Ai racer

Analysis:

Introduction

My project idea is to create and train a neural network using reinforced learning to be able to play a simple top-down 2D racing game and to achieve faster times as it is trained. My aim is to produce an AI which can make autonomous decisions while driving on a track of when to turn, accelerate, brake, etc to reach the finish line. I want to do this project specifically as I have been interested in reinforced learning as a method of machine learning for a long time and will gain a much deeper understanding of the process behind it than I already have by implementing reinforced learning myself inside of a game. To make the project outstanding, I could also implement additional features such as the ability for the user to race against the AI to make the game more engaging and fun to strengthen my users' interest in reinforced learning.

I have been inspired by watching videos and reading articles previously about how reinforced learning can be used especially to play games. Reinforced learning is suitable for this as this branch of machine learning can take positive or negative signals from the game based on how well it has performed and can use that to fine tune the neural network and perform better in the future.

To complete this project, I must research and learn the mathematics behind machine learning and how neural networks can be implemented in python by using various online and offline materials.

Similar ideas I have researched:

A similar project which uses reinforcement learning inside of a game is Andrej Karpathy's pong from 2016.

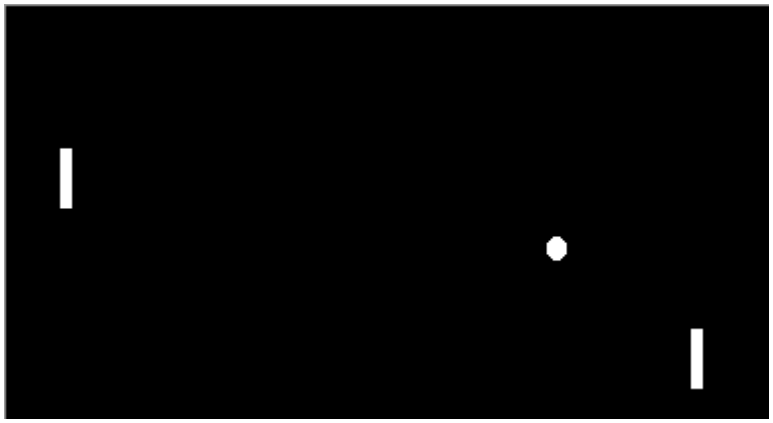


Figure 1 Karpathy's Pong from Pixels (https://karpathy.github.io/2016/05/31/rl/)

Here, Karpathy used reinforcement learning to give an array of every pixel in the game pong to a neural network. Then that neural network uses the input to decide if the paddle should move up or down to increase its chances of winning. This program allows the user to train the neural network from scratch and slowly improving by reinforcement learning and allows the user to watch the AI play a game against a hard coded pong bot.

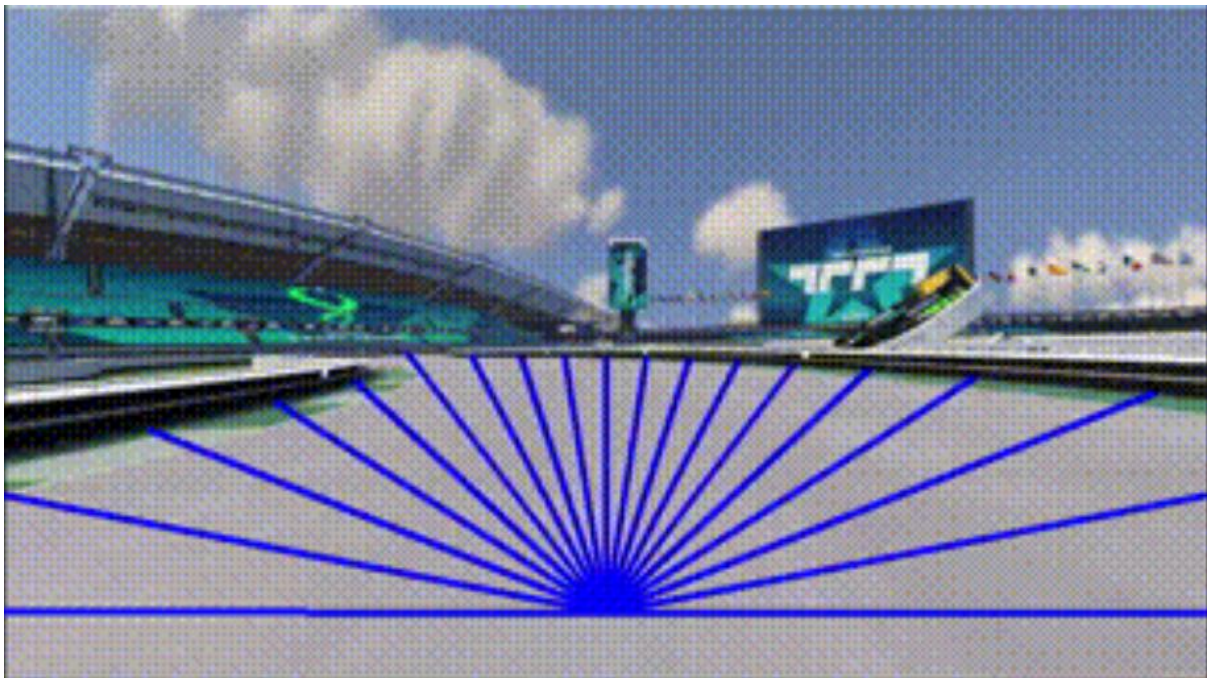
Positives	Negatives
The use of reinforcement learning to play a game is a very similar idea to what I want to implement.	The user interface is very bare and allows for very little interaction from the user.
It is very clear what is happening on screen due to its simple nature.	The colour scheme and design of the graphics is very simplistic which makes watching the AI less engaging.
I want to use and train a neural network to play a game and to update its internal values to become more efficient.	The game of Pong used is very simple with the only option for the neural network is to move up or down which is less enjoyable for the user.
Here, machine learning is used to optimally play a video game which is what I want to implement.	Karpathy chose to give the neural network the state of every pixel on screen each frame however this would result in the scope of the neural network to become too large for this project so I will reduce the number of inputs received to specific values in the game.
The AI is very strong at pong and can consistently beat a very good hard coded bot with enough training.	

Features I want to incorporate:

I want to use reinforcement learning to train a neural network instead of other methods of machine learning.

I want to use reinforcement learning to train a neural network to play a game specifically as this is an easy to visualise concept for the users.

Trackmania is a 3D racing game with an emphasis on user generated content such as custom tracks and cars. TMRL (Trackmania reinforcement learning) is a project created using the game where a neural network is trained using reinforcement learning to drive the car in a 3D environment. The neural network takes in various input parameters such as the speed of the car, the distances from the sides of the track and the image on the screen and uses it to calculate optimal values for the car's steering, acceleration and braking for its next frame.



Example of some information gathered by the AI about the game environment (trackmania-rl/tmrl: Reinforcement Learning for real-time applications - host of the TrackMania Roborace League)

Positives	Negatives
The use of reinforced learning inside of a racing game is especially like what I want to create and is an engaging concept.	The 3D racing game with advanced physics is far too complex to be used for this project.
The pretrained AI is very strong at playing the game and can beat many human players.	This AI uses many input parameters such as an image of the screen which is too complex to use for this project.
The graphics and user interface of trackmania are realistic and engaging and I would also like to use realistic colours	The AI itself comes with no user interface and requires editing of files by the user to configure which makes the program more difficult to use.

for my game to convey the environment more clearly to the user.	
TMRL comes with a pretrained AI so that users can experiment with it without requiring extensive testing as well as the ability to create one from scratch.	
The AI can be used effectively on a large variety of tracks.	

Features I want to incorporate:

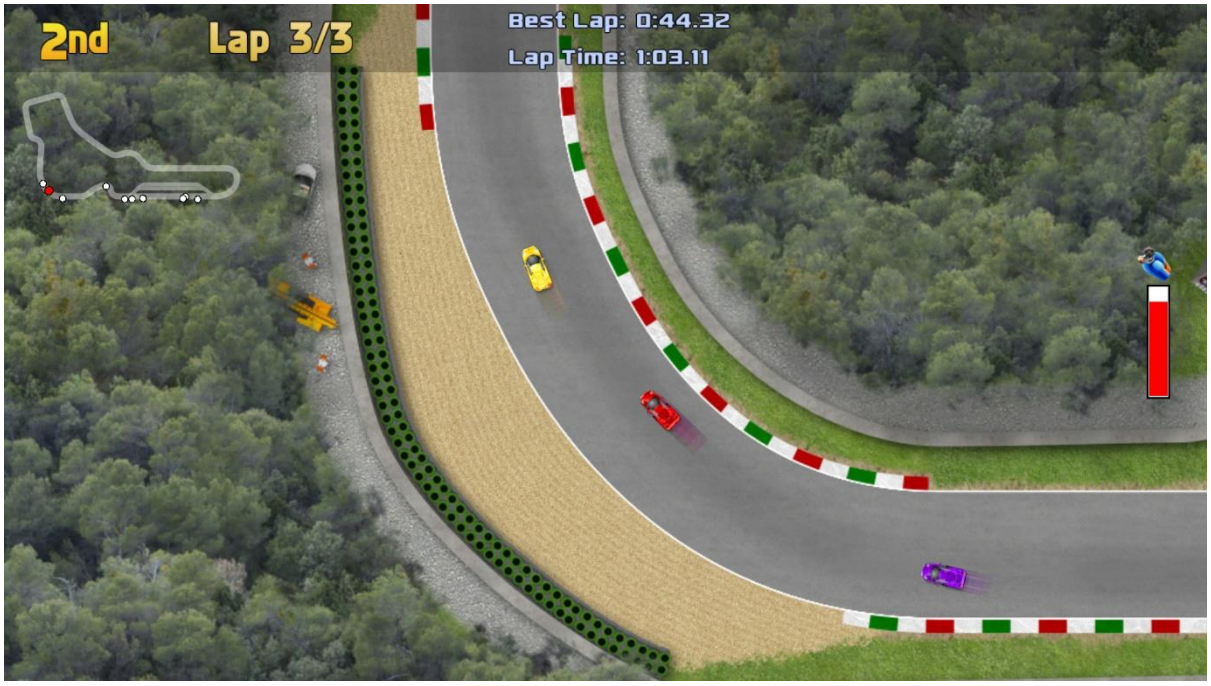
I want to use reinforcement learning inside of a racing game as this is both is an engaging concept and reinforcement learning is also suited to be used for video games.

I want to use a similar colour scheme with realistic colours to convey the environment so that it is easy to understand for the user and that the cars are visually distinct.

I want my project to contain a pretrained AI which can be immediately used by the user so that they do not have to sit through the long training process to use the AI.

I want my neural network to be able to drive correctly on tracks with different layouts so that it can learn the core aspects of the racing game and not just how to beat one specific track.

Ultimate Racing 2D is a top-down racing game featuring many unique tracks and realistic physics. This game does not feature any machine learning however the 2D racing game aspects are very similar to what I want to use in my own project as the game environment that the AI interacts with.



Screenshot of the gameplay from Ultimate Racing 2D (Save 70% on Ultimate Racing 2D on Steam)

Positives	Negatives
The game is a 2D racing game which is relatively easy to create while still allowing for some depth and enjoyment.	This game does not feature any machine learning which is a key feature I want to use in my project.
The car's physics are realistic which allows for a more enjoyable experienced from the user.	This game features different types of cars with different attributes such as speed which is not suitable for my project as each race should be similar so that the AI itself can be compared.
The cars have many mechanics such as drifting which adds a good level of complexity to the game.	The game's tracks are very long which would increase the time it would take to train an AI to play it.
The game's user interface is very simple but conveys all relevant information in the game.	
The game has many unique tracks and cars so that it is replayable.	
The game has simplistic but realistic graphics and art which I would be able to achieve something similar while still being easy to understand.	

Features I want to incorporate:

I want to create a similar 2D racing game for the AI to be able to play.

I want to add similar car physics and game mechanics so that there is more depth to the gameplay making it harder for the AI to perfect.

I want to use a similar art style and user interface so that all the information on screen is very clear to the user so that they can focus on the machine learning itself.

I want to have multiple different tracks in the game so that the AI can quickly adapt to new tracks created and is sufficient at using the core game mechanics instead of the series of inputs required to beat one specific level.

Stakeholders:

The stakeholders of my project are students who are interested in machine learning and artificial intelligence but who may not have personal experience with working with it. My target demographic is students aged between 14-18 as they are likely to have an initial curiosity about how machine learning can be used but not a full idea of how it works and may have not previously been encouraged to explore the concept independently. My program is targeted towards a male demographic as there is sadly a larger proportion of males who are interested in machine learning and the racing game setting is also a genre with a largely male player base however females interested in machine learning and AI are also able to use this program.

I intend for this project to be used to show my users an example of machine learning in action inside of a familiar and entertaining context. This project may be used by the users on a personal computer at home if they are intrigued by the concept and want to experiment with machine learning or it may be used by teachers in a classroom to provide their students with an enjoyable and educational lesson outside of the usual curriculum. My hope is that the project itself is engaging and fun for students so that it does not seem initially daunting to begin learning about AI but using the project inspires the students to research further into the topic and develop their own understanding and the program will contain a link to further resources if the users want to then learn about how machine learning works.

I want to involve my stakeholders in the initial design process by surveying students from my college with an interest in machine learning with what they would want to see from my program and in the final testing process to review if they feel the success criteria has been met and to get their feedback on a finished version of the program.

Survey results:

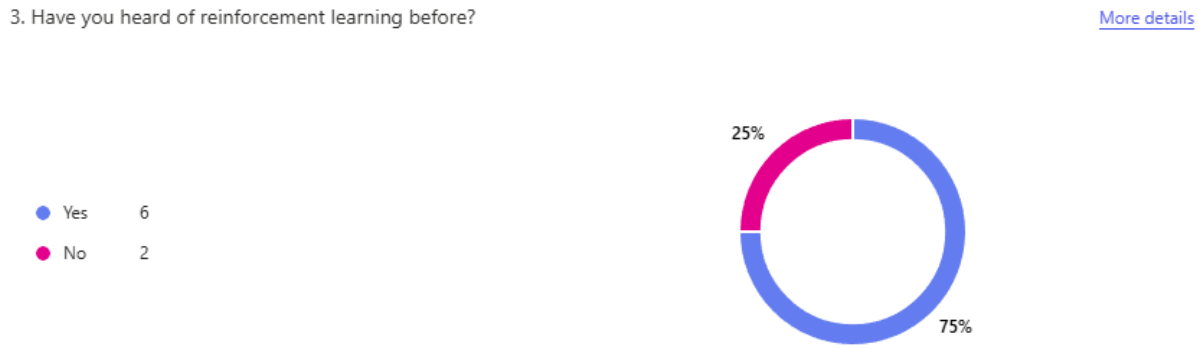
I asked computer science students at my college who are interested in machine learning and AI specifically questions from an online form.

1. Would you prefer a racing game played using machine learning to be more educational to teach people about machin e learning or more for fun? More details

Most stakeholders would like the project to be mostly for entertainment so an emphasis will be made to make the project engaging and fun however this is somewhat split between respondents so an emphasis will be placed on both areas.

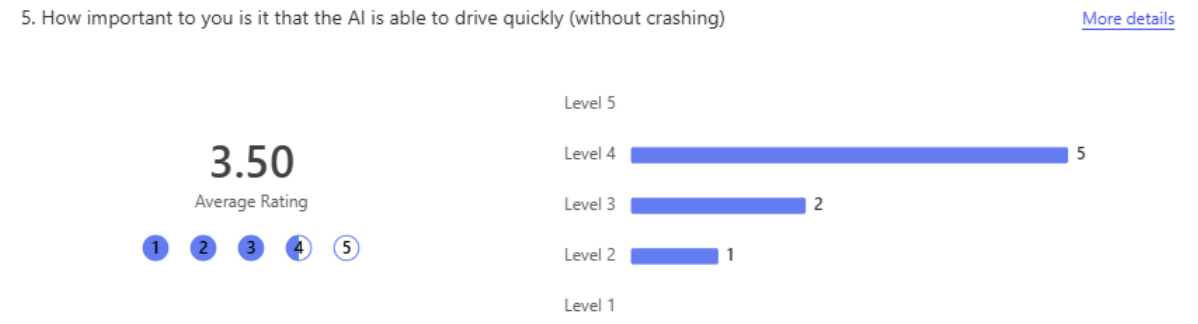
2. How much prior knowledge do you have about AI and machine learning? More details

The stakeholders surveyed have a mixture of a fair knowledge base on AI and a vague idea about the concept so some basic knowledge may be assumed in my program about what AI and machine learning is and what it is used for however, to make my project as accessible and educational as possible a small amount of prior knowledge will be assumed for any educational aspects.



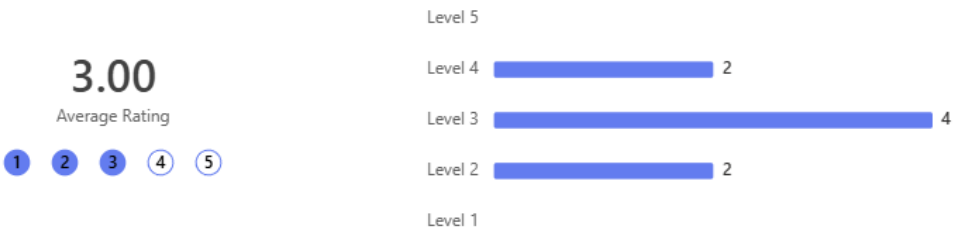
Many stakeholders have heard of the concept of reinforced learning so it may not need to be completely introduced from scratch however to continue making the project more accessible no detailed knowledge about the process of reinforcement learning will be assumed from the users.

Most users view it as very important that the AI can consistently reach the finish line and complete a track so this will be the focus when training the neural network.



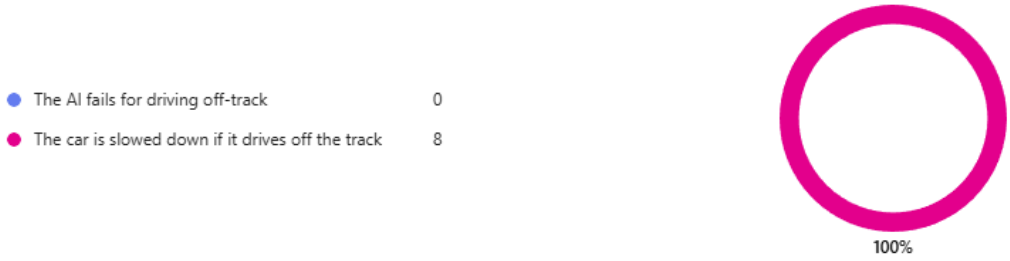
It is viewed as less important that the AI can drive quickly with 0 stakeholders rating it as the highest priority so when training the AI, the speed of it will be a lower priority but still paid attention to.

