2, m2)

                x = (c2 - c1) / (m1 - m2) #get x point of intersection

                y = (m1 \* x) + c1 #for y plug in x into either

            elif m2 is None:#if line 2 vertical (but not line 1)

                c1 = CT.calcYintercept(s1, m1)

                x = s2[0]#line 1 cuts 2 at 2's x which is contstant as it's vertical line

                #plug in for y

                y = (m1 \* x) + c1

            elif m1 is None:#if line 1 vertical

                c2 = CT.calcYintercept(s2, m2)

                x = s1[0]

                y = (m2 \* x) + c2

        else: #if parallel they don't intersect

            return None

        return ((x,y))#

    def getIntersectBetweenLineSegments(line1, line2):

        ver = False

        hor = False

        if line2[0][0] == line2[1][0]:

            ver = True

        elif line2[0][1] == line2[1][1]:

            hor = True

        intersect = CT.getIntersect(line1[0], line1[1], line2[0], line2[1])

        if intersect is not None:

            if hor == True:

                if intersect[0] >= line2[0][0]and intersect[0] <= line2[1][0]:

                    if (line1[0][1] <= intersect[1] and intersect[1] <= line1[1][1]) or (line1[1][1] <= intersect[1] and intersect[1] <= line1[0][1]):

                        return intersect

            elif ver == True:

                if intersect[1] >= line2[0][1] and intersect[1] <= line2[1][1]:

                    if (line1[0][0] <= intersect[0] and intersect[0] <= line1[1][0]) or (line1[1][0] <= intersect[0] and intersect[0] <= line1[0][0]):

                        return intersect

            else:

                print('line2 must be a horizontal or vertical line, make sure it was passed in correctly')

        return None

### neural.py

import sys

import numpy as np

import matplotlib.pyplot as plt #I only added this so I could see what the sigmoid function looked like

from numpy.ma.core import outerproduct

class Neural:

    def \_\_init\_\_(self):

        #determine sizes for feed forward nn

        self.inputLayerSize = 4

        self.hiddenLayer1Size = 5

        self.hiddenLayer2Size = 5

        self.outputLayerSize = 2

        self.weights1 = np.random.randn(self.inputLayerSize, self.hiddenLayer1Size)

        self.weights2 = np.random.randn(self.hiddenLayer1Size, self.hiddenLayer2Size)

        self.weights3 = np.random.randn(self.hiddenLayer2Size, self.outputLayerSize)

        #randomises weights

    def calculateOutput(self, inputs):

        e = np.array([inputs]) #puts inputs into an array

        output = self.forward(e)[0] #determines outputs

        angle = output[0]

        if angle > 0.6:

            angleChange = -2

        elif angle < 0.4:

            angleChange = 2

        else:

            angleChange = 0

        #changes angle based on the neural network's output for the first output neuron

        acceleration = output[1]

        #acceleration changes based on the second output neuron

        return(angleChange, acceleration)

    def forward(self, e):

        #Traverses through neyral network using matrix multiplication to find an output

        hidden1 = np.matmul(e, self.weights1)

        activated1 = self.sigmoid(hidden1)

        hidden2 = np.matmul(hidden1, self.weights2)

        activated2 = self.sigmoid(hidden2)

        hidden3 = np.matmul(hidden2, self.weights3)

        activated3 = self.sigmoid(hidden3)

        return activated3

        #This function uses matrix multiplication and the sigmoid function (which ranges from 0 to 1) as the activation function

    def sigmoid(self, x):

        return 1/(1 + np.exp(-x)) #this is the sigmoid function (returns values between 0 and 1). it is just a rescalement and translation of the hyperbolyc tan function so it ranges from 0 to 1

### population.py

from car import Car

import random

from CT import CT

import numpy as np

class Population:

    def \_\_init\_\_(self, numCars, frontPoint, carSize, mutationLevel):

