

Development of a racing game as an AI teaching aid

David Lambeth

BSc (hons) Computer Science

2018

This dissertation may be made available for consultation within the University Library and may be photocopied or lent to other libraries for the purposes of consultation.

Automobile

COPYRIGHT


Attention is drawn to the fact that copyright of this dissertation rests with its author. The Intellectual Property Rights of the products produced as part of the project belong to the author unless otherwise specified below, in accordance with the University of Bath