6. How important is it to you that the track(s) are difficult (such as with many twists and turns) More details



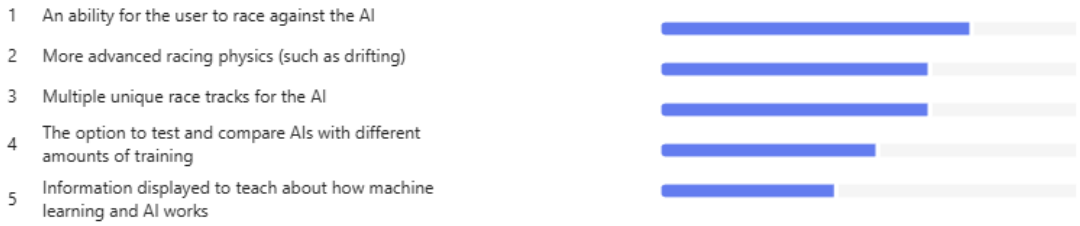
The complexity of the tracks is viewed as less important than the performance of the AI so there will be some focus on this area but not enough to inhibit the AI from performing quickly or consistently finishing the race.

7. Would you prefer the Ai to fail if it drives outside of the track or for it to just be slowed down? More details



Everybody surveyed stated that they would prefer the AI to just be slowed down when it drives off of the track so this will be implemented as a feature.

8. Which additional features below are most interesting to you for a project where an AI plays a racing game. More details



Out of the non-essential features asked about, the ability for the user to race against the AI and more advanced racing physics and abilities were highly sought after so they are important to be completed whereas the other features will have less of an emphasis to be developed.

Requirements:

	Feature	Explanation	Justification	Importance
1.	Display window.	The car racing along the track will be visually shown in a main window which will also contain the user interface.	This must be displayed to the user so that they can see and understand the AI's decisions.	Essential
2.	A car which can be controlled.	There will be a simple car which can be controlled in a top down 2d environment with simple abilities such as accelerating/ decelerating and turning.	Having a car to be controlled by the Ai is a crucial part of the project as this is what the Ai will learn to use.	Essential
3.	A racetrack.	There will be a track for the car to race on with a defined start, finish and sides of the track.	Having a complex track layout with turns will develop a more sufficient Ai which can respond to changes on the track which both makes the program more engaging for users but also shows more of the capabilities of machine learning.	Essential
4.	A neural network	There will be a neural network which will take various information about the car in the input layer and output probabilities of the car making various actions such as accelerating.	The neural network is a fundamental part of this program and will be used to control the car and show the user an example of machine learning being used in action.	Essential
5.	More advanced physics	Including more advanced physics will include the ability for the car to drift on the race train and the	The implementation of more advanced physics gives is rated as highly important to my stakeholders, so it is a	Important

		effects of resistive forces on the car.	desirable feature for my project. The Ai more options of how to traverse the course as well as making the game a more realistic simulation of real life which will be more engaging for the user to view.	
6.	Ability to play against the Ai	This will allow for the user to control their own car and try to beat the neural network in a race.	The implementation of this feature is rated as the most important to my stakeholders, so it is a very important feature to implement to make the project more engaging. This gives the user interactivity with the AI which will cause the users to spend more time on the program and have a better experience.	Important
7.	Multiple racetracks	This will allow the AI to perform and race on different tracks with different layouts which means that the AI will need an understanding of the game mechanics in general and not just a specific track.	The implementation of this feature is rated as moderately important to my stakeholders so it is a feature which will add benefit to the engagement of my users. As this causes the AI to have to perform well even on different track layouts, it will make the AI more robust and stronger which will aid in the teaching of machine learning.	Optional
8.	Select Ais with different amounts of training	This will allow the user to watch Ais with a different amount of time having been spent on its training and compare how	This feature will aid in the education of the users as the ability to compare the neural network after different amounts of training will demonstrate how the	Optional

		they perform on the track.	time spent training impacts the performance of the agent.	

Limitations:

Some of the requirements of this project have been labelled as “Important” or “optional”. The features labelled important are requested highly by my stakeholders but are not essential for the functionality of the project and the requirements labelled as optional are not as highly requested by my stakeholders and are also not essential for the functionality of my project and will be completed with lower priority.

More advanced physics:

This feature would make the racing game more realistic and engaging which would aid in the immersion and interest from my users. However, only basic racing options are essential for the project to be completed such as accelerating/ decelerating and turning. Additionally, the implementation of the neural network is a much larger task and is more impactful than the quality of the racing game, so I am prioritising the creation and training of that before implementing additional physics.

Ability to race against the AI:

This feature would allow the users to become more involved with the AI and adds a fun game to the project which would make it more memorable, engaging and educational. However, the AI only needs to be watched completing the tracks by itself for the completion of this project, so this feature is labelled as “important” as the creation and training of the neural network is more crucial than an additional game mode for this project. Additionally, creating this feature will require the creation of an entirely separate game mode with competitive racing elements and user inputs so due to the scope of this project it would be too time consuming to deem an essential feature.

Multiple racetracks:

This feature would cause the AI to have to perform well even on different track layouts, which would make the AI more capable to work in different scenarios which will aid in the teaching of machine learning as neural networks allow for the AI to work well even in

new scenarios. However, only a single racetrack is essential for this project to be completed as this is sufficient to show most of the capabilities of machine learning and the creation of different racetracks will not only cost time to design them to test different skill sets of the neural network's racing but it will also significantly increase the amount of training time required for the neural network to consistently perform well . Additionally, this feature was only moderately requested by my stakeholders so it would not increase the engagement of my users as much as some other features.

Comparing different amounts of training:
This feature would require multiple neural networks to be stored as part of the application to be used and compared with each other which will likely use a large amount of memory and will be complicated to store. Also, it would be much simpler to only work with a single neural network instead of saving its state at multiple points during training to compare with and this was a lowly rated feature by my stakeholders meaning that it should not be of a high priority to include.

Hardware and software requirements:

I will be creating this project in the language python using pygame for my GUI and windows as my operating system. The implementation of a neural network increases the hardware requirements significantly as they are computationally expensive to train and implement.

Hardware:

- This project is just run in python and requires no additional hardware so a typical general-purpose computer can be used.
- A decently capable CPU is required to train the neural network (such as an intel i7/i9 or AMD ryzen 7/9) however the end user will only need to use the neural network and not train it themselves therefore they may be able to use a less effective CPU
- A strong GPU is required to train a neural network (such as an NVIDIA RTX 3080) however the end user will only need to use the neural network and not train it themselves therefore they are very likely to be able to use a less effective GPU
- 8GB of VRAM is also required for training the neural network but this may be less for the final user when the network is fully trained.

- 16GB of RAM is recommended for training a neural network but this also may be less for the final user.
- A large amount of storage may be required to contain the neural network (A terabyte is recommended for training a large neural network however this neural network is small) so this is likely to be required from the user.
- A mouse and keyboard are required for all the major inputs in the program.

Software:

- Both Python and Pygame are available on many operating systems, however as this project is being created in windows, a windows OS may be optimised for this project.
- A graphical user interface is required for the operating system but not much else.
- The user will need to have installed Python and Pygame independently, however these are both free and accessible.

Computational methods:

I believe that this project is suitable for a computational approach because the following methods can be used to complete it:

- Abstraction can be used to ensure that all key elements of the project are focused on and completed. This also means that more unnecessary features such as an improved racing game or image recognition with the neural network do not need to be implemented so that the core features can be created in the given timeframe. Abstraction is a tool widely and easily used for computational problem solving as it is straightforward to decide which elements of a program are more important than others which means the problem is suited to a computational approach.
- Visualisation can be used to effectively show the user's machine learning being displayed in action inside of a visual environment which is a much more natural and engaging way to get students interested in machine learning rather than describing it. Being able to visualise abstract concepts such as machine learning in engaging and interactive ways such as using a racing game is greatly strengthened using a computational approach as concepts can be graphically demonstrated on a screen so the problem is suitable for a computational approach.
- Decomposition will be used to break down the sections of the project into smaller parts to be designed individually which will make the project easier to design as only a smaller problem needs to be solved at a time. For example, the racing game and the neural network elements can be designed separately and

they themselves can be broken down into smaller features to be made. This can be done computationally by dividing processes in the program into modular subroutines which means that a computational approach works well with decomposition of the problem meaning that the solution to the problem is easier to plan and design.

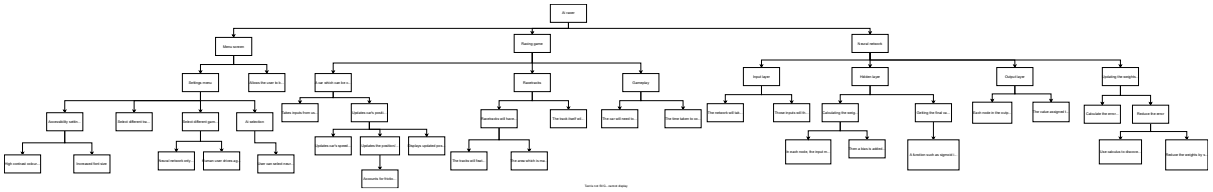
- Some elements of the program will be executed concurrently. For example, the neural network will make the decision of which actions to take when driving in the same frame as the driving game is updated. This is suited to a computational approach as concurrent processing is regularly performed on multi-core processors to execute multiple processes inside of a program faster.
- Q-learning which is a reinforcement learning algorithm will be used to train the neural network to drive the car. This is a suitable algorithm to meet the needs for this project as this method allows the agent to begin training initially making completely random decisions however as the agent receives positive and negative feedback as the results of its actions, its decisions begin to shift towards ones with a higher probability of positive feedback. This means that for a racing game, the car will begin driving randomly and poorly but after further training, the agent will begin to drive in a way which allows it to reach the finish line. This algorithm is good to visually show how machine learning adapts and evolves and will be capable of performing well in the racing game context as positive reinforcement can be getting closer to the finish line and negative reinforcement can be driving off the track and staying still.

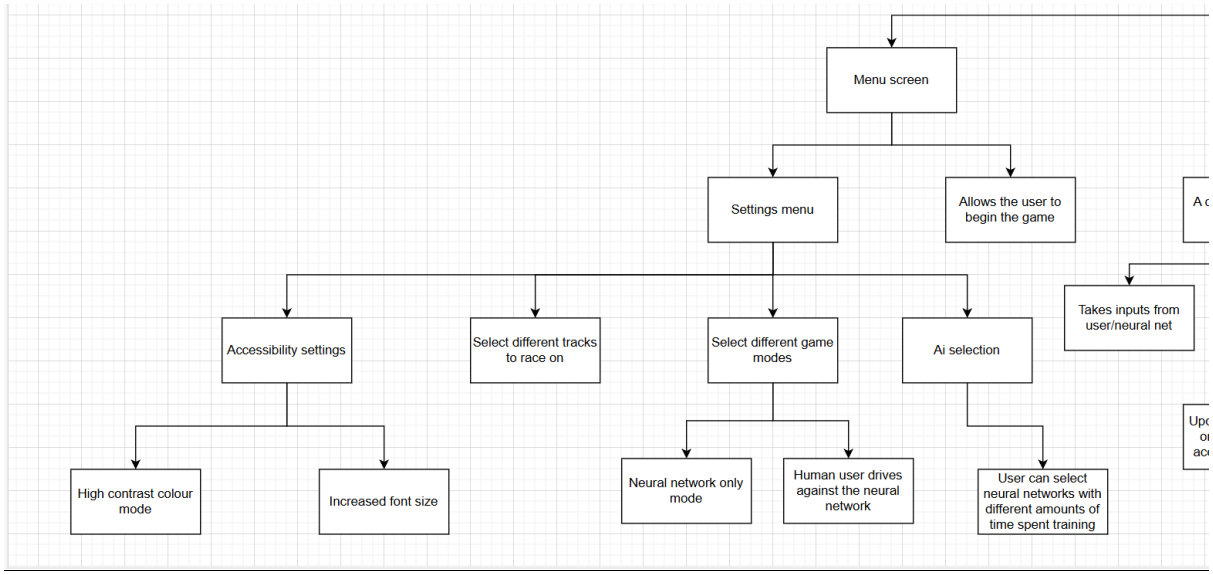
Design:

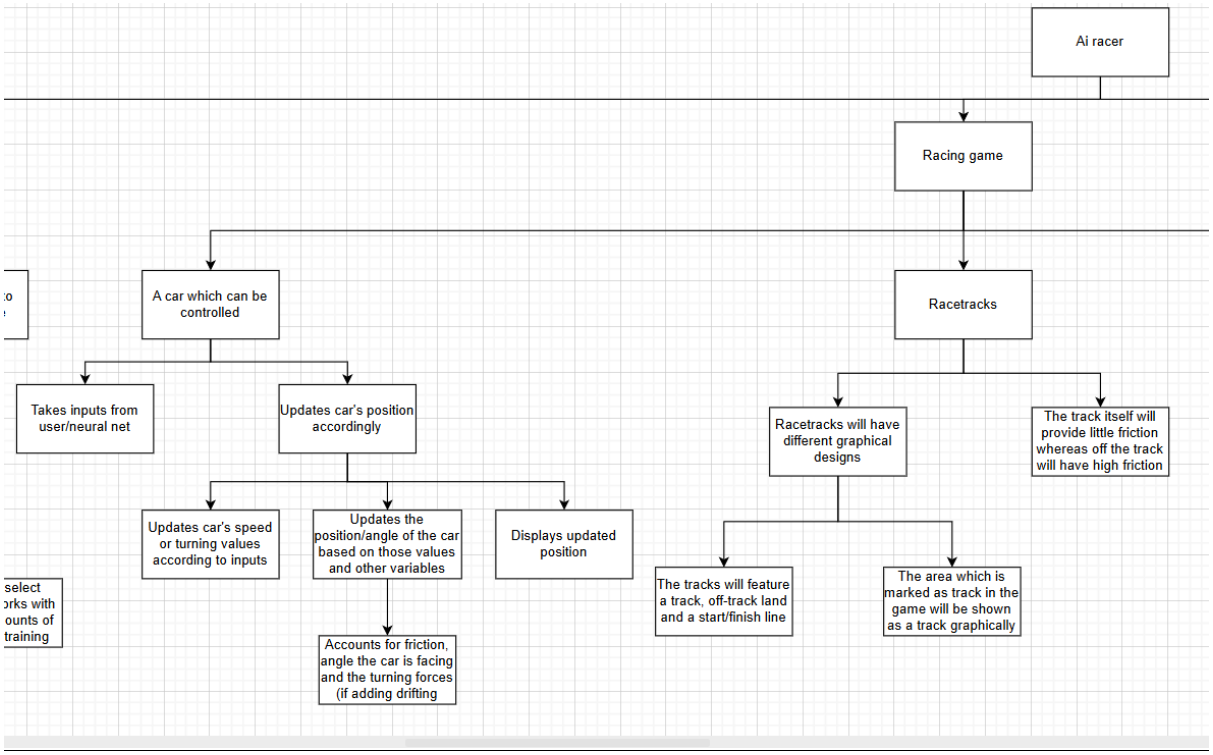
Structure diagram:

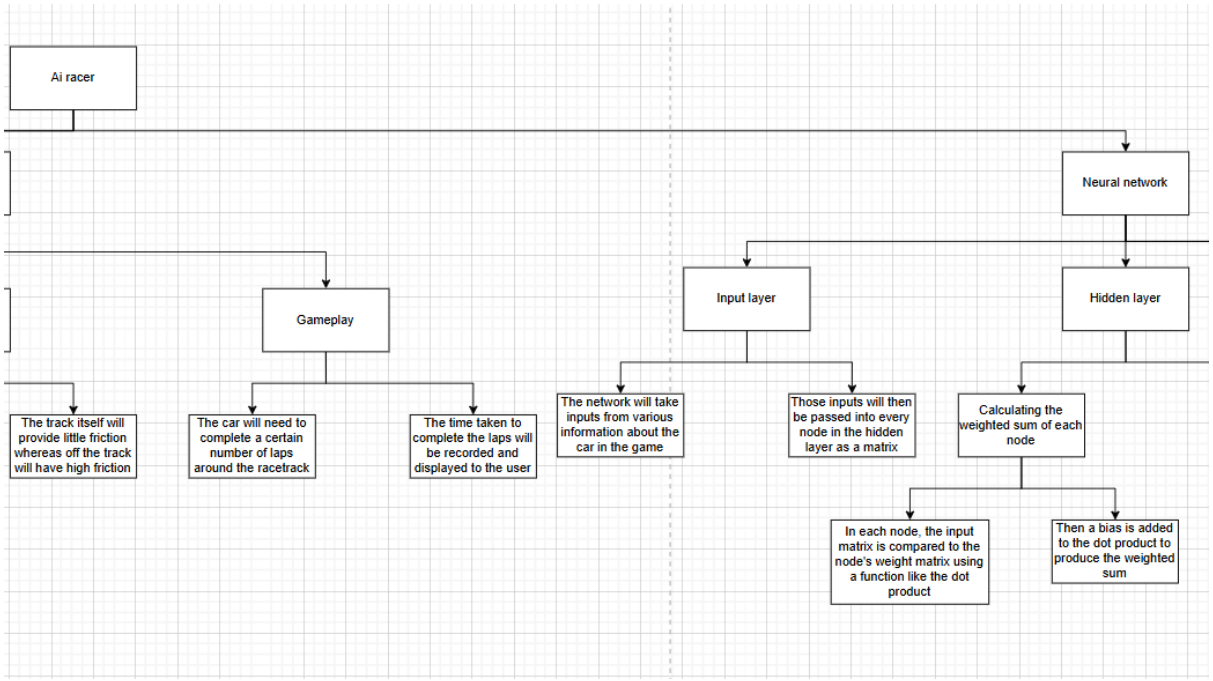
Here is my structure diagram. I decided to break the overall structure into three main sections: the main menu, the racing game and the neural network. This is because the racing game is the game itself which is processed and displayed on screen and functions separately from the neural network which calculates how the car in the racing game should use. The neural network takes values as inputs from the racing game, processes them and then returns to the game what the car should do so although they communicate with each other, the racing game and the neural network are separate processes and can be designed separately. The main menu allows the user to customize the program to their preferences which will bring greater engagement from the user. Then the main menu allows the user to begin the racing game itself which uses

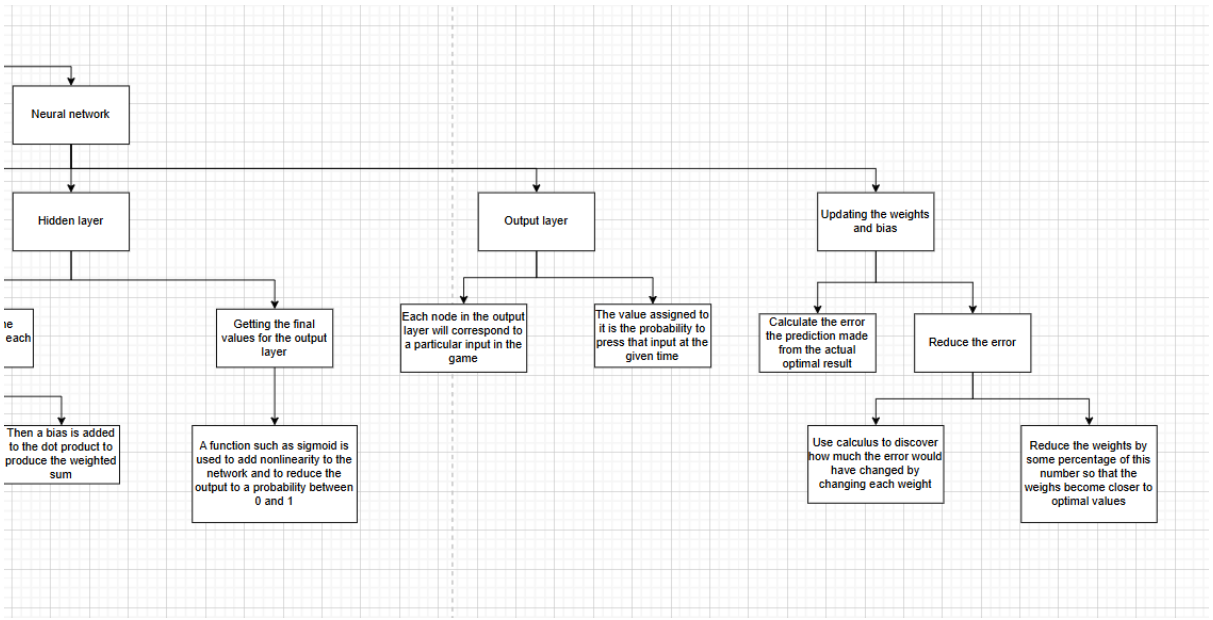
the neural network so the main menu can also be designed as a separate process to the racing game as it is a separate process.