        Car.setSize(carSize) #sets size of car

        self.bestCarFitness = 0 #initialises variable for car fitness (so that it can be used for future generations)

        self.dead = False #Boolean for whether the car is dead or not

        self.startPoint = frontPoint

        self.mutationLevel = mutationLevel

        self.numCars = int(numCars)

        self.generation = 1

        #print(self.mutationLevel)

        self.cars = []

        for \_ in range(int(numCars)):

            self.cars.append(Car(frontPoint))

    def createDistribution(self):

        distribution = []

        for c in self.cars:

            for f in range(0,c.fitness):

                distribution.append(c.id)

        return distribution

    def crossover(weights1, weights2):

        newWeights = np.zeros((weights1.shape[0], weights1.shape[1]))

        for row in range(len(newWeights)):

            for weight in range(len(newWeights[0])):

                ranNum = random.randint(0,1)

                if ranNum == 0:

                    newWeights[row][weight] = weights1[row][weight]

                else:

                    newWeights[row][weight] = weights2[row][weight]

        return newWeights

    def mutate(self, car):

        for i in range(len(car.brain.weights1)):

            for j in range(len(car.brain.weights1[0])):

                randNum = random.randint(0, 100)

                if randNum < self.mutationLevel:

                    car.brain.weights1[i][j] = np.random.randn()

        for i in range(len(car.brain.weights2)):

            for j in range(len(car.brain.weights2[0])):

                randNum = random.randint(0, 100)

                if randNum < self.mutationLevel:

                    car.brain.weights2[i][j] = np.random.randn()

        for i in range(len(car.brain.weights3)):

            for j in range(len(car.brain.weights3[0])):

                randNum = random.randint(0, 100)

                if randNum < self.mutationLevel:

                    car.brain.weights3[i][j] = np.random.randn()

    def createNextGeneration(self):

        probDistribution = []

        bestCar = self.cars[0]

        for c in self.cars:

            if c.fitness > bestCar.fitness:

                bestCar = c

        self.generation += 1

        if bestCar.fitness > self.bestCarFitness:

            self.bestCarFitness = bestCar.fitness

        newCars = []

        Car.idCounter = 0

        for i in range(self.numCars):

            newCars.append(Car(self.startPoint))

        try:

            probDistribution = self.createDistribution()

        except:

            for c in self.cars:

                for i in range (0, self.cars.fitness):

                    probDistribution.append(i)

        #print(probDistribution)

        for c in newCars:

            try:

                parent1 = self.cars[probDistribution[random.randint(0, len(probDistribution) - 1)]]

            except:

                pass

            try:

                parent2 = self.cars[probDistribution[random.randint(0, len(probDistribution) - 1)]]

            except:

                pass

            c.brain.weights1 = Population.crossover(parent1.brain.weights1, parent2.brain.weights1)

            c.brain.weights2 = Population.crossover(parent1.brain.weights2, parent2.brain.weights2)

            c.brain.weights3 = Population.crossover(parent1.brain.weights3, parent2.brain.weights3)

            self.mutate(c)

        newCars[-1].brain.weights1 = bestCar.brain.weights1

        newCars[-1].brain.weights2 = bestCar.brain.weights2

        newCars[-1].brain.weights3 = bestCar.brain.weights3

        newCars[-1].bestOfPrevGen = True

        self.cars = newCars

    def update(self, borderLines, checkPoints):

        horLines = borderLines[0]

        verLines = borderLines[1]

        for c in self.cars:

            if not c.dead:

                for e in c.edges:

                    for line in horLines:

                        if CT.getIntersectBetweenLineSegments(e, ((line[0],line[2]), (line[1],line[2])) ) is not None:

                            c.dead = True

                            c.fitness = int(((len(c.collidedCheckPoints)\*\*7) + 1)/(c.framesAlive))

                            break

                    if not c.dead:

                        for line in verLines:

                            if CT.getIntersectBetweenLineSegments(e, ((line[2],line[0]), (line[2],line[1])) ) is not None:

                                c.dead = True

                                c.fitness = int(((len(c.collidedCheckPoints)\*\*7) + 1)/(c.framesAlive))

                                break

            if not c.dead:

                for e in c.edges:

                    for cp in checkPoints:

                        if CT.getIntersectBetweenLineSegments(e, cp) is not None:

                            if cp not in c.collidedCheckPoints:

                                c.collidedCheckPoints.append(cp)

            if not c.dead:

                c.update(borderLines)

            self.checkDead()

    def checkDead(self):

        self.dead = True

        for c in self.cars:

            if c.dead == False:

                self.dead = False

                break

    def show(self, screen):

        for c in self.cars:

            if not c.dead:

                c.show(screen)