Development plan:

Stage 1- creating a controllable car:

Features to implement:

- A display window to contain the game
- A car which speeds up when up is pressed and slows down when down is pressed

- The car can also turn right when right is pressed and turn left when left is pressed
- Create boundaries at the edge of the display window so that the car cannot leave the screen

In this stage, a display window will be created to display all of the events which occur inside the racing game. Next, a car will be displayed on screen which will accelerate and decelerate with user input and is also able to turn left and right. A frictional force will be applied as well which will slow down the car when it is moving. I will also implement boundaries at the edge of the display window so that the car cannot leave the screen.

I decided to create the controllable car before the gameplay and the track as both of those features require the existence of the car to some extent. For example, the racetrack's collision detection with the car cannot be implemented until the car is created. The neural network which will be trained on the game cannot be trained until all of the racing game elements of the project is built (the car, racetrack and gameplay) so that must be done before the neural network.

Stakeholders voiced that it is important for the game to have accurate and advanced racing physics, so care should be taken to ensure that the car's movement is realistic and fun to use.

Test number	Required inputs	Test description	Success criteria	Pass/Fail
1	Run the program.	When the program is run, see if a display window appears.	A display window will appear.	
2	Run the program.	When the program is run, see if a car is displayed on the display window.	A car will appear inside of the display window.	
3	Run the program and press W.	When W is pressed, see if the car moves forwards.	The car will move forward when W is pressed.	
4	Run the program and press S.	When S is pressed, see if the car moves backwards.	The car will move backwards when W is pressed.	
5	Run the program and press A.	When A is pressed, the car's image will rotate anticlockwise.	The car will turn anticlockwise when A is pressed.	
6	Run the program and press D.	When D is pressed, the car's image will rotate clockwise.	The car will turn clockwise when D is pressed.	

7	Run the program and press and hold A. Then press W.	Pressing A will cause the car to rotate anticlockwise and then pressing W will cause the car to move forwards. The movement of the car should be going forward in the same direction that the car is facing after it is turned.	The car moves in the same direction the car is facing when W is pressed.	
8	Run the program and press and hold A. Then press S.	Pressing A will cause the car to rotate anticlockwise and then pressing S will cause the car to move backwards. The movement of the car should be going backwards in the opposite direction that the car is facing after it is turned.	The car moves in the opposite direction the car is facing when S is pressed.	
9	Run the program and press W then let go.	When W is pressed, the car will move forward. After W is let go, the force of friction should be applied to it.	After W is let go, the car should gradually slow down and eventually stop.	
10	Run the program and hold W until the car reaches the edge of the display window.	The car should be stopped when it reaches the edge of the display window so it cannot leave.	The car will stop moving as it reaches the edge of the display window.	

Stage 2- creating the racetrack:

Features to implement:

- The background of the game should indicate where on the screen is the track, off-track sections and the start/finish line

- The car should store whether it is currently on or off track and update the friction acting on the car due to that

The racetrack must be created next. This is because the gameplay features require a finish line to be defined so that the car is able to win the race so the gameplay stage is dependent on the racetrack being made first.

In this stage, a graphical track will be displayed on the screen as the background. When the car is on the track, one of its attributes will be set to being on the track which will cause the car to have a lower friction value used, however when the car is detected as leaving the track, then its attributes will be set to being off the track and it will have a higher friction value used. Additionally, the start and finish line will also be created and displayed on the track which will allow for collisions with it to be later used.

Stakeholders unanimously agreed that the car should only be slowed down when leaving the track, so implementing that feature will cause the game to be more engaging for its users.

Test number	Required inputs	Test description	Success criteria	Pass/Fail
1	Run the program.	When the program is run, there should be a background image displayed.	A background image is displayed.	
2	Run the program.	The background image should be of a racetrack which includes track, off-track sections and the start/finish line.	The background image is of a racetrack and displays the track, off track and start/finish line.	
3	Run the program and press W then let go on track. Then repeat the same off track.	There should be stronger friction applied to the car off track than on track.	When off the track, the car slows down faster than on the track.	
4	Before running the program, add a temporary line of code which prints the text “Test passed” when the car collides with the start/finish line. Run the program and then drive the car over the start/finish line.	There should be a collision check implemented with the car and the finish line. To test this, the string “Test passed” will be printed when there is a successful collision between	“Test passed” will be printed only when the car collides with the finish line.	

		the car and the finish line.		

Stage 3- creating the gameplay and menu:

Features to implement:

- The car will be tracked for how many laps it makes around the track and once it reaches a set number the game will end
- The car's time spent on each lap will be recorded and displayed when the game has finished.
- A main menu, settings screen and race finish screen will be created which will open when the program is run

Now that the car and the racetrack have been completed, the gameplay is able to be implemented. This has been chosen to be created before the neural network as the neural network needs to be rewarded when it performs well in the game and penalised when it performs poorly, however this cannot happen before the gameplay itself has been made.

The main menu, settings menu and race finish screen will also be made at this stage as some of the settings which will be available to change are directly involved with the gameplay element so should be developed at the same time and the menu is a relatively small feature to include.

Stakeholders should be consulted on which settings they would like to have in the program so that the application can be as customizable as they please and to increase the functionality of the program.

Test number	Required inputs	Test description	Success criteria	Pass/Fail
1	Run the program.	The main menu screen should appear instead of the usual racetrack.	The main menu screen should appear instead of the usual racetrack.	
2	Run the program and click the "start racing" button with the mouse.	This button should cause the racetrack to appear and the gameplay to begin.	The racetrack then appears, and the gameplay begins.	

3	Run the program, press "start racing" on the main menu and drive the car around the track so that it passes over the finish line 3 times.	When the car drives over the finish line for the third time, the race finish screen should appear.	The race finish screen appears.	
4	Run the program, press "start racing" on the main menu.	A timer should appear at the top of the screen which counts the time passed after the race began.	The timer appears and counts down.	
5	Run the program, press "start racing" on the main menu.	A lap counter should appear at the top of the screen which counts how many laps have occurred.	A lap counter should appear at the top of the screen which begins at 1/3	
6	Run the program, press "start racing" on the main menu and race a single lap.	The lap counter should increment by one when the finish line is reached.	A lap counter should appear at the top of the screen which begins at 1/3 and reaches 2/3 when the finish line is first reached	
7	Run the program, press "start racing" on the main menu and race two laps.	The lap counter should increment by one when the finish line is reached each time.	A lap counter should appear at the top of the screen which begins at 1/3 and reaches 2/3 when the finish line is first reached and 3/3 when it is reached a second time.	
8	Run the program, press "start racing" on the main menu and attempt to drive the car in the opposite direction through the finish line.	The game should not allow the car to simply drive back through the finish line resulting in a completed lap so this should not affect the lap counted.	The lap counter is unaffected	

Stage 4- creating the neural network:

Features to implement:

- An input layer for the neural network to take inputs from the game.
- A hidden layer for the neural network to process the given information and make predictions
- An output layer to return the probabilities that the neural network should use each input

After the game is fully completed, then the neural network can begin to be made. The neural network must be created before it is able to be trained therefore it is an earlier stage than training the neural network.

The input layer will take various inputs from the game such as the distance from the side of the track and the speed of the car. These values will then be stored in a vector which will be passed into every node in the hidden layer. Here, the vector's dot product is found with the weight for each node and a bias value is added to find the weighted sum of each node. Then a nonlinear function such as sigmoid is used which reduces the sums to a probability value between 0 and 1 for each output node. The output layer will then return the probabilities to the game which will execute the inputs and then repeat the process. At this stage, the weights and biases are random values, and the neural network is not trained however this will allow for communication between the game and the neural network which must be done before it can be trained.

The majority of stakeholders said that it was very important that the A.I. can reach the end of the track consistently so when choosing which inputs/outputs the neural network has access to in the game, inputs and outputs which would make the A.I. more likely to reach the finish line should be prioritized.

Test number	Required inputs	Test description	Success criteria	Pass/Fail
1	Run the program, press "start racing" on the main menu.	The car should move randomly. This is as the neural network now controls the car's inputs however it has not been trained so it makes random decisions.	Once the gameplay begins, the car randomly moves.	
2	Before running the program, add some code which prints the input data to the neural network from the game. Then	Analyse each variable and measure if each value is correct	Each variable is correct which is passed into the neural network.	

	run the program, press “start racing” on the main menu.	as it is passed into the neural network.		
3	Before running the program, add some code which prints all of the outputs from the neural network to the game.	Ensure all values are between 0 and 1.	All probabilities are between 0 and 1.	
4	Before running the program, add some code which prints all of the outputs from the neural network to the game.	Ensure the probability values of each car input vary each frame.	The probability values vary.	

Stage 5- training the neural network:

Features to implement:

- The error that the neural network makes from what would be optimal is calculated
- Calculus and mathematics is used to determine how much the error would vary by changing each weight
- The weights are subtracted by a fraction of this amount to put them closer towards their optimal values
- The neural network is trained

After the neural network is completed, it can now be trained on the game which means this stage had to be the final essential stage. This will result in the neural net being able to control the car much more effectively which means that it will be able to race around the racetrack and complete laps instead of moving randomly.

Here, the neural network calculates the error it has made from the optimal prediction by using a cost function. Then, calculus and partial derivatives are used to determine the differential of the error with respect to the weight which means how much the error would change if the weight was increased by 1. This value multiplied by a learning rate (which is a constant value used to decide the speed the neural network trains at as well as the precision) is subtracted from the weight so that the weight is now slightly closer to the optimal value it can be. Then, this is repeated many times so that eventually the weights become very close to their optimal values, and the neural network can play the game effectively.

A large amount of stakeholders said that it is important that the A.I. can drive quickly so the speed of the A.I. will be considered when deciding the cost function so that the A.I. drives quickly and consistently.

Test number	Required inputs	Test description	Success criteria	Pass/Fail
1	Run the program, press "start racing" on the main menu. Observe if the car approaches the finish line. Before running the program again, add some code which runs a training subroutine in the neural network to train it inside of 10000 racing games. Run the program, press "start racing" on the main menu. Observe if the car approaches the finish line.	The car should not approach the finish line before training but after training the neural network it should.	After training, the neural network's ability to reach the finish line improves.	
2	Add some code to run the training subroutine in the neural network a single time and print the values of the error calculated by the neural network and then each of the derivatives calculated.	Compare the values calculated by the program with values calculated by hand to confirm that the neural network's error calculations are correct.	The neural network correctly calculates how much each weight should be updated by due to the calculated error.	

Stage 6- implementing additional features:

Features to implement:

- More advanced physics
- The game mode to play against the Ai
- Multiple different racetracks to be selected
- The ability to select Ai's with different amounts of training

After all the essential features have been completed, the features identified as important or optional can be implemented. As this stage only contains non-essential features, it would be acceptable for this stage to be only partially completed considering the scope of the project which is why this stage will be developed last.

In this stage, the car's movement is updated to allow for it to "drift" which means that when the player turns the car when it is at a high speed, the back of the car will turn faster than the front of the car. Additionally, more resistive forces will be considered apart from friction such as the effects of wind on the car.

A game mode to play against the Ai will also be included inside of the settings menu. This mode will mean that two cars will be created and displayed on the screen when the gameplay begins. One of the cars will remain the same and will be controlled by the neural network however the other car will be controlled by the user and both cars will compete to reach the finish line first. This will also cause two separate lap values to be displayed during gameplay to show which lap each of the cars are on and the race finish screen will display a different time for each car.

In the settings menu, different racetracks will be able to be selected. Changing this setting will mean that when the gameplay starts, there will be a different background image which will display a different racetrack and the on track and off track sections of the screen that the car can drive on will be changed to correspond with the background image.

In the settings menu, there will be the option to select different Ais to race and control the car which will each have had different amounts of training which means that the ones with little training will likely perform worse than ones with high amounts of training.

A large amount of stakeholders said that it is important that the A.I. is able to drive quickly so the speed of the A.I. will be taken into account when deciding the cost function so that the A.I. drives quickly and consistently.

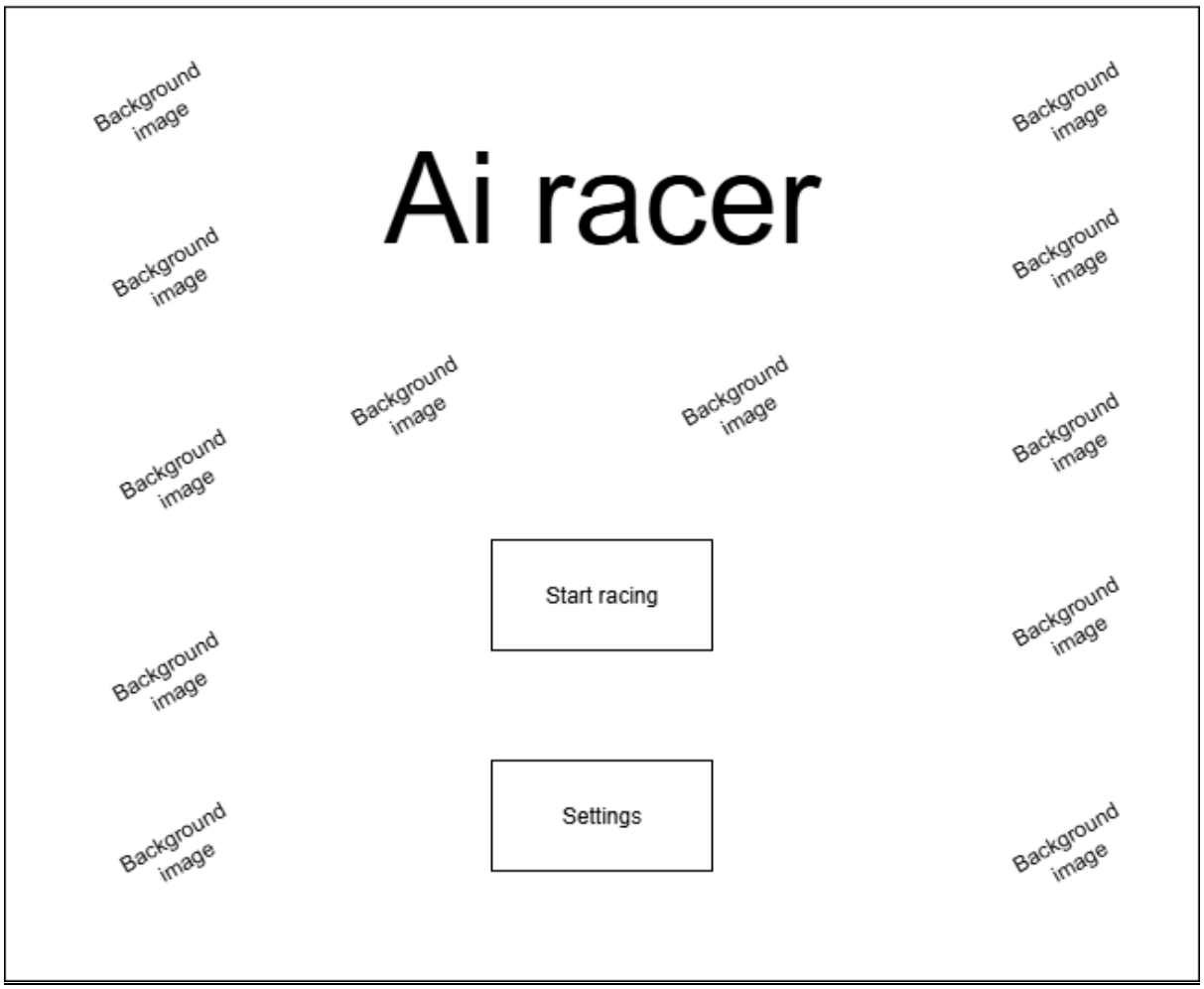
Test number	Required inputs	Test description	Success criteria	Pass/Fail
1	Run the program, press “start racing” on the main menu.	The car should be observed until it reaches a high velocity and attempts to turn. The back of the car should turn faster at higher velocities.	The back of the car turns faster than the front at higher velocities.	
2	Before running the code, ensure that there is a high wind value. Run the program, press “start racing” on the main menu.	The wind should decrease the velocity of the car if the car heads against it and it should increase the velocity of the car if the car heads with it.	The car’s velocity decreases when driving towards the wind and the velocity increases when driving away from it.	
3	Before running the code, ensure that there is a high wind value. Run the program, press “start racing” on the main menu.	There should be a visual indication on the screen that there is wind.	There is a visual indication on screen that there is wind.	