### simulation.py

#original

import pygame

import sys

from pygame.locals import \*

from pygame import \*

import tkinter

import pickle

import os

from tkinter.filedialog import asksaveasfile

import re #regular expression

from borderLineGenerator import BorderLineGenerator

from population import Population

class Tile: #Parent class

    \_\_size = 10 #Pixel size of tile

    tileID = 0

    def \_\_init\_\_(self, x ,y):

        self.x = x

        self.y = y

        self.tileID = Tile.tileID

        Tile.tileID += 1

    def getSize():

        return Tile.\_\_size

    def setSize(size):

        if Tile.\_\_size is not None:

            Tile.\_\_size = size

class Wall(Tile): #Child class which inherites the previous class' attributes and methods

    colour = (245,208,51) #sets clour to red

    def show(self, screen):

        #subroutine to display wall

        pygame.draw.rect(screen, Wall.colour, (self.x, self.y, Tile.getSize(), Tile.getSize())) #should draw the wall onto screen(the screen) using the colour

        pygame.draw.rect(screen, (255, 255, 255), (self.x, self.y, Tile.getSize(), Tile.getSize()),1)

         #updates pygame screen (not working for some reason)

class Track(Tile): #same thing but for the track (another type of tile so Tile is the parent class again

    colour = (0,0,0)

    def \_\_init\_\_(self, x, y):

        Tile.\_\_init\_\_(self, x, y)

        self.north = self.east = self.south = self.west = False

    def show(self, screen):

        pygame.draw.rect(screen, Track.colour, (self.x, self.y, Tile.getSize(), Tile.getSize()))

        pygame.draw.rect(screen, (255, 255, 255), (self.x, self.y, Tile.getSize(), Tile.getSize()),1)

        #pygame.display.update()

class statsBox:

    def \_\_init\_\_(self, x, y, w, h): #initialises class with x and y for positions and w and h for width/height

        self.x = x

        self.y = y

        self.w = w #width

        self.h = h #height

        pygame.font.init() #initialises font

        self.font = pygame.font.SysFont('Irongate', 25) #sets font

    def show(self, screen, bestFitness, generationNum, carsLeft):

        #subroutine to show the stats pane

        bestFitnessText = self.font.render("Best fitness: " + str(bestFitness), False, (0,0,0))

        generationNumtext = self.font.render("Current gen: " + str(generationNum), False, (0,0,0))

        carsLeftText = self.font.render("Cars per gen: " + str(carsLeft), False, (0,0,0)) #Test 12.2

        pygame.draw.rect(screen, (220,220,220), (self.x, self.y, self.w, self.h))

        screen.blit(bestFitnessText, (self.x + 10, self.y + 10))

        screen.blit(generationNumtext, (self.x + 10, self.y + 50))

        screen.blit(carsLeftText, (self.x + 150, self.y + 50)) #Test 12.2

        #pygame.display.update()

class Simulation:

    def \_\_init\_\_(self, MapDict, mutation, loadedWeights, carNum):

        pygame.init()

        SCREEN\_W = pygame.display.Info().current\_w

        SCREEN\_H = pygame.display.Info().current\_h

        self.carNum = carNum

        (self.W, self.H) = self.setWindowSize(MapDict, SCREEN\_W - 100, SCREEN\_H - 100)

        self.screen = pygame.display.set\_mode((self.W, self.H)) #initialises pygame display under variable screen

        self.screen.set\_alpha(0) #alpha value determines transparency

        pygame.display.set\_caption('grid') #caption for window

        self.fpsClock = pygame.time.Clock()

        self.screen.fill((0,255,0), rect = None) #screen col

        #info section

        pygame.draw.rect(self.screen, (220,220,220), (0, (self.H - Tile.getSize()), self.W, Tile.getSize())) #changed - to , (solved black screen error)

        self.statsPane = statsBox(0, (self.H - Tile.getSize()), self.W, Tile.getSize() )

        (self.walls, self.tracks) = Simulation.generateMap(MapDict)

        for w in self.walls:

            w.show(self.screen)

        for t in self.tracks:

            t.show(self.screen)

        self.lines = BorderLineGenerator(self.tracks, MapDict["rows"], MapDict["columns"], Tile.getSize())

        self.lines = self.lines.generate()

        self.CHECKPOINTS = self.calcCheckpoints()

        startTile = self.tracks[MapDict["startingID"]]

        frontX = startTile.x + Tile.getSize()\*1/3

        frontY = startTile.y + Tile.getSize()/2

        self.population = Population(int(carNum), (int(frontX), int(frontY)), int(Tile.getSize()\*1/3), mutation)

        if loadedWeights is not None:

            self.population.cars[0].brain.weights1 = loadedWeights[0]

            self.population.cars[0].brain.weights2 = loadedWeights[1]

            self.population.cars[0].brain.weights3 = loadedWeights[2]

        self.animationLoop()

    def setWindowSize(self, MapDict, devW, devH):

        tileSize = 0

        while tileSize \* ((MapDict["rows"] + 1)) < devH and tileSize \* (MapDict["columns"]) < devW:

            tileSize += 1

        tileSize -= 1

        Tile.setSize(tileSize)

        return (tileSize \* MapDict["columns"], tileSize \* (MapDict["rows"] + 1))

    def calcCheckpoints(self):

        checkPoints = []

        size = Tile.getSize()

        for t in self.tracks:

            if t.north:

                northLine = ([(t.x, t.y), (t.x + size, t.y)])

                if northLine not in checkPoints:

                    checkPoints.append(northLine)

            if t.east:

                eastLine = ([(t.x + size, t.y), (t.x + size, t.y + size)])

                if eastLine not in checkPoints:

                    checkPoints.append(eastLine)

            if t.south:

                southLine = ([(t.x, t.y + size), (t.x + size, t.y + size)])

                if southLine not in checkPoints:

                    checkPoints.append(southLine)

            if t.west:

                westLine = ([(t.x, t.y), (t.x, t.y + size)])

                if westLine not in checkPoints:

                    checkPoints.append(westLine)

        return checkPoints

    def generateMap(MapDict):

        rows = MapDict["rows"]

        columns = MapDict["columns"]

        mapRLE = MapDict["data"]

        mapRLEarray = re.split(r'(\d+)', mapRLE)[1:]

        mapRLE2Darray = []

        for i in range(0, len(mapRLEarray), 2):

            mapRLE2Darray.append([mapRLEarray[i], mapRLEarray[i+1]])

        #create tiles

        walls = []

        tracks = []

        x = 0

        y = 0

        for group in mapRLE2Darray:

            for n in range(int(group[0])):

                if x > Tile.getSize()\*(columns - 1):

                    x = 0

                    y += Tile.getSize()

                if group[1] == 'W':

                    walls.append(Wall(x,y))

                elif group[1] == 'T':

                    tracks.append(Track(x,y))

                x += Tile.getSize()

        return (walls, tracks)

    def saveWeights(self, weights):

        root = tkinter.Tk()

        root.withdraw()

        file = asksaveasfile(initialdir= os.getcwd() + "\\weights", mode = 'wb', defaultextension=".pkl")

        if file is None:

            return

        pickle.dump(weights, file)

    def animationLoop(self):

        self.statsPane.show(self.screen, "", 1, self.carNum) #shows it's the first generation, no best fitness yet ++ Test 12.2

        while True:

            for event in pygame.event.get():

                if event.type == QUIT:

                    pygame.quit()

                    sys.exit()

            if event.type == pygame.KEYDOWN:

                if event.key == pygame.K\_s and pygame.key.get\_mods() & pygame.KMOD\_CTRL:

                    weights = [self.population.cars[-1].brain.weights1, self.population.cars[-1].brain.weights2, self.population.cars[-1].brain.weights3]

                    self.saveWeights(weights)

            if self.population.dead:

                print("Dead")

                print(self.population.bestCarFitness)

                self.population.createNextGeneration()

            self.statsPane.show(self.screen, self.population.bestCarFitness, self.population.generation, 30) #updates stats pane ++ Test 12.2

            for t in self.tracks:

                t.show(self.screen)

            self.population.update(self.lines, self.CHECKPOINTS)

            self.population.show(self.screen)

            for c in self.population.cars:

                if c.framesAlive >= 120 and len(c.collidedCheckPoints) == 0:

                    c.dead = True

            pygame.display.update()

            self.fpsClock.tick(60)

### borderLineGenerator.py

class BorderLineGenerator:

    def \_\_init\_\_(self, tracks, rows, columns, trackSize):

        self.horLines = []

        self.verLines = []

        self.tracks = tracks

        self.rows = rows

        self.columns = columns

        self.trackSize = trackSize

    def generate(self):

        self.calculateNeighbours()

        self.calculateBorderLines()