4.	Before running the code, ensure that there is a high wind value. Run the program, press "start racing" on the main menu.	There should be an indication on screen of the direction where it is blowing	There is a visual indication on screen of the direction which the wind is blowing.	
5	Run the program, press "settings" on the main menu. Change the "game mode" setting to be "Vs Ai". Return to the main menu and press start racing.	There should be two cars displayed on the screen when this mode is selected.	Two cars are displayed on screen.	
6	Run the program, press "settings" on the main menu. Change the "game mode" setting to be "Vs Ai". Return to the main menu and press start racing.	One of the cars should be controlled by the neural network and should move on its own whereas the other should be controlled by the player.	One car moves on its own while the other car does not move.	
7	Run the program, press "settings" on the main menu. Change the "game mode" setting to be "Vs Ai". Return to the main menu and press start racing. Hold W for 3 seconds.	The car which is controlled by the player should move forwards when W is pressed as it takes inputs from the user instead of the neural network.	The user's car moves forward when W is pressed.	
8	Run the program, press "settings" on the main menu. Change the "Track" setting to be a different one than normal. Return to the main menu and press start racing.	The background image should be changed to a different one.	The background image will be different to the usual background.	
9	Run the program, press "settings" on the main menu. Change the "Track" setting to be a different one than normal. Return to the main menu and press start racing. Then drive the car over the off-track section displayed.	The car's friction should be lower on the newly displayed track and higher on the newly displayed off-track.	The car's friction is lower on the track and higher on off-track.	
10	Run the program, press "settings" on the main menu. Change the "Track" setting to be a different one than normal. Return to the main menu and press start racing. Then drive around the track and reach the finish line.	The lap counter should increase by 1. This is because the newly displayed finish line on screen should be the position which increases	The lap counter increases by 1.	

		the lap counter in game.		
11	Run the program, press “settings” on the main menu. Change the “Ai selection” setting to be the Ai with the least amount of training. Then, return to the main menu and press start racing. Then watch the neural network drive around the track and record the time taken by the Ai once the race is over. Then, return to the settings menu and change the Ai to the one with the highest amount of training and return to the main menu. Next, press start racing and record the time that it takes for this Ai to complete the race.	The neural network with little training should perform worse than the neural network with lots of training so their times should be compared. This test should be completed at least 5 times in case some results differ from the average time for each network.	The Ai with little training should have a higher time taken than the Ai with lots of training.	

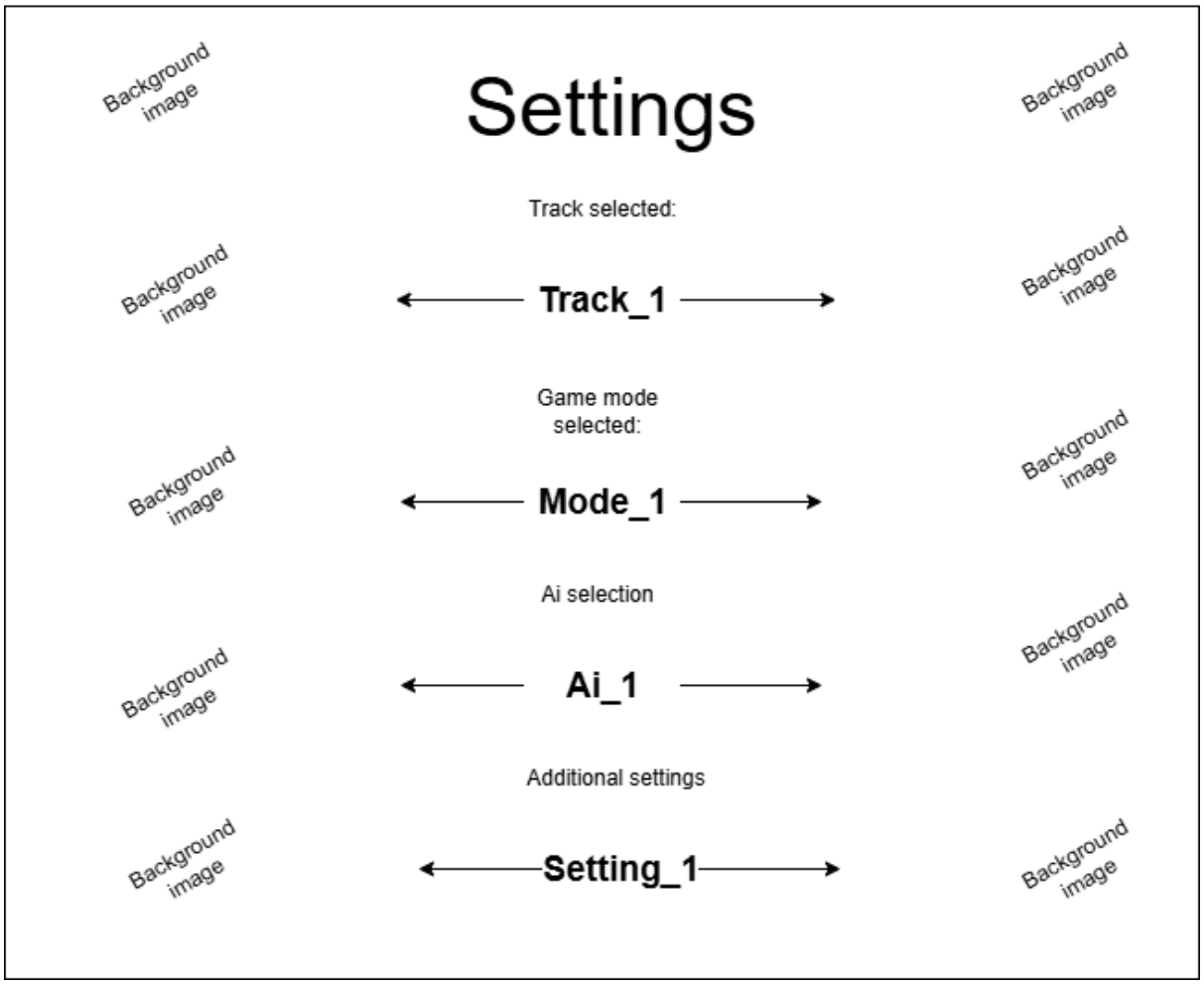
Gui design:

Main menu

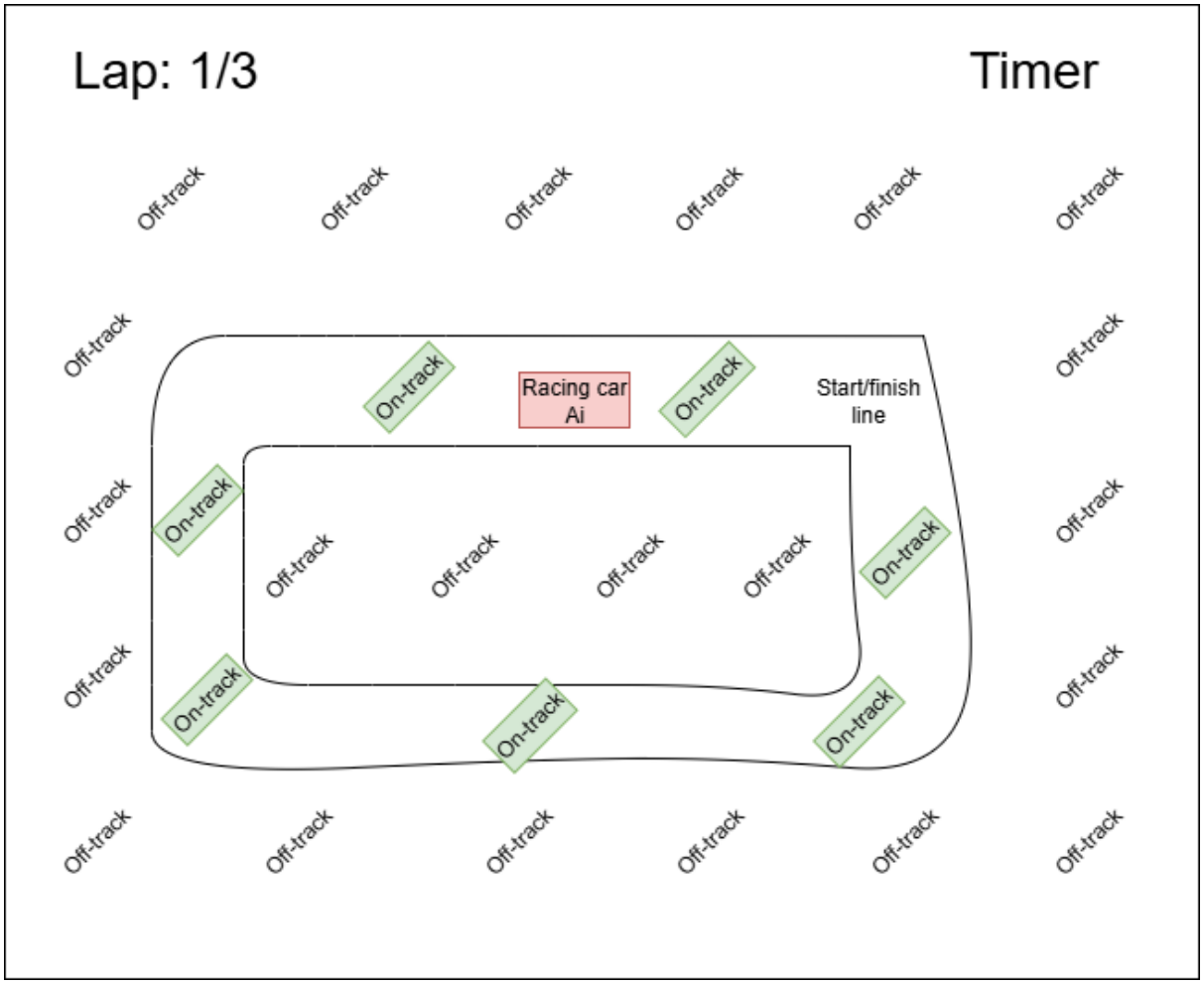


This is the main menu of the program which will be the first screen that the user is met with when running the program. The “Start Racing” button will change the screen to display the car and the track and the racing game itself will begin using the currently applied settings. The “Settings” button will change the screen to the settings menu which will change certain functionality and accessibility features in the game for the user’s requirements. The background image will be a sample image of the racing game.

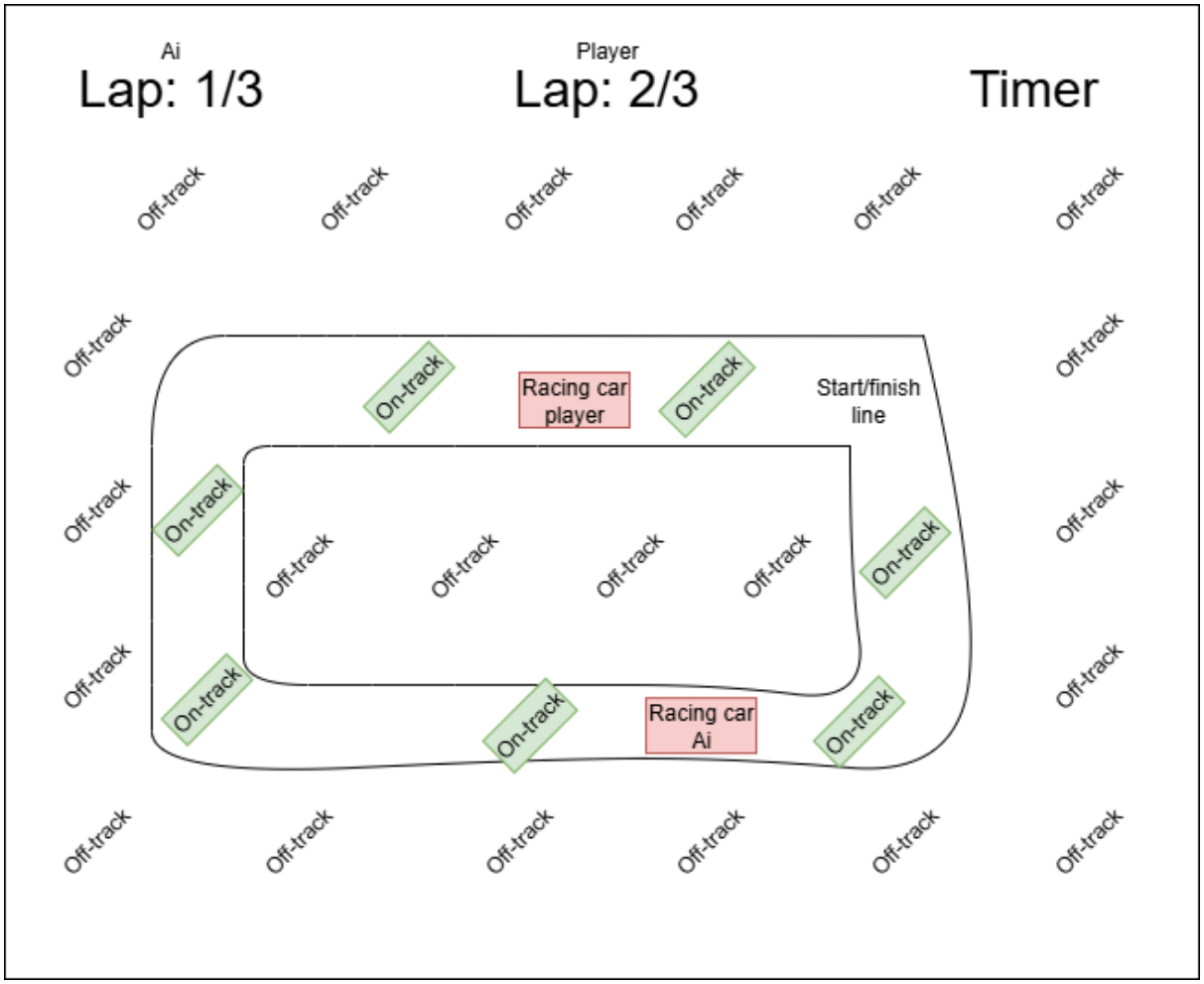
Settings



This is the settings menu which will be displayed after the user presses the settings button. When the arrows are clicked on each setting, the setting will switch to the next value it can be set as. The track selected setting will change the layout and image of the racetrack when the gameplay begins. The game mode setting will allow the user to choose between the neural network driving by itself or the player racing against the neural network. The Ai selection setting will allow the user to chose to use different neural networks with different amounts of time spent training. The additional settings will contain settings to modify other things such as a high contrast colour mode to assist with colour blind users and other accessibility settings.



This is the gameplay screen which is accessed by pressing the “Start racing” button. The timer begins from 0s and counts up as the game progresses. The lap counter displays the current lap. The “on-track” sections of the screen are which would be classed as the track in this example which are enclosed by the two track edges. The “off-track” sections of the screen are the areas which would be classed as off the track in this example as they are not between the two edges of the track. The start/finish line represents where the start/finish line could be which is where the race starts and ends. Instead of text, the off-track and on-track sections will be displayed graphically with the track likely having a colour such as black which is typical of roads and racetracks whereas the off-track sections will be made up of colours such as green and brown to represent grass and mud which will allow make the user understand which sections of the screen are on and off the track. Additionally, a setting will be added for colour blind users which will use a similar labelling system as used above to differentiate between the on and off-track sections. The racing car Ai also represents the neural network’s car which will drive on the track. This gameplay screen is only accessible if the game mode setting is set as Ai only. If the play against the AI mode is selected, the following screen will be displayed instead:



This is very similar to the other gameplay screen, however there is an additional racing car for the player to control on the track and there are separate lap counters for the player and the Ai as they are able to be on different laps at the same time.

The race finish screen will be displayed once all laps have been completed in a race. The “Time:” section will display the total time it took for the Ai to complete all of the laps. The new race button will begin a new race with the same settings already applied whereas the main menu button will display the main menu screen. This is the race

finish screen when the Ai races by itself. The player vs Ai race finish screen is here:

This screen is very similar to the Ai only screen however it displays a different time for the player and the Ai which displays the time it took for each of them to finish the race individually.

Development:

Stage 1- creating a controllable car:

Now that the program design is complete, the iterative development of the program can begin. Firstly, a display window will be created. To do this, I will be using pygame so, I watched and followed this pygame tutorial (Pygame tutorial) to familiarize myself with the basic functionality of pygame. After this, a basic car image will be displayed onto the screen. Then, user input will be implemented which will mean that if W is pressed, then the speed of the car will increase going forwards and if S is pressed then this will increase the speed of the car going backwards. Additionally, if A is pressed, then the car

will rotate anticlockwise and if D is pressed then the car will rotate clockwise. It is important to ensure that the car's position on screen is updated every time it is moved and if the car rotates a certain angle, then it will move forwards and backwards parallel to the direction of the car. I will also need to ensure that the car is unable to continue driving beyond the edge of the screen so that it remains on the race track.

The features that will be implemented in this stage are:

- A display window to contain the game
- A car which speeds up when up is pressed and slows down when down is pressed
- The car can also turn right when right is pressed and turn left when left is pressed
- Create boundaries at the edge of the display window so that the car cannot leave the screen

Class diagram:

Car
-XPos: int -YPos: int -XSpeed : int -YSpeed: int -ResultantSpeed: float -Rotation: int
+__init__(int XPos,int YPos,int Rotation): void +get_XPos(): int +get_YPos(): int +get_XSpeed(): int +get_YSpeed(): int +get_ResultantSpeed(): float +get_rotation(): int +set_speed(int ResultantSpeed): void +move_car(): void +rotate_car(Angle): void +display_car(): void

This is a class diagram for the "Car" class which will contain all the necessary methods and attributes for the car in the game to be displayed on screen and move. The __init__ method is the constructor for the class in python which I learnt how to implement in this python tutorial: Python for Beginners – Full Course [Programming Tutorial]. The class will store the x position and the y position of the car as attributes which will correspond to the display window in pygame. It will also store the x and y speeds of the car which will be used to move the x and y position attributes. Finally, it will store the rotation of the car which will be used to display the image of the car at an angle and will be used outside of this class to determine how much of the overall velocity will be used as x speed and y speed.

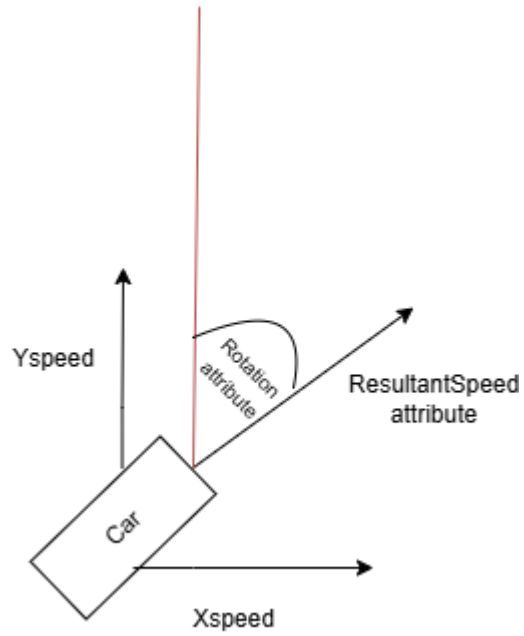
The __init__ method is the constructor of the class and it will initialise the x, y and resultant speeds as 0 and set the x and y positions as well as the rotation to arguments passed in. The get_ methods will return the corresponding attribute e.g. get_XPos() will return the value of the XPos attribute. The set_speed method will take in the resultant speed of the car and then calculate the X and Y speeds using the rotation of the car. The move_car method will add the XPos attribute to the XSpeed attribute and set this to the XPos and will do the same for the YPos and YSpeed which will happen every frame so that the car moves smoothly and naturally. If the XSpeed or YSpeed values are negative, then this will decrease the x and y positions of the car as well. The rotate_car method will take an angle in degrees as a parameter and will use the pygame "rotate()" function to rotate the image of the car by that amount. This value will also be added to the rotation attribute and can be negative.

I decided to use an object orientated class for the car as in the final stage of development a second car will be created to race against the first one and it is easy to instantiate two objects from the same class in python and both cars will require the exact same methods and attributes. The alternative option was to store all the attributes and methods from the car as variables and subroutines outside of a class and either create duplicate attributes for each car or store each attribute as an array of values which stores each value for each car however this would be more time consuming and difficult to work with and would introduce additional array handling or many repeated lines of code so creating an object orientated class was the best decision.

I decided to use snake case for the method names as this is the standard in python, and I decided to use Pascal case for the attributes so that the attributes were visually distinct from the methods meaning the code will be easier to read.

Algorithm design:

As the set_speed method involves a complicated calculation to calculate the X and Y speeds from the rotation and the resultant speed, I decided to design the method in detail before coding.



I created a diagram to visualize how some of the different attributes stored in the car class will act in the game.

The ResultantSpeed attribute will be directly affected by the inputs from the player or the neural network. This is because if there is a “W” input then the resultant speed attribute will increase and if there is an “S” input then it will decrease. The rotation attribute will store the angle that the car is rotated at clockwise which is shown as the angle from the red line on the diagram. As the car can only be moved pixel values in the X and Y directions, trigonometry will be used to calculate the XSpeed and YSpeed that will update the X and Y positions of the car which are stored as XPos and YPos.

I listed the calculations required to find the X and Y speeds in the table below which I deduced using my diagram and trigonometry:

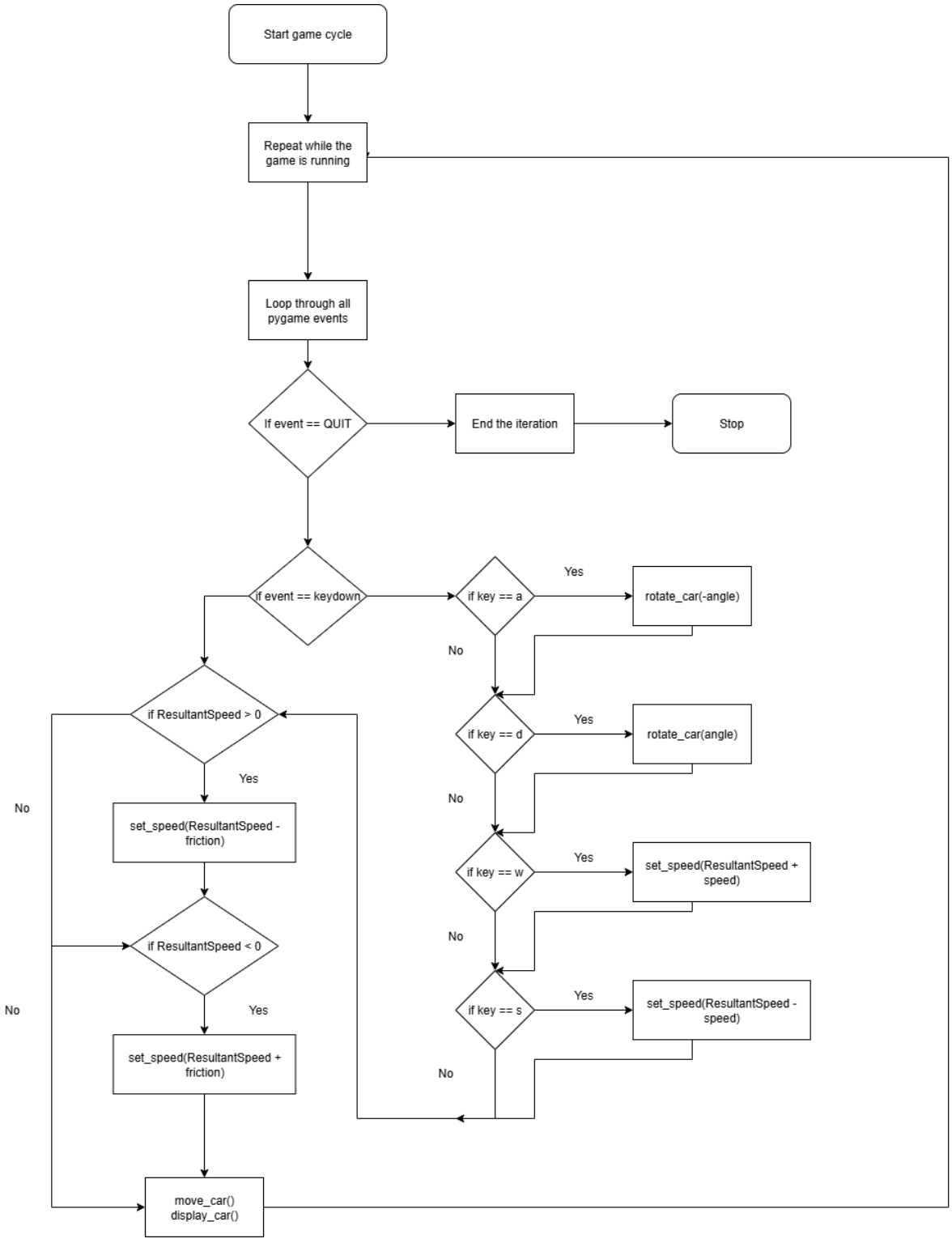
Rotation	YSpeed calculation	XSpeed calculation
Between 0 and 90	sin(90-Rotation) * ResultantSpeed	cos(90-Rotation) * ResultantSpeed
Between 90 and 180	sin(Rotation - 90) * ResultantSpeed	cos(Rotation - 90) * ResultantSpeed
Between 180 and 270	sin(270-Rotation) * ResultantSpeed	cos(270-Rotation) * ResultantSpeed
Between 270 and 360	sin(Rotation-270) * ResultantSpeed	cos(Rotation-270) * ResultantSpeed

Here is the final pseudocode I created using this for the set_speed method:

```
FUNCTION set_speed(ResultantSpeed)
        this.ResultantSpeed = ResultantSpeed
        IF 0 ≤ Rotation AND Rotation < 90 THEN
                XSpeed = cos(90-Rotation) * ResultantSpeed
                YSpeed = sin(90-Rotation) * ResultantSpeed
        ELSE IF 90 ≤ Rotation AND Rotation < 180 THEN
                XSpeed = cos(Rotation - 90) * ResultantSpeed
                YSpeed = sin(Rotation - 90) * ResultantSpeed
        ELSE IF 180 ≤ Rotation AND Rotation < 270 THEN
                XSpeed = cos(270-Rotation) * ResultantSpeed
                YSpeed = sin(270-Rotation) * ResultantSpeed
        ELSE THEN
                XSpeed = cos(270-Rotation) * ResultantSpeed
                YSpeed = sin(270-Rotation) * ResultantSpeed
        ENDIF
END FUNCTION
```

In the beginning of the set_speed method, I update the ResultantSpeed attribute inside of the car class into the new resultant speed which is passed into the method as a parameter. I decided to use if statements instead of a switch statement as python does not have inbuilt switch case functionality so when I develop this method in python I will need to use if statements. Here, I used the table that I created to determine which trigonometric function multiplied by the resultant speed should be used for the X speeds and Y speeds for different angles.

The flowchart for the “game cycle”



As this game will continuously run until the user decides to quit the program, a loop will repeatedly iterate so that the program cannot end which I learn how to do from the pygame tutorial. Inside of the loop contains the operations which must be performed every frame while the program is running for stage 1. Here, the program loops through and checks every event which occurred since the last frame in the program. If this event was the user attempting to quit, then the iteration will end and pygame will quit. If the event was one of either w,a,s or d being pressed then a certain operation occurs because of it. If the user pressed A, then the rotate_car method is called from the car class which will rotate the car anticlockwise. If the user pressed D, then the rotate_car method will be called to rotate the car clockwise. If W was pressed then the set_speed method will be called to increase the resultant speed of the car. If S was pressed, then the set_speed method will be called to decrease the resultant speed of the car. Also each frame, if the resultant speed of the car is positive then a friction value will be taken away from it and if the resultant speed is negative then a friction value will be added to it to take the resultant speed closer to 0. Finally, the move_car and display_car methods are called to move the car depending on its speed and to display the car on screen.

Data dictionary:

Here is the data dictionary for variables which are not encapsulated inside of classes for this stage. Note that I modified the data dictionary throughout Stage 1 as new variables/attributes were needed.

Name	Data type	Description	Validation
screen	Surface (pygame)	This is the display window which will display the entire program to the user.	None
CarImage	Surface (pygame)	This is the image of the car which will be stored and used by the DisplayImage. The CarImage should never be modified in the program and should reflect the original car image.	Must store an image file
DisplayCarImage	Surface (pygame)	This is the image of the car which will be displayed to the user and will be modified by	Must store an image file

		transformations such as rotations.	
running	Boolean	This is a boolean value which will be true when the program is running continuously and will be false when the user is quitting the program.	None
IsGoingUp	Boolean	Stores whether the user is holding down the W key or not	None
IsGoingDown	Boolean	Stores whether the user is holding down the S key or not	None
IsTurningLeft	Boolean	Stores whether the user is holding down the A key or not	None
IsTurningRight	Boolean	Stores whether the user is holding down the D key or not	None

Here is the data dictionary for attributes which are encapsulated inside of classes for this stage.

Name	Data type	Description	Validation	In class
XPos	int	The x position of the car in pixels	Must be between 0 and the maximum horizontal pixel value of the screen	Car
YPos	int	The y position of the car in pixels	Must be between 0 and the maximum vertical pixel value of the screen	Car
XSpeed	int	The x speed of the car which is how much the	None	Car

		XPos will be added to each frame.		
YSpeed	int	The y speed of the car which is how much the YPos will be added to each frame.	None	Car
Rotation	int	The clockwise angle which the car is rotated at	Must be between 0 and 360	Car
ResultantSpeed	float	The overall speed of the car which is affected by user or neural network inputs. This speed is then resolved into x and y speeds.	None	Car

Programming:

After completing the iterative design for stage 1, the programming of the stage can now begin.

```
1   import pygame
2
3   pygame.init() #Initialises pygame so its functionality can be used
4
5   screen = pygame.display.set_mode((800, 600)) #Creates a display window with 800 horizontal pixels and 600 vertical pixels
6
7
8   running = True
9   while running: #Infinite loop to prevent the display window from closing until the user decides to
10      for event in pygame.event.get():
11          if event.type == pygame.QUIT:
12              running = False
13
14  pygame.quit()
15
```

Initially, I set up a display window using pygame. The “screen” variable holds the display window created using pygame which currently has 800 horizontal pixels and 600 vertical pixels, however I will likely change this value after implementing the car after determining which screen size is most suitable to work with the car. The while loop ensures that the program does not end without the user choosing to quit as the player would have a better experience if they can choose to quit after running the program for any amount of time. Line 10 loops through all of the pygame events which occurred in

the last frame such as player inputs and Line 11 and 12 decide that if the event is the user pressing the quit button on the display window, then the program will end.

```
8   class Car:
9
10      def __init__(self,XPos,YPos,Rotation):
11          self.XPos = XPos
12          self.YPos = YPos
13          self.Rotation = Rotation
14          self.XSpeed = 0
15          self.YSpeed = 0
16          self.ResultantSpeed = 0
17          pass
18
19
20
```

Next, I began creating the car class. I chose to implement this early in the stage as most of the features of this stage depend heavily on the car class's methods and attributes. Here, I created the constructor for the car class where the initial XPos, YPos and Rotation of the car are set to passed in parameters and all of the speeds of the car are initialised to 0. I chose to initialise the attributes in this way as when the car(s) are initially placed on the racetrack, they may begin in different positions and at different rotations depending on which track was selected. However, at the beginning of each game the car's speeds will always be 0 so this can be determined.

```
19  #Getters and setters
20      def get_XPos(self): #Getter for the X position of the car
21          return XPos
22
23      def get_YPos(self): #Getter for the Y position of the car
24          return YPos
25
26      def get_XSpeed(self): #Getter for the X speed of the car
27          return XSpeed
28
29      def get_YSpeed(self): #Getter for the Y speed of the car
30          return YSpeed
31
32      def get_ResultantSpeed(self): #Getter for the resultant speed of the car
33          return ResultantSpeed
34
35      def get_Rotation(self): #Getter for the rotation of the car
36          return Rotation
37
```

Next, I created all of the getters for the class which were explained in the class diagram.

```
38
39      def set_speed(self,ResultantSpeed): #This method takes in a new resultant speed as a parameter and updates the ResultantSpeed attribute and then calulates the correct
40          self.ResultantSpeed = ResultantSpeed
41          if self.Rotation < 90:
42              self.XSpeed = np.cos(90-self.Rotation) * ResultantSpeed
43              self.YSpeed = np.sin(90-self.Rotation) * ResultantSpeed
44
45          elif self.Rotation < 180:
46              self.XSpeed = np.cos(self.Rotation-90) * ResultantSpeed
47              self.YSpeed = np.sin(self.Rotation-90) * ResultantSpeed
48
49          elif self.Rotation < 270:
50              self.XSpeed = np.cos(270-self.Rotation) * ResultantSpeed
51              self.YSpeed = np.sin(270-self.Rotation) * ResultantSpeed
52
53          else:
54              self.XSpeed = np.cos(self.Rotation-270) * ResultantSpeed
55              self.YSpeed = np.sin(self.Rotation-270) * ResultantSpeed
56
```

Here, I created the set_speed method which follows the previous design. The if statements from the pseudocode could be simplified by using elifs because for the range 0≤Rotation<90, The rotation will be validated to always be positive so the 0 is not necessary and for the range 90≤Rotation<180, as the range 0 to 90 has already been checked only the rotation being less than 180 needed to be specified. Inside each if statement follows the pseudocode for the method where the correct trigonometric functions are applied to calculate the X and Y speeds from the rotation and the resultant speed.

```
57
58          def move_car(self):
59              self.XPos += self.XSpeed
60              self.YPos += self.YSpeed
61
62          def rotate_car(self,angle):
63              self.Rotation += angle
64              if Rotation >360:
65                  Rotation -= 360
66              pygame.transform.rotate("Car temp",angle * -1)
67
68          def display_car(self):
69              screen.blit("Car temp",(self.XPos,self.YPos))
70
```

Next I created the move_car, rotate_car and display_car methods. The move_car method will be called once every frame and it will add the X and Y speeds to the X and Y positions of the car to move them. The rotate_car method will add an angle which is passed in as a parameter to the Rotation attribute. It will then rotate the image of the car by that angle multiplied by -1 as the pygame rotate function rotates the image anticlockwise whereas my rotation attribute stores the clockwise rotation. Finally, I created the display_car procedure will which display the image of the car in the current X and Y positions. I created a temporary image of the car named car temp which is a pink rectangle which will be used to test that the code involving the car works correctly

and then more accurate car images will be used in the future when I implement the final images to be used in the game. This ensures that in the early stages of development, all of my focus can be spent on ensuring that the program functions correctly which will speed up the development overall instead of splitting my focus over both functionality and aesthetics.

```
screen = pygame.display.set_mode((800, 600)) #Creates a display window with 800 horizontal pixels and 600 vertical pixels
Car1 = Car(100,100,0)


running = True
while running: #Infinite loop to prevent the display window from closing until the user decides to

    for event in pygame.event.get(): #event handling
        if event.type == pygame.QUIT:
            running = False
        if event.type == pygame.KEYDOWN: # This means any key has been pressed
            if event.key == pygame.K_a: #This means it was the a key
                Car1.rotate_car(-2)

            if event.key == pygame.K_d: #This means it was the d key
                Car1.rotate_car(2)

            if event.key == pygame.K_w: #This means it was the w key
                Car1.set_speed(Car1.get_ResultantSpeed() + 2)

            if event.key == pygame.K_s: #This means it was the s key
                Car1.set_speed(Car1.get_ResultantSpeed() - 2)


        #end event handling
    if Car1.get_ResultantSpeed() > 0:
        Car1.set_speed(Car1.get_ResultantSpeed() - 1)
    elif Car1.get_ResultantSpeed() < 0:
        Car1.set_speed(Car1.get_ResultantSpeed() + 1)


    screen.fill((0,0,0))
    Car1.move_car()
    Car1.display_car()
    pygame.display.update()
```

Next, I followed the flowchart for this stage to get user input which was included inside the while loop which repeats until the user quits. Firstly, a new car object called Car1 is instantiated at 100 pixels horizontal and 100 pixels vertical. Next, if the player presses the a key, then the car will rotate anticlockwise by 2 and if the player presses the d key then the car will rotate clockwise by 2. I chose the value 2 temporarily but when running the code, I will determine the correct value to rotate the car by. If the player presses the w key, the car's set_speed method will set the resultant speed 2 greater than it currently is. If the player presses the s key, the car's set_speed method will set the resultant speed 2 less than it currently is. Again, I picked the 2 value randomly and this will be updated after testing to ensure that the movement of the car feels as natural as possible as this is important to stakeholders. After the events are handled, a frictional force will be applied to the car where if the car's resultant speed is positive then it will be reduced by 1, and if it is negative then it will increase by 1 which ensures that the car will slow down when the player stops pressing any keys.