        #print(self.horLines,'\n\n\n\n', self.verLines)

        self.condense()

        return (self.horLines, self.verLines)

    def calculateNeighbours(self):

        for t in self.tracks:

            if ((t.tileID + 1) % self.columns != 0 ) and self.findTrack(t.tileID + 1):

                t.east = True

            if (t.tileID % self.columns !=0) and self.findTrack(t.tileID - 1):

                t.west = True

            if (t.tileID > (self.columns - 1)) and self.findTrack(t.tileID - self.columns):

                t.north = True

            if (t.tileID < (self.columns \* (self.rows-1))) and self.findTrack(t.tileID + self.columns):

                t.south = True

    def calculateBorderLines(self):

        for t in self.tracks:

            if not t.north:

                self.horLines.append((t.x, t.x + self.trackSize, t.y))

            if not t.south:

                self.horLines.append((t.x, t.x + self.trackSize, t.y + self.trackSize))

            if not t.west:

                self.verLines.append((t.y, t.y + self.trackSize, t.x))

            if not t.east:

                self.verLines.append((t.y, t.y + self.trackSize, t.x + self.trackSize))

    def condense(self):

        self.mergeSort(self.verLines, 2)

        self.mergeSort(self.horLines, 2)

        self.verLines = self.groupAndSort(self.verLines)

        self.horLines = self.groupAndSort(self.horLines)

        self.verLines = self.adjoin(self.verLines)

        self.horLines = self.adjoin(self.horLines)

    def findTrack(self, ID):

        #Simple binary search

        found = False

        index = -1

        first = 0

        last = len(self.tracks) - 1

        while first <= last and not found:

            midpoint = (first + last)//2

            if self.tracks[midpoint].tileID == ID:

                found = True

                index = midpoint

            else:

                if self.tracks[midpoint].tileID < ID:

                    first = midpoint + 1

                else:

                    last = midpoint -1

        return found

    def mergeSort(self, myList, x): #recursive mergeSort algorithm

        if len(myList) > 1:

            mid = len(myList)//2

            lefthalf = myList[:mid]

            righthalf = myList[mid:]

            self.mergeSort(lefthalf, x)

            self.mergeSort(righthalf, x)

            i = 0

            j = 0

            k = 0

            while i < len(lefthalf) and j < len(righthalf):

                if lefthalf[i][x] < righthalf[j][x]:

                    myList[k] = lefthalf[i]

                    i += 1

                else:

                    myList[k] = righthalf[j]

                    j += 1

                k +=1

            while i < len(lefthalf):

                myList[k] = lefthalf[i]

                i += 1

                k += 1

            while j < len(righthalf):

                myList[k] = righthalf[j]

                j += 1

                k += 1

            #print(myList)

    def groupAndSort(self,anArray):

        """turns 1d array of sorted tuples by last element into 2D array of grouped tuples sorted"""

        groupedLines = []

        groupedLine = []#array of tuples with same last element

        currentLast = anArray[0][-1]

        for l in anArray:

            if l[-1] != currentLast:

                self.mergeSort(groupedLine,0)

                groupedLines.append(groupedLine)

                groupedLine = [l]#clear and add new line

                currentLast = l[-1]

            else:

                groupedLine.append(l)

            #add the last line

        self.mergeSort(groupedLine,0)

        groupedLines.append(groupedLine)

        groupedLine = [l]

        return groupedLines

    def adjoin(self, sortedGroups): #Joins up adjacent lines

        adjoinedLines = []

        for group in sortedGroups:

            if len(group) == 1:

                adjoinedLines.append((group[0][0], group[0][1], group[0][2]))

            else:

                start = group[0][0]

                end = group[0][1]

                constant = group[0][2]

                for line in group[1:]:

                    if line == group[-1] and line[0] == end:

                        adjoinedLines.append((start, line[1], constant))

                    elif line[0] == end:

                        end = line[1]

                    else:

                        adjoinedLines.append((start, end, constant))

                        start = line[0]

                        end = line[1]

                    if line == group[-1]:

                            adjoinedLines.append((line[0], line[1], constant))

        return adjoinedLines

# Testing

## Testing table

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Test no: | Objective no: | Test Description: | Test input: | Input type: | Expected Outcome: | Actual Outcome: | Evidence: |
| 1 | 1 | Main menu should be displayed with working buttons. | Run main.py and press buttons on menu. | Valid | Each button pressed should carry out its action. | Success. | [Test 1](#_Test_1) |
| 2 | 2 | User should be able to choose rows and columns for map size | Enter two integers 8 and 10. | Valid | 8 by 10 map should be displayed with red tiles. | Success. | [Test 2](#_Test_2) |
| 3 | 2 | User should be able to choose rows and columns for map size | Enter a stirring ‘hi’ and then try entering a real ‘9.08’. | Erroneous | Error window should be displayed for both. | Success. | [Test 3](#_Test_3) |
| 4 | 3 | User should be able to select blocks | Select blocks on the map. | Valid | When the user clicks on a block, it should turn light blue to indicate that it has been selected. | Success. | [Test 4](#_Test_4) |
| 5 | 3 | Undo and Clear All buttons should work | User places a tile and presses undo.  User presses Clear All. | Valid | After pressing undo, the ast placed tile should reset. After pressing clear all, all tiles should reset. | Success. | [Test 5](#_Test_5_1) |
| 6 | 3 | Undo and Clear All buttons should work | User attempts to undo before placing their first tile. | Erroneous | Error message displays telling the user they cannot undo. | Success. | [Test 6](#_Test_6_2) |
| 7 | 3 | User can select start button | User presses ‘pick start tile’ | Valid | Window popup will appear telling the user to select a start tile. Next tile selected should have an ‘S’ on it. | Success. | [Test 7](#_Test_7_1) |
| 8 | 3 | User attempts to save map without starting tile selected. | User presses save. | Erroneous |  | Success. | [Test 8](#_Test_8_1) |
| 9 | 3 | User attempts to save map with a start tile selected | User presses save | Valid | Map should save in map file, then the correct map should be able to be used in the simulation. | Success. | [Test 9](#_Test_9) |
| 10 | 4 | Radars should be displayed to detect walls. | None, all part of the simulation process. | Valid | Green beams should extend from the cars to the walls in 5 directions. | Success. | [Test 10](#_Test_10) |
| 11 | 5, 6 | Cars should die upon impact with wall; cars should regenerate when the last car is dead. | None, watch until a car makes it through the track. | Valid | Eventually, cars should be able to make it through the track. | Success. | [Test 11](#_Test_11) |
| 12 | 7 | Display Stats | None, watch the cars die and see the change in fitness level and generation. | Valid | At the start there should be no fitness and the generation number should be 1. This will change when the cars die for the first population. | Success. | [Test 12](#_Test_12) |
| 13 |  | Change number of cars per generation. | Enter ‘10’ for number of cars | Valid | 10 cars should appear each generation | Success | [Test 13](#_Test_13) |
| 14 |  | Change number of cars per generation | User enters ‘abcd’ | Erroneous | Error message should appear | Success | [Test 14](#_Test_14) |
| 15 | Context detailed on evidence page |  |  |  |  |  | [Test 15](#_Test_15) |

## Test evidence

### Test 1

Graphical user interface, text, application, chat or text message

Description automatically generated

Graphical user interface, text, application

Description automatically generated

Graphical user interface, text

Description automatically generated

Test outcome: As intended, the buttons all lead to the right places, apart from ‘ADD/REMOVE OBSTACLES’ as this program has not been made. Test is successful.

### Test 2

Chart

Description automatically generated

### 

As intended, the 8x10 map is displayed with buttons ready to be selected for the track. Test is successful.

### Test 3

Graphical user interface, application

Description automatically generated

In both erroneous instances, the entry error window popup is displayed, prompting the user to re-enter the values. No map design GUI is displayed. The test is successful.

Graphical user interface, text, application

Description automatically generated Graphical user interface, text, application, chat or text message

Description automatically generated

### Test 4

Chart, bar chart

Description automatically generated

As intended, selected squares change colour to blue, as part of the track. Test is successful.

### Test 5

Chart, bar chart

Description automatically generated

Pressing undo deleted the tile at (3,7), as it was the last placed tile.

Pressing Clear All deleted all tiles and reset the GUI. Test is successful.

Chart

Description automatically generated

# 

### Test 6

Chart

Description automatically generated

When the user presses redo, the error message is displayed. Test is successful.

Graphical user interface, text, application

Description automatically generated

### Test 7

Graphical user interface, text, application, email

Description automatically generatedChart

Description automatically generated

Test is successful.

### Test 8

Chart, bar chart

Description automatically generated Graphical user interface, application

Description automatically generated

As there is no start tile selected, an error is displayed prompting the user to select a start tile before saving. Test is successful.