After implementing the code, I attempted to run the program, however I was met with this error message:

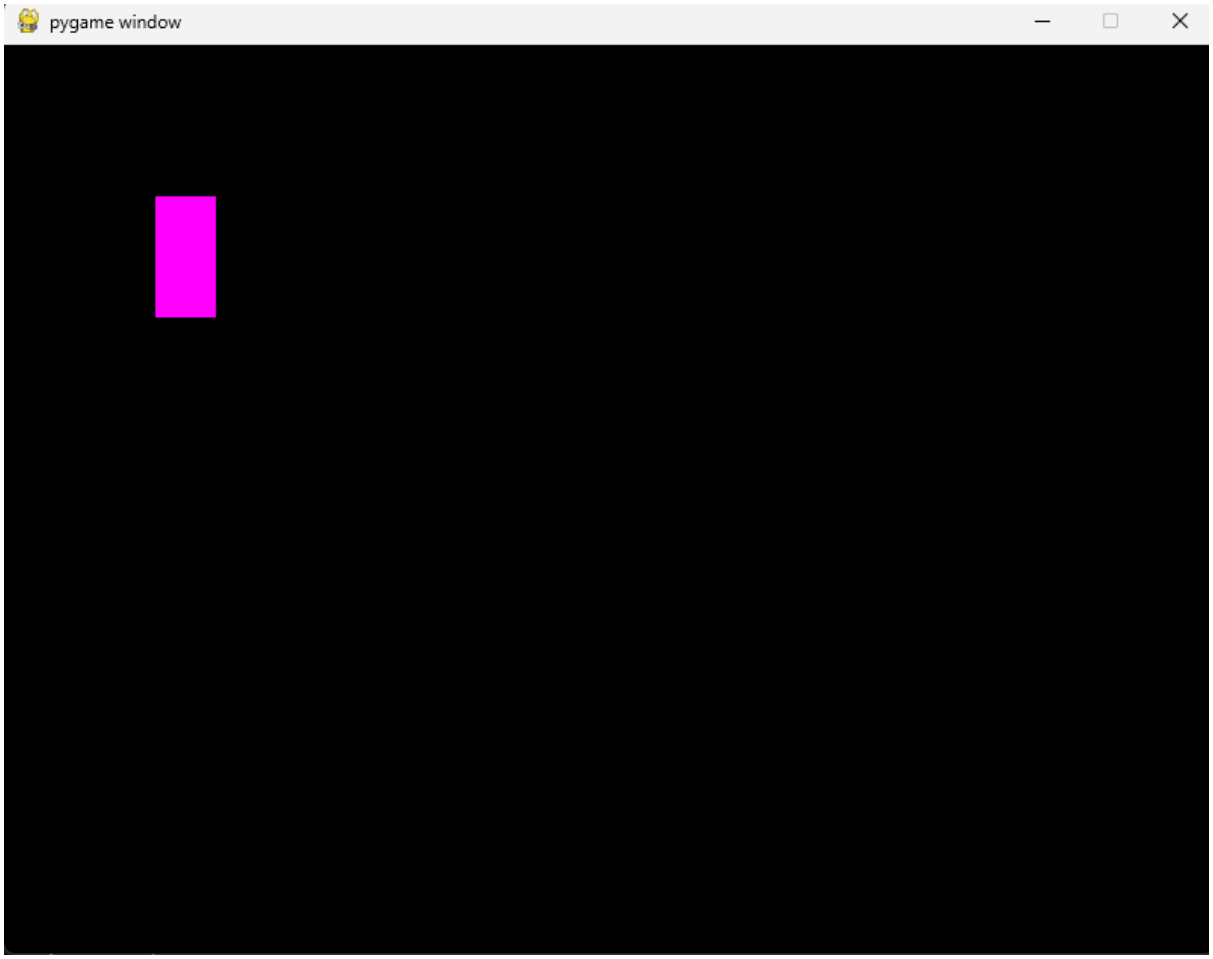
```
    screen.blit("Car temp",(self.XPos,self.YPos))
TypeError: argument 1 must be pygame.surface.Surface, not str
```

I then realised than “Car temp” is the name of the image file I created for the car however I should have specified that the file name is “Car temp.png” instead. To fix this I created a variable to store the image using pygame’s surface data type and I referenced the variable name instead of the image name as follows:

```
    CarImage = pygame.image.load('Car temp.png')
```

```
def display_car(self):
    screen.blit(self.CarImage1,(self.XPos,self.YPos))
```

Next, the code executed, and a display window was created which contained the correct car image at the correct place on the screen:



However, I noticed two errors with the handling of player inputs. Firstly, when pressing the a or d keys, the image of the car did not rotate. Secondly, when w or s is pressed, the car did move correctly however it did not move continuously as the key is held down as I planned instead it moved the instant the key is pressed and then no more as the key is held down.

To fix the first issue, I looked again at the pygame documentation for rotating images (pygame.transform — pygame v2.6.0 documentation). I then realised that the rotate function does not apply the rotation to the image itself but instead returns a rotated image. To fix this, I modified the rotate_car method as follows:

```
def rotate_car(self,angle):
    self.Rotation += angle
    if self.Rotation >360:
        self.Rotation -= 360
    CarImage = pygame.transform.rotate(CarImage,angle * -1)
```

Where I set the CarImage equal to the rotation. However, when I ran this code and pressed a to rotate the car I was met with this error:

```
    CarImage = pygame.transform.rotate(CarImage,angle * -1)
                                       ^^^^^^^^
UnboundLocalError: cannot access local variable 'CarImage' where it is not associated with a value
```

I noticed that CarImage was described as a local variable to the rotate_car method when it actually is a global variable. This meant that instead of the method accessing the real CarImage, it instead attempted to work with its own version so to fix this I passed the CarImage into the rotate_car method as a parameter.

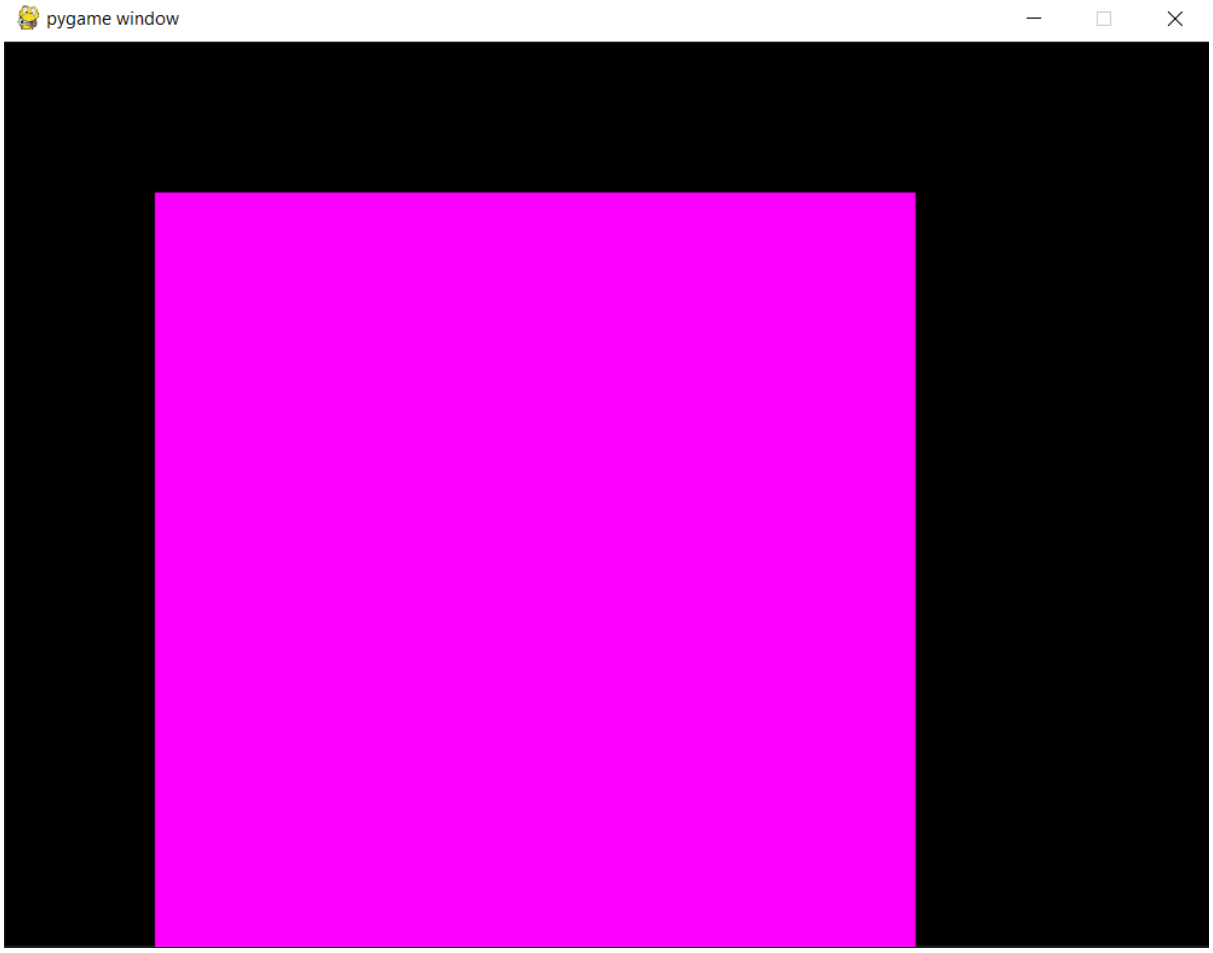
```
62      def rotate_car(self,angle, theCarImage):
63          self.Rotation += angle
64          if self.Rotation >360:
65              self.Rotation -= 360
66          theCarImage = pygame.transform.rotate(theCarImage,angle * -1)
67          return theCarImage
68
```

As the passed in car image is now local to the method, it has to be returned as well.

```
if event.type == pygame.KEYDOWN: # This means any key has been pressed
    if event.key == pygame.K_a: #This means it was the a key
        CarImage = Car1.rotate_car(-2, CarImage)

    if event.key == pygame.K_d: #This means it was the d key
        CarImage = Car1.rotate_car(2, CarImage)
```

I also changed the player input handling so that the result of the rotate_car method was set to the CarImage. However, instead of the image rotating it grew every time either a or d was pressed. After reading the pygame documentation further, I realised that what was happening was that when the image was being rotated, the image was padded larger to hold the new size however the new pixels which are introduced are unable to be transparent as the image does not have pixel alphas, so the new pixels matched the colour of the top-left pixel which was pink. This meant that instead of the image rotating, it was padded larger and then given additional pink pixels to fill the space. As this is done to the same image, every time a or d was pressed it grew even larger. Here is the result of pressing a and d many times:



To fix this, Firstly I decided to fix the issue where each time the image is padded, the padded image is reused and padded again.

```
5    CarImage = pygame.image.load('Car temp.png')
6    DisplayCarImage = CarImage
```

```
89              if event.key == pygame.K_a: #This means it was the a key
90                  DisplayCarImage = Car1.rotate_car(-2, CarImage)
91
92              if event.key == pygame.K_d: #This means it was the d key
93                  DisplayCarImage = Car1.rotate_car(2, CarImage)
94
```

```
70      def display_car(self):
71          screen.blit(DisplayCarImage,(self.XPos,self.YPos))
72
```

```
62      def rotate_car(self,angle, theCarImage):
63          self.Rotation += angle
64          if self.Rotation >360:
65              self.Rotation -= 360
66          theCarImage = pygame.transform.rotate(theCarImage,(self.Rotation) * -1)
67          return theCarImage
```

I created the variable DisplayCarImage which will be the image which is displayed. Initially DisplayCarImage is the same as the CarImage however if the car is rotated then the original CarImage is rotated and set equal to DisplayCarImage. This means that the original car image is never modified so the image padding which occurs will only happen a single time before the image is displayed. In the rotate_car method, I changed the angle the image is rotated by from the angle passed in as a parameter to the entire rotation variable. This is because the image which is passed into this method is CarImage which is not rotated at all which means the entire rotation must be applied to get the new image.

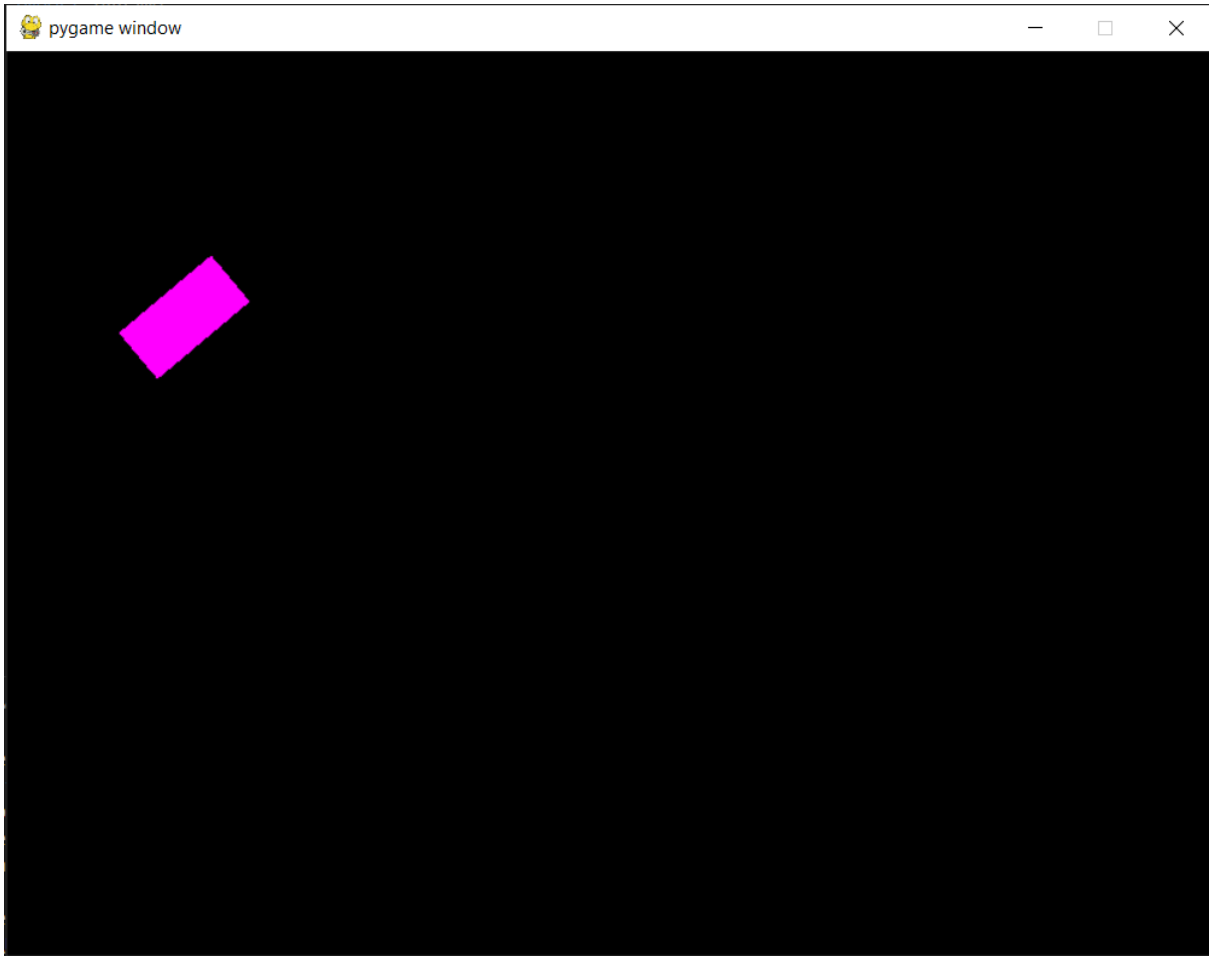
Next, I read the pygame documentation (pygame.Surface — pygame v2.6.0 documentation) to search for a way to add pixel alphas to an image so the padded pixels would be transparent. To do this, the method pygame.Surface.convert_alpha() can be used. To implement this, I wrote the following code:

```
4   pygame.init() #Initialises pygame so its functionality can be used
5   CarImage = pygame.image.load('Car temp.png') #Sets the Surface object CarImage equal to Car temp.png
6   CarImage = pygame.Surface.convert_alpha(CarImage) #Converts that image so that it can contain pixel alphas
7   DisplayCarImage = CarImage
8   """ Class definitions"""
9
```

This meant that the CarImage and DisplayCarImage now supports pixel alphas meaning that the padded pixels would be transparent. However when running the code I encountered this error message:

```
    CarIMAGE = pygame.Surface.convert_alpha(CarImage)
               ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
pygame.error: No video mode has been set
```

I realised that no video mode has been set because I was executing this before the display window had been created, so I moved the line to create the display window before the CarImage is set. After doing this and running the program, the car image rotated correctly with no visible padded pixels:



Next, I fixed the issue where the Car only moved immediately as the key is pressed and not continuously. To do this I created the variables “IsGoingUp”, “IsGoingDown”, “IsTurningLeft” and “IsTurningRight” which will store whether the car is moving up, moving down, turning left or turning right. Then when the corresponding key is pressed for each movement, the variable will be set to true and when the key is let go of (which can be checked using the KEYUP event in pygame) the variable will be set to false. Then every time the while running loop iterates, it will do the corresponding car method if the variable is set to true. Here is the code I wrote to do this:

```
84  running = True
85  IsGoingUp = False
86  IsGoingDown = False
87  IsTurningLeft = False
88  IsTurningRight = False
89  while running: #Infinite loop to prevent the display window from closing until the user decides to
```

```
for event in pygame.event.get(): #event handling
    if event.type == pygame.QUIT:
        running = False
    if event.type == pygame.KEYDOWN: # This means any key has been PRESSED
        if event.key == pygame.K_a: #This means it was the a key
            IsTurningLeft = True

        if event.key == pygame.K_d: #This means it was the d key
            IsTurningRight = True

        if event.key == pygame.K_w: #This means it was the w key
            IsGoingUp = True

        if event.key == pygame.K_s: #This means it was the s key
            IsGoingDown = True
```

```
if event.type == pygame.KEYUP: # This means any key has been LET GO OF
    if event.key == pygame.K_a: #This means it was the a key
        IsTurningLeft = False

    if event.key == pygame.K_d: #This means it was the d key
        IsTurningRight = False

    if event.key == pygame.K_w: #This means it was the w key
        IsGoingUp = False

    if event.key == pygame.K_s: #This means it was the s key
        IsGoingDown = False
```

Running the code now, meant that the car rotated and moved correctly when each button was pressed and it stopped moving when the key was let go of. However, the car moved and rotated much too fast with the current values. After experimenting with different values, I decided upon 0.001 for the amount the speed is increased while pressing the W or S keys, 0.0005 for the friction and 0.2 for the angle the image is rotated by while pressing the A or D keys.

However, when rotating the car, the direction that the car travelled in did not match the direction which the car was facing. To identify the issue, I printed the value stored in the rotation attribute every time the car is displayed:

```
-2.0
-1.5
-1.0
-0.5
0.0
0.5
1.0
1.5
2.0
```

Here, I noticed that the rotation attribute could go into negative values. This was because when I created the rotate_car method I accidently left out the validation for when the rotation is less than 0. To fix this I corrected the code as follows:

```
64      def rotate_car(self,angle, theCarImage):
65          self.Rotation += angle
66          if self.Rotation >360:
67              self.Rotation -= 360
68          if self.Rotation <0:
69              self.Rotation += 360
70          theCarImage = pygame.transform.rotate(theCarImage,(self.Rotation) * -1)
71          return theCarImage
```

Now the validation for the Rotation is correct. However, the movement of the car still did not match the car image. After reading the numpy documentation for its sin and cos functions (numpy.sin — NumPy v2.3 Manual), (numpy.cos — NumPy v2.3 Manual), I realised that those functions treat the argument passed into them as if they were in radians however, I was using degrees in the set_speed method. To fix this, I used the numpy documentation again to find the radians function which converts a degree values into radians (numpy.radians — NumPy v2.3 Manual).

```
41      def set_speed(self,ResultantSpeed): #This method takes in a new resultant speed as a parameter and updates 
42          self.ResultantSpeed = ResultantSpeed
43          if self.Rotation < 90:
44              self.XSpeed = np.cos(np.radians(90-self.Rotation)) * ResultantSpeed
45              self.YSpeed = np.sin(np.radians(90-self.Rotation)) * ResultantSpeed
46
47          elif self.Rotation < 180:
48              self.XSpeed = np.cos(np.radians(self.Rotation-90)) * ResultantSpeed
49              self.YSpeed = np.sin(np.radians(self.Rotation-90)) * ResultantSpeed
50
51          elif self.Rotation < 270:
52              self.XSpeed = np.cos(np.radians(270-self.Rotation)) * ResultantSpeed
53              self.YSpeed = np.sin(np.radians(270-self.Rotation)) * ResultantSpeed
54
55          else:
56              self.XSpeed = np.cos(np.radians(self.Rotation-270)) * ResultantSpeed
57              self.YSpeed = np.sin(np.radians(self.Rotation-270)) * ResultantSpeed
58
```

This is the updated set_speed function with the added conversion to radians inside each sin or cos function. However, when attempting to drive the car with this function, some of the x and y directions seemed to be backwards from what they should be. I

realised that this was because my calculations were finding the positive magnitude of the X and Y speeds, however for the car to be able to turn left or downwards, some of the speeds would be negative. To fix this I changed my code as shown below:

```
def set_speed(self,ResultantSpeed): #This method takes in a new resultant speed as a parameter and updates the ResultantSpeed attribu
    self.ResultantSpeed = ResultantSpeed
    if self.Rotation < 90:
        self.XSpeed = np.cos(np.radians(90-self.Rotation)) * ResultantSpeed
        self.YSpeed = np.sin(np.radians(90-self.Rotation)) * ResultantSpeed

    elif self.Rotation < 180:
        self.XSpeed = np.cos(np.radians(self.Rotation-90)) * ResultantSpeed
        self.YSpeed = -1 * np.sin(np.radians(self.Rotation-90)) * ResultantSpeed

    elif self.Rotation < 270:
        self.XSpeed = -1 * np.cos(np.radians(270-self.Rotation)) * ResultantSpeed
        self.YSpeed = -1 * np.sin(np.radians(270-self.Rotation)) * ResultantSpeed

    else:
        self.XSpeed = -1 * np.cos(np.radians(self.Rotation-270)) * ResultantSpeed
        self.YSpeed = np.sin(np.radians(self.Rotation-270)) * ResultantSpeed
```

This means that for the YSpeed, whenever the angle is between 90 and 270 degrees, the result should be multiplied by -1 as this means the car is facing downwards. For the XSpeed, whenever the angle is between 180 and 360 then the result should be multiplied by -1 as the car is now facing left and the XSpeed is positive in the right direction.

```
def move_car(self):
    self.XPos += self.XSpeed
    self.YPos -= self.YSpeed
```

I updated the move_car method as well. I made it so the Y speed is subtracted from the Yposition of the car. This is because in pygame the y values increase as you go down however I am used to the y values increasing as you go up. Working with YSpeed values which treated YSpeed as if positive Y speed was directed upwards made developing the program easier as I personally am used to positive Y values being directed upwards. This meant that I could develop the trigonometry in the set_speed method without being potentially confused by the Y values being the opposite of what I would expect and simply subtracting this in the move_car method is a very simple thing to include.

Now when I attempted to drive the car, it travelled correctly in the direction which the car was facing.

The final feature to implement as part of this stage is making it so that the car cannot go past the edges of the display window. The XPos and YPos attributes which are passed into pygame's blit function refer to the middle of the car image. This means that if I

restricted the values of XPos and YPos to the dimensions of the screen then half of the car image would still be outside of the screen. This means that when the XPos and YPos are updated in the move_car function, they must be kept between 0 and the edge of the display window – half of the image's size.

```
61      def move_car(self):
62          self.XPos += self.XSpeed #update X and Y values
63          self.YPos -= self.YSpeed
64
65          #Adding boundaries to the screen
66          if self.XPos > screen.get_width() - CarImage.get_width():
67              self.XPos = screen.get_width() - CarImage.get_width()
68          if self.YPos > screen.get_height() - CarImage.get_height():
69              self.YPos = screen.get_height() - CarImage.get_height()
70
71          if self.XPos<0:
72              self.XPos = 0
73          if self.YPos<0:
74              self.YPos = 0
```

Here is the validation of the X and Y positions. When I ran the program, The top and left edges of the screen correctly prevented the car from passing past them and the car displayed correctly. However, with the right and bottom edges of the screen, the car was also prevented from moving past them but at certain rotations of the car, the car appears to be either beyond or behind the edge of the screen when it is prevented from moving. The functionality itself of this feature has been successfully implemented and as this is a small graphical bug, I have decided to not fix this issue until stage 6 of development where the GUI and graphical design of the program will be enhanced in greater detail. This is because it is highly important that the functionality of the program is developed as soon as possible in the development cycle as that is the main focus of the project.

Now that stage 1 has been developed, I will work through the test plan for the stage which I created in the design section. This will allow me to evaluate if this stage has met the success criteria which has been set out for it.

Test number	Required inputs	Test description	Success criteria	Pass/fail
1	Run the program.	When the program is run, see if a display window appears.	A display window will appear.	Pass
2	Run the program.	When the program is run, see if a car is displayed on the display window.	A car will appear inside of the display window.	Pass

3	Run the program and press W.	When W is pressed, see if the car moves forwards.	The car will move forward when W is pressed.	Pass
4	Run the program and press S.	When S is pressed, see if the car moves backwards.	The car will move backwards when W is pressed.	Pass
5	Run the program and press A.	When A is pressed, the car's image will rotate anticlockwise.	The car will turn anticlockwise when A is pressed.	Pass
6	Run the program and press D.	When D is pressed, the car's image will rotate clockwise.	The car will turn clockwise when D is pressed.	Pass
7	Run the program and press and hold A. Then press W.	Pressing A will cause the car to rotate anticlockwise and then pressing W will cause the car to move forwards. The movement of the car should be going forward in the same direction that the car is facing after it is turned.	The car moves in the same direction the car is facing when W is pressed.	Pass
8	Run the program and press and hold A. Then press S.	Pressing A will cause the car to rotate anticlockwise and then pressing S will cause the car to move backwards. The movement of the car should be going backwards in the opposite direction that the car is facing after it is turned.	The car moves in the opposite direction the car is facing when S is pressed.	Pass
9	Run the program and press W then let go.	When W is pressed, the car will move forward. After W is let go, the force of friction should be applied to it.	After W is let go, the car should gradually slow down and eventually stop.	Pass
10	Run the program and hold W until the car reaches the edge of the display window.	The car should be stopped when it reaches the edge	The car will stop moving as it reaches the edge	Pass (functionality wise however it

		of the display window so it cannot leave.	of the display window.	is not perfect graphically)

Test 1:

Here I need to run the program to see if a display window appears. After running the program, a display window did appear so this test is a pass.

Test 2:

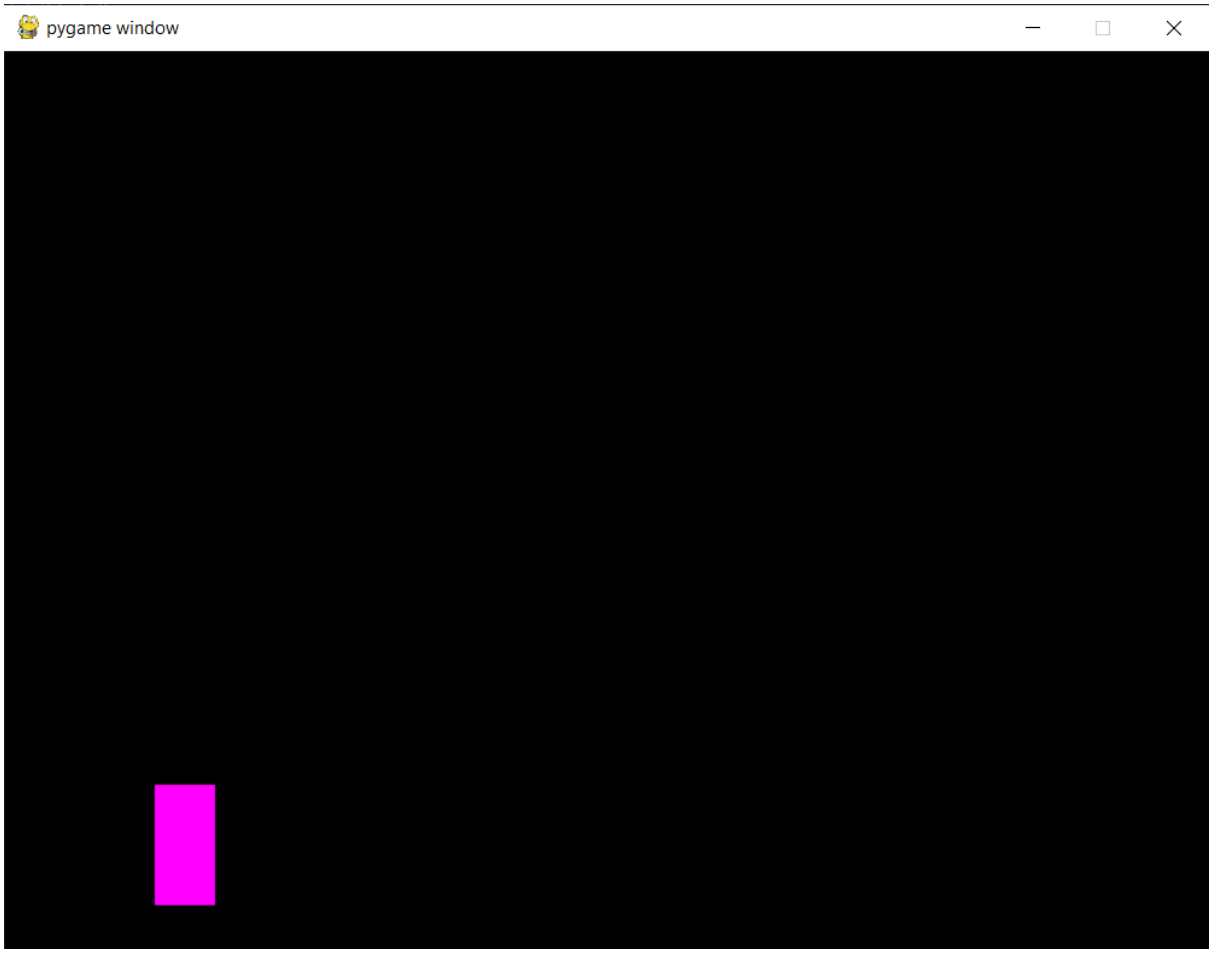
Here, I need to run the program to see if the image of a car is displayed. After running the program, the image of the car was displayed so this test is a pass.

Test 3:

Here, I need to run the program and press W to see if the car moves forwards. After running the program and pressing W, the car did move forwards after I pressed W so this test is a pass.

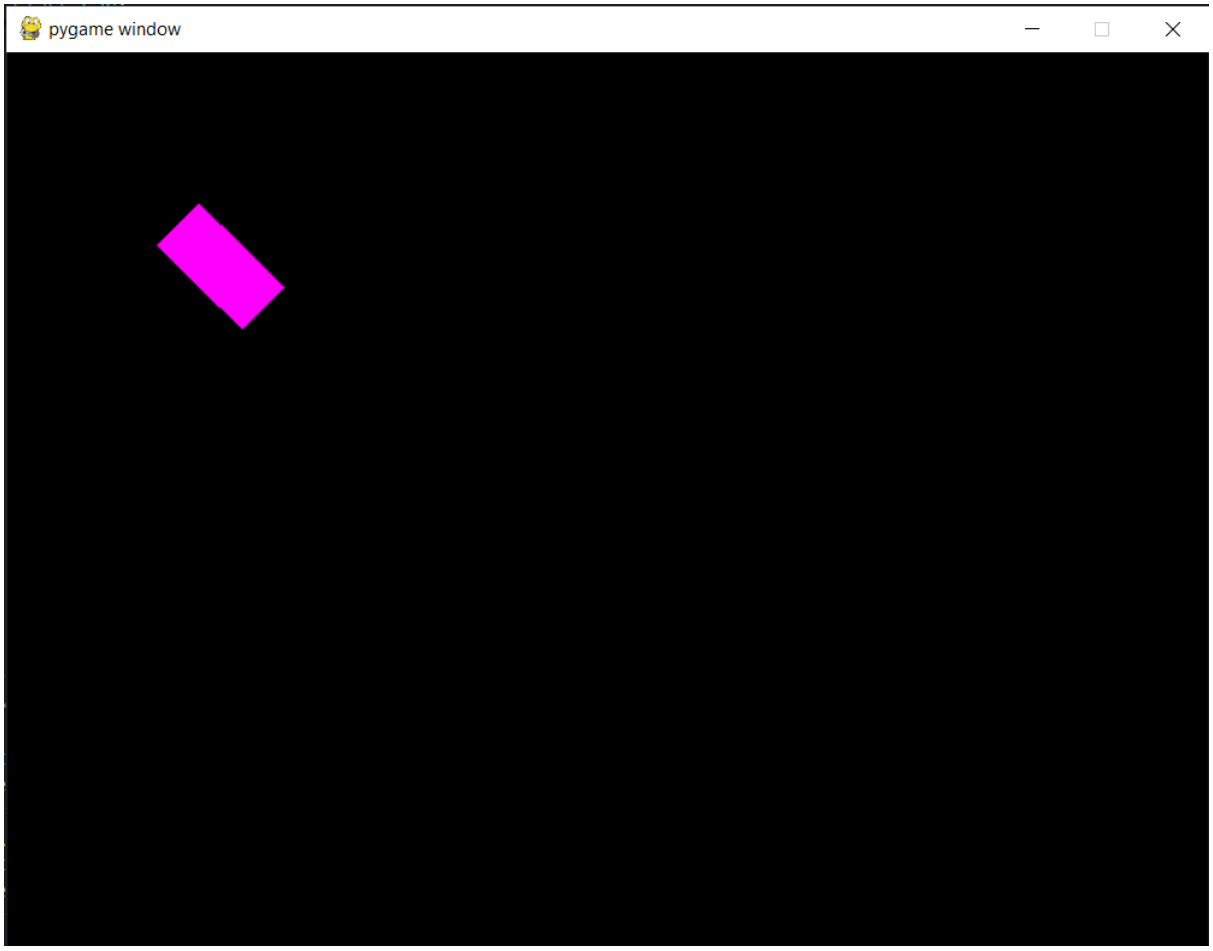
Test 4:

Here, I need to run the program and press S to see if the car moves backwards. After running the program and pressing S, the car did move backwards after I pressed S so this test is a pass.



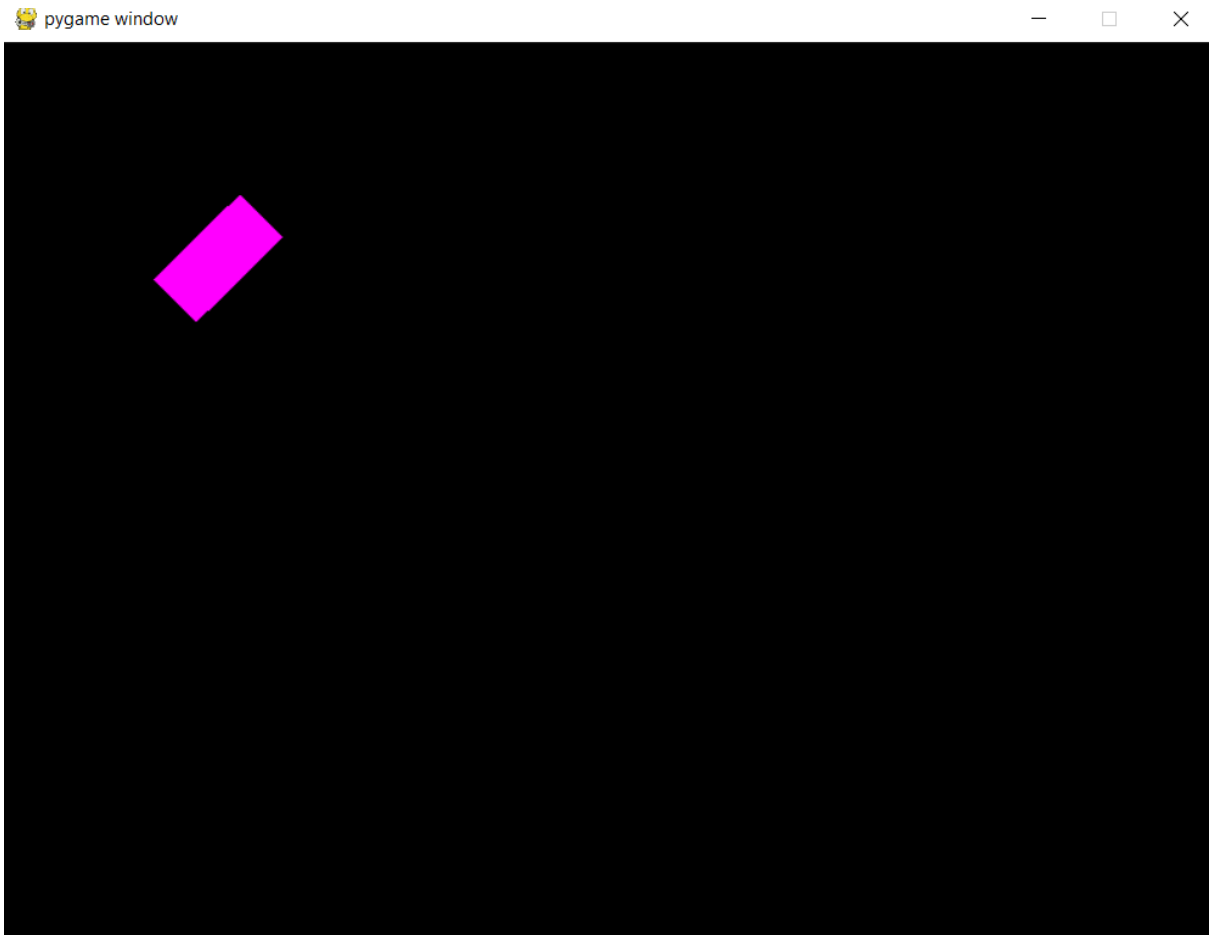
Test 5:

Here, I need to run the program and press A to see if the car turns left. After running the program and pressing A, the car did turn left after I pressed A so this test is a pass.