### Test 9

Chart

Description automatically generated

A screenshot of a computer

Description automatically generated with medium confidence

Map saved in maps folder.

### Graphical user interface, application Description automatically generated

User can then select the map when running the simulation.

Chart

Description automatically generatedA screenshot of a computer

Description automatically generated with medium confidence

After pressing start, simulation begins with the correct map. Test is successful.

### Test 10

Radars shown. Test is successful.

### A picture containing laser Description automatically generatedTest 11

Graphical user interface, text

Description automatically generated

s

Chart

Description automatically generatedBackground pattern

Description automatically generated with low confidence

Eventually, the first car passes the halfway point, and coincidentally completes the track. Subsequent generations are successful. Test is a success for this simple map.

First test: simple map. On the first generation they all die relatively quickly before the first turn.

### Chart Description automatically generated

For this more complicated map, it takes 20 generations for one car to traverse fully around the track.

Interestingly, no cars decided to turn left at the bottom, they just carried on going straight, shortcutting the track.

### Test 12

# A picture containing chart Description automatically generated

# Text Description automatically generated with low confidence

### Test 13

At the start, the best fitness is 0 and current gen is 1. After the first generation of cars die, the stats pane updates. Test is successful.

Graphical user interface, application

Description automatically generated

When the user enters 10, the corresponding number of cars appear correctly. Test is successful.

Background pattern

Description automatically generated with low confidence

### Test 14

Graphical user interface

Description automatically generated

Graphical user interface, application

Description automatically generated

### Test 15

Context:

**Problem details**

The end user said that the cars, being arrow shaped, were not realistic, although reasonable for the program. They requested that I use an actual car png to represent the cars.

**Solution**

In order to solve this, I created a new file: newcar.py. It contained mostly the same contents as the previous car class, but optomised for a sprite rather than a drawn polygon.

A picture containing text, yellow

Description automatically generated

The majority of the test was successful, however the cars did not rotate at the correct angles at times. Due to time issues, I was unable to solve this issue.

# Evaluation

## Evaluation table

|  |  |  |
| --- | --- | --- |
| Objective | Feedback from end user | How could it be improved |
| Designing a display menu  Display option to design a map  Display option to add/remove obstacles from maps  Display option to run simulation  Quit program | The end user was satisfied with the design of the main menu. It was simple and easy to use. However, as I did not manage to create the algorithm for adding or removing obstacles, they suggested I remove it as it is misleading. | Remove the add/remove obstacles option. Otherwise it is perfect. |
| Creating a user-designed track  Allow user to design shape of map with a Tkinter GUI  Allow user to place obstacles  Allow user to undo/redo actions using a stack module  Allow user to select start and end locations | Most of the described functions of the map creator GUI were added successfully. The user can undo and redo with the button provided, and squares can be selected to part of the map. However, the end user expressed that the lack of option to select an end location may have hurt the program, as they noticed some tracks where cars could not traverse successfully. | To improve this, I would implement an option to select an end location, and optomise the generic algorithm so the cars know what the end square they’re meant to go to is. Therefore the fitness of cars will be based on distance to end location as well as frams alive and distance travelled. |
| Allowing user to create a map  Display entry box for map size  Should display a map creator window, with the correct size and an option to select blocks  Must prompt user to select a starting block  User should be able to save map and choose the location and file name  Map should save to a folder called ‘Maps’ | The end user said that the display entry box for map size works well, with exception handling ensuring that no program errors prevent the code from executing. Moreover, the map creator GUI displays as intended with the correct size (in terms of rows and columns).  The program prompts the user to select a starting block when they try to save without doing so, by using a popup window. The end user expressed how easy it was to save the map to a file location, as a maps folder is made for the user within the program. | Aesthetically: The user suggested an option to select the colour of the tracks and walls in order to promote customisation. |
| Creating a neural network  Network should be able to detect wall boundaries with 5 sensors  Distance from wall edges will be considered as inputs to each starting neuron  Outer layers of neural network should be commands to turn left, turn right, slow down or speed up  Outputs of input neurons will determine which command will be executed | As the end user cannot see the neural network for themselves, they could not give feedback on this. However, the cars managed to complete most of the tracks successfully. | A possible way to improve the neural network is to add or remove hidden layers. I have not tested whether or not this would make the cars complete the tracks faster or not, so this would be an interesting test to carry out. |
| Cars should run around the chosen track as quick as possible without crashing  Cars that move the furthest gain a higher fitness  Cars that move the fastest gain a higher fitness  Higher fitness cars regenerate in the next generation | As the program intends, cars eventually manage to run around the track without crashing, as quick as possible. The highest fitness is displayed to prove this,and cars regenerate successfully. | One way to improve this was by changing the car model from arrows to actual car pictures. I began to improve this in [test 15.](#_Test_15) |
| Allowing user to run car simulation  Neural networks should trigger commands turn left, right, speed up or slow down  Algorithm senses when the edge of a track is hit by a car  This car will die and be deleted  Algorithm selects cars with the highest fitness and regenerates them when cars die  Algorithm runs until a car makes it through the track | The end user is satisfied with how the simulation looks. As the user can select the number of generations per car, they can decrease this to make it more clear what cars are doing, as when the simulation is run using the default 30 cars per generation, the majority die within the first two squares of the track. When cars die, they are removed from the screen, and the car with the highest fitness is highlighted in green to show it is the best car of the previous generation. | None. |
| Display map and stats interface  Decompress map from PKL file and display in pygame window  Display cars left for each generation  Display generation number | The end user said that the stats interface is useful for their research, accurately displaying the fitness and generation number. However, they suggested that the fitness is updated every frame, so they can see how fast it increases. | In order to implement this change, I would need to update the stats pane every frame, and this may cause the simulation to lag. For this reason, I decided not to add this change when creating my technical solution; I would need a find a different way to implement this. |
| Allowing user to quit  Exits the program when option is selected. | The end user can quit on command from the main menu. The end user suggested that I add a quit button to the actual simulation GUI to make it more user friendly. | To improve this, I would add a quit button on the pygame screen near the stats pane. |