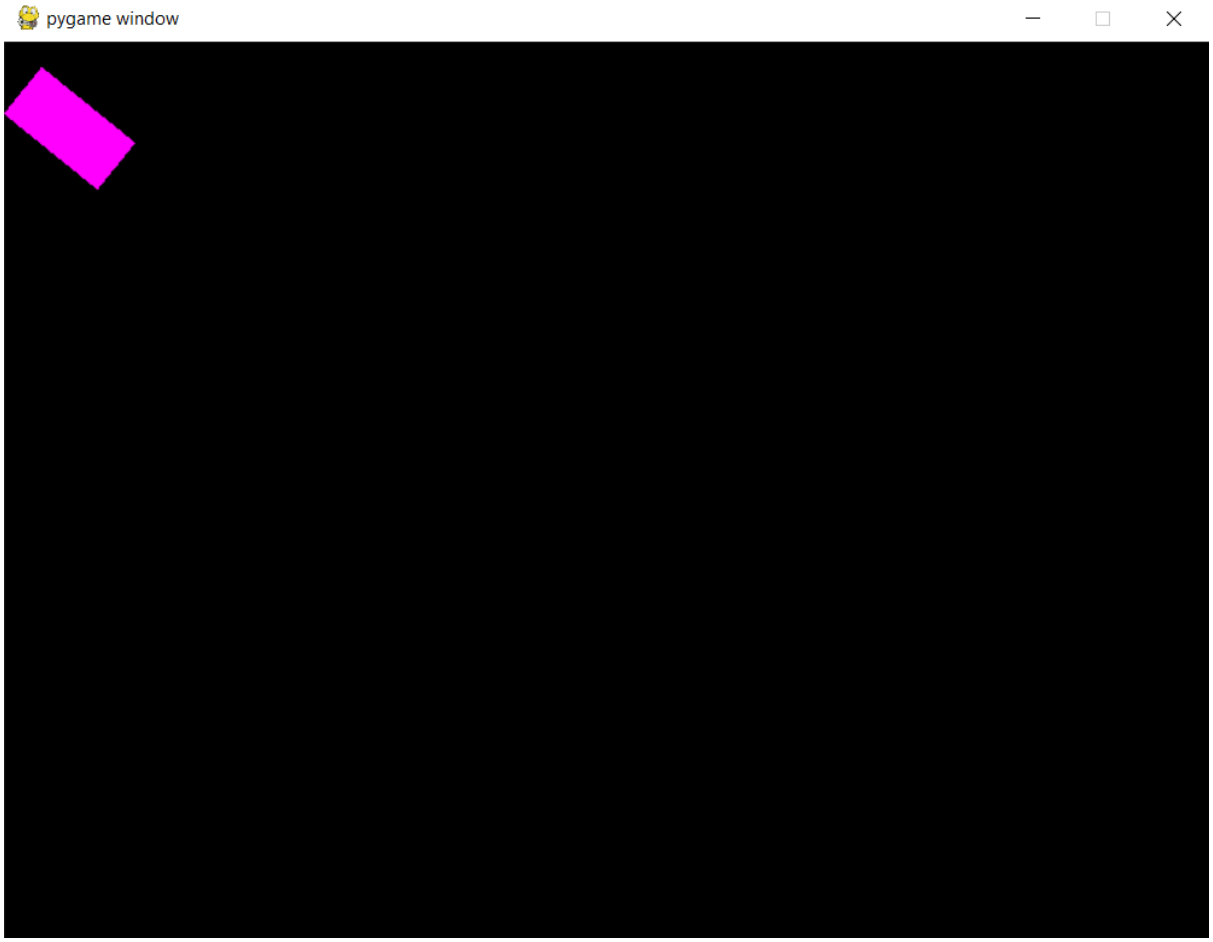
Test 6:

Here, I need to run the program and press D to see if the car turns right. After running the program and pressing D, the car did turn right only after I pressed D so this test is a pass.



Test 7:

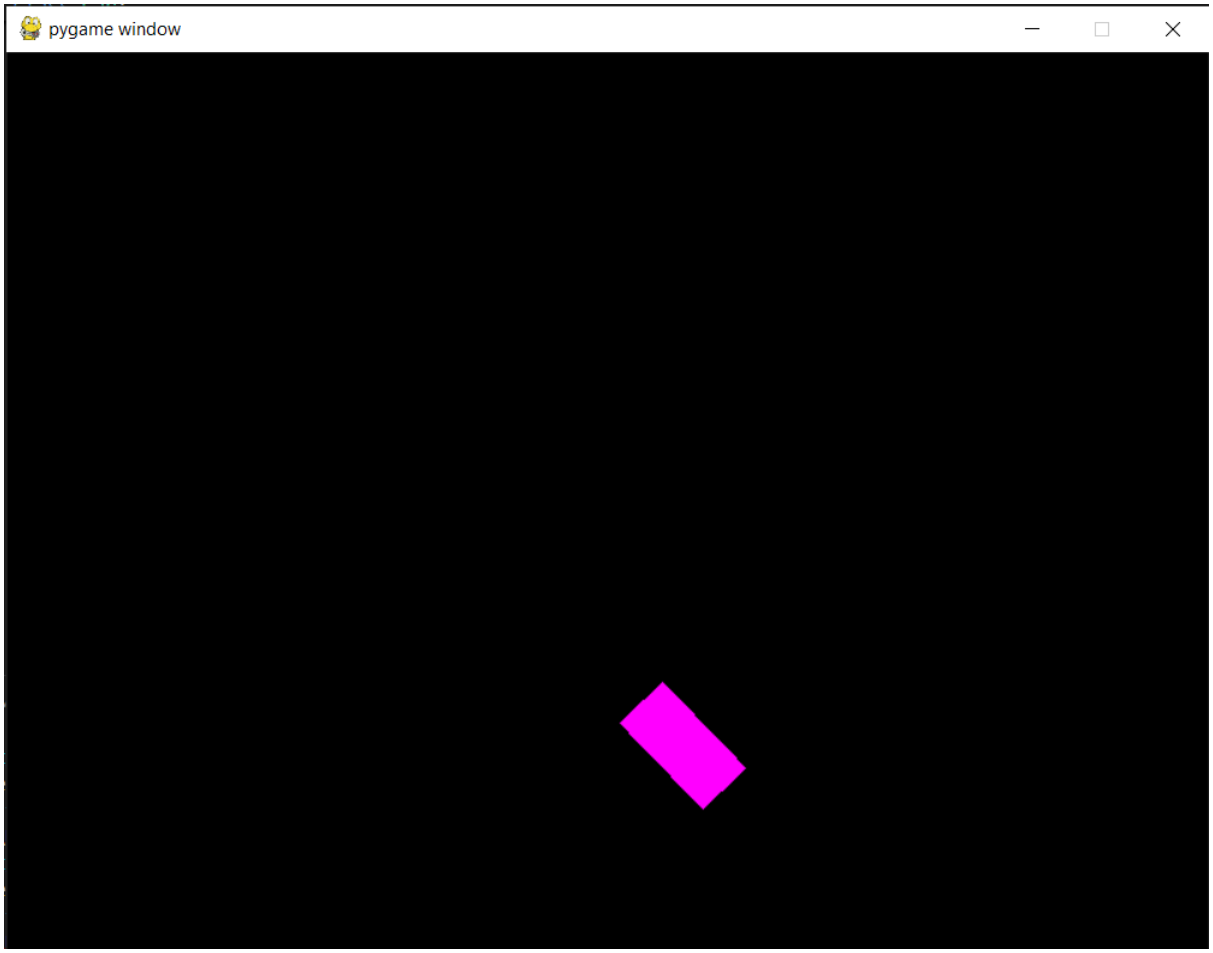
Here, I need to run the program and press and hold A. Then press W to see if the car moves in the direction that the car is facing. After running the program and doing this test, the car did move in the direction it was facing so this test is a pass.



Test 8:

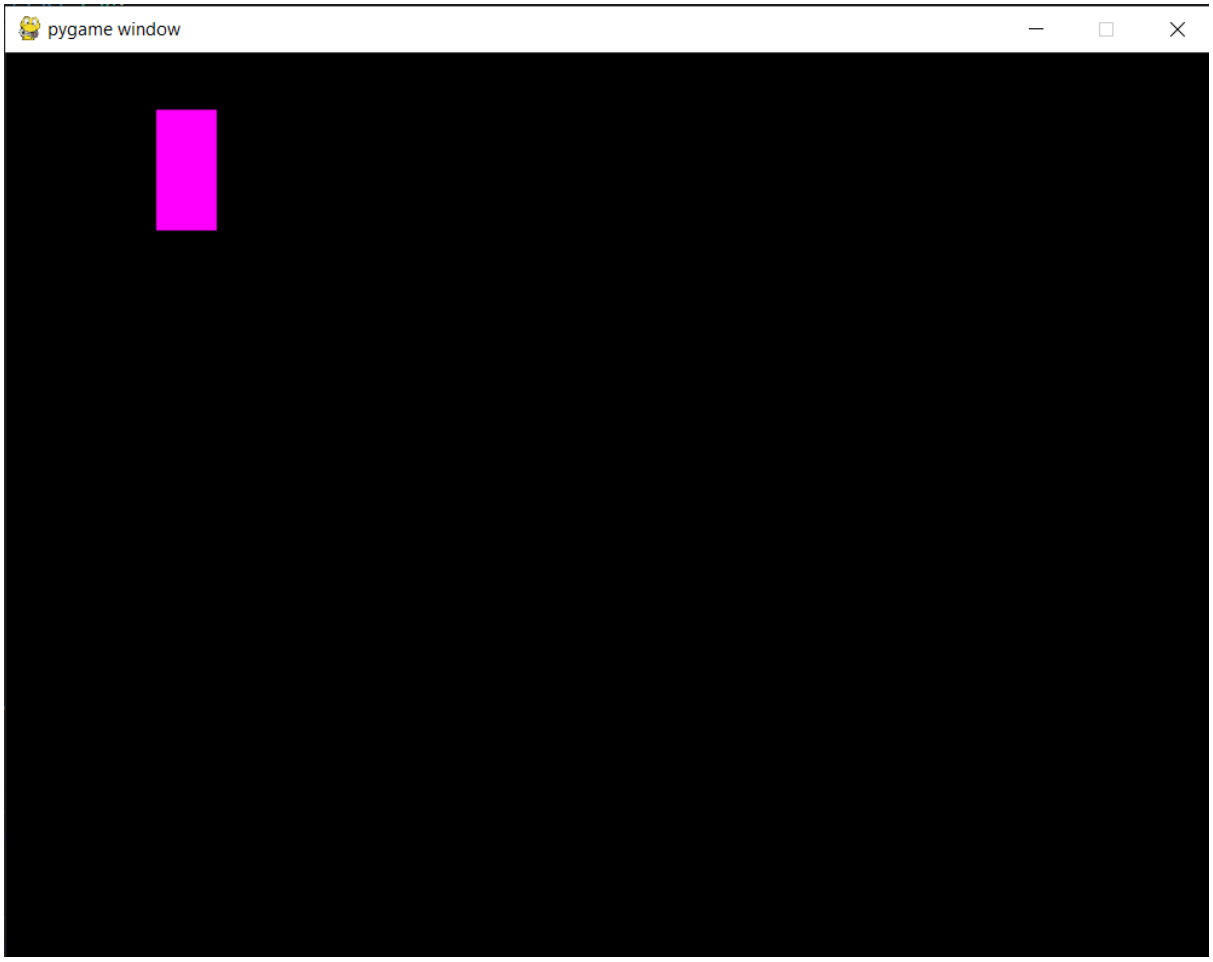
Here, I need to run the program and press and hold A. Then press S to see if the car moves in the opposite direction that the car is facing. After running the program and doing this test, the car did move in the opposite direction than it was facing so this test

is a pass.



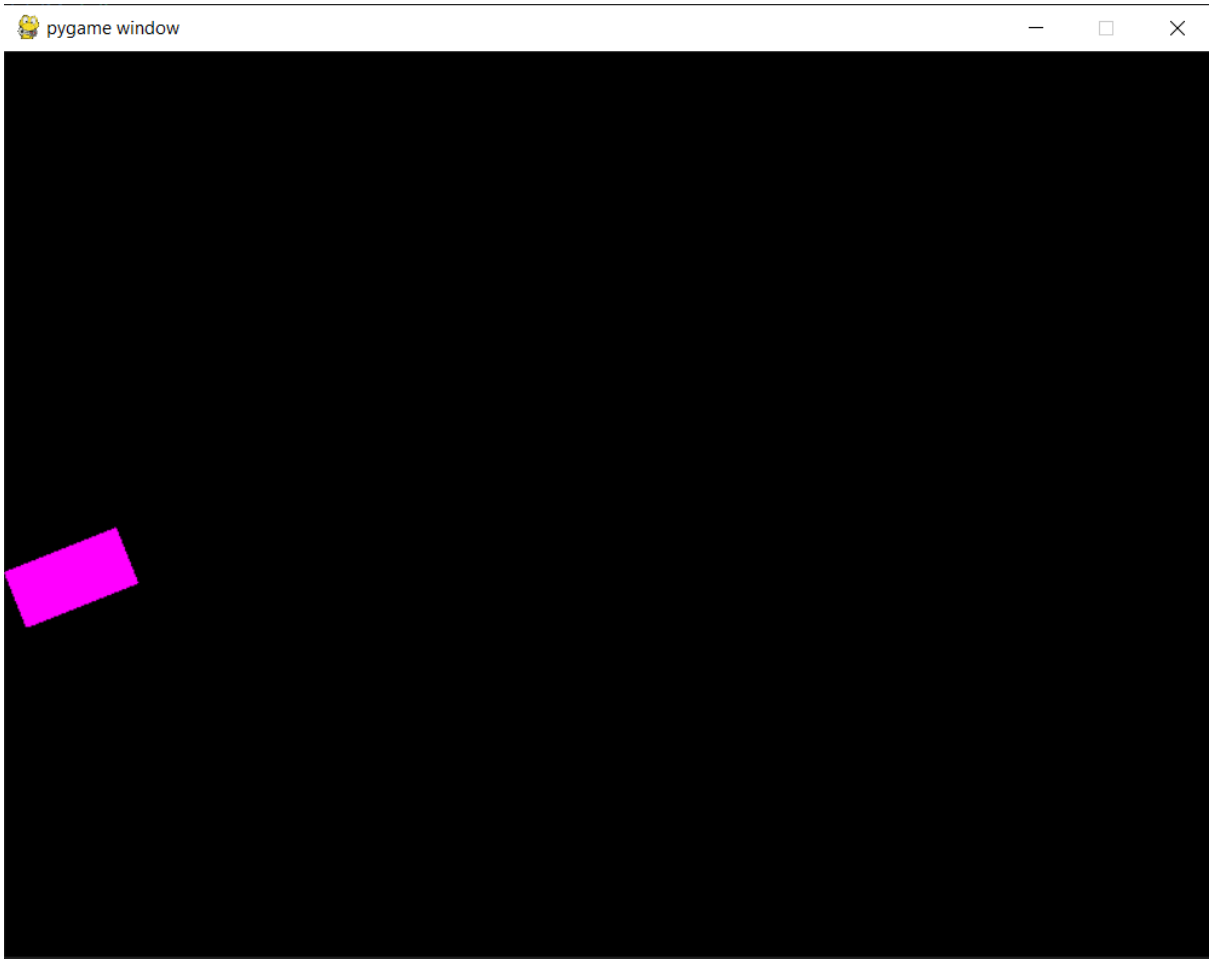
Test 9:

Here, I need to press and hold W and then let go to see if the frictional force slows down the car and then stops it from moving. After running the program and doing this test, after W was let go of the car slowed down and eventually stopped so this test is a pass.

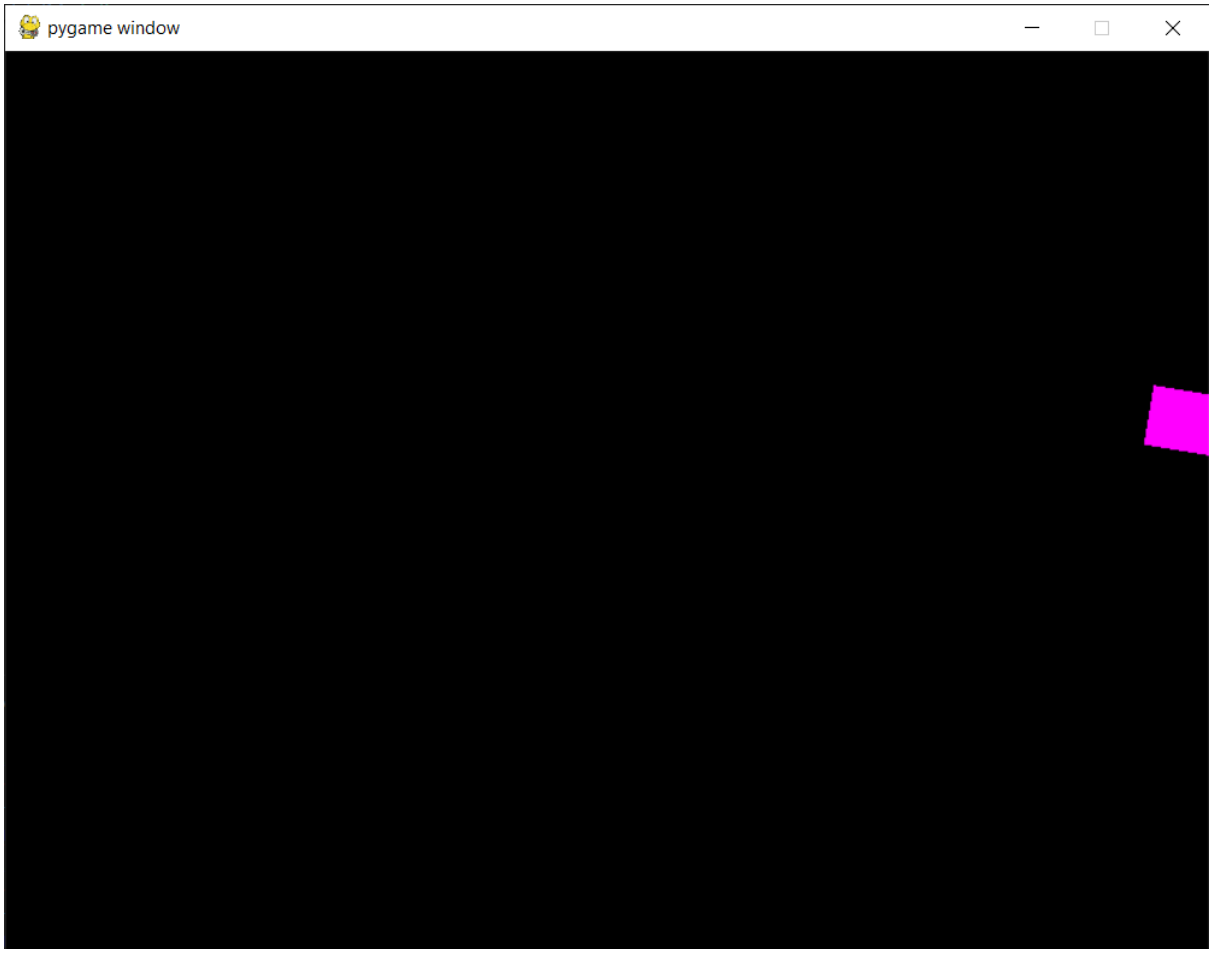


Test 10:

Here I need to press and hold W until the car reaches the edge of the display window. After running the program and completing this test, the car could not move past the edge of the display window, so this test is a pass. However, at the end of this stage, this only displays properly all the time on the top and left edges of the screen whereas the bottom and right edges of the screen display the car as stopping too far or too short of the edge of the window at certain rotations of the car. This test is still a pass as the functionality of this feature is correct to be used in the rest of the project, however fixing the issue in a later stage would make the program more appealing to users as the engagement of the program is highly important to ensure that this project is a success.



The image above is where the car cannot pass the edge of the screen and it is displayed correctly.



The image above is where the car still cannot pass the edge of the screen however it is not displayed correctly.