## Final interview with end user

Q: What are your overall thoughts on the program?

**A: The program works very well. The majority of objectives were met without much issue, so I’m happy with the end result. The main menu works well, with the map creator being simple and easy to use. Although I would have liked an option to select an end tile as you did with the start tile. Sometimes when loops were created in the map, the cars ignored them as if to take shortcuts. Because of this, they weren’t really traversing the whole track as intended. I think you should mark points where the cars have to cross, but I understand this is hard to do with the limited amount of time you were given.**

**The simulation works well; cars died early in earlier generations and constantly improved until the whole track was able to be completed, so this was done well. I did notice that sometimes it would take a while for generations of cars to reach a breakthrough, and they would continue dying at the same point for a while, but I assume this is down to the mutation levels I selected. When I chose a mutation level of 0, all the cars followed the car with the best fitness after the first generation, so I know the mutation works. Also with a mutation level of 100, the cars did not follow the best car, but I didn’t stay to see whether one would eventually successfully travel around the track as it was unlikely.**

**The option to choose the number of cars per generation helped as it allowed me to see individual cars clearer, so this was an important aspect I liked about your project.**

**The ability to save and load weights was also interesting; I found that weights that were used on simulations where cars had completed tracks worked again when loaded for the same track, so it is clear this works as intended.**

**To improve the project, I would suggest you also implement the option to add or remove obstacles, possible within the map creator GUI. This is just to reflect real life more accurately, as I know that an acceptable limitation of yours was that you could not create the simulation in 3d.**

Analysis of response: It is clear that I have satisfied my end user, so the project is a success overall. As the project was intended to educate my end user about the application of neural networks and algorithms working to create the simulation, perhaps I could have added more points to the simulation that indicated where each part of the program worked, for example which weights were chosen for each generation. This would give a better insight to why the cars were improving after each run.

To create the obstacles, I would use the Tkinter module again. First I would need a popup to select the map file, and this would need to be displayed. Then, I would need to select different shaped obstacles could be blit onto the track, and then edit the Population class so that cars die whenever obstacles are touched.

# Bibliography

NEAT Python - <https://neat-python.readthedocs.io/en/latest/neat_overview.html>

### Videos that helped me gain understanding of deep learning:

<https://youtu.be/3bhP7zulFfY>

<https://youtu.be/r428O_CMcpI>

Similar system - <https://youtu.be/Cy155O5R1Oo>

Help on neural networks - <https://realpython.com/python-ai-neural-network/>

Tkinter - <https://realpython.com/python-gui-tkinter/>

Numpy - <https://numpy.org/>

Waymo - <https://waymo.com/waymo-driver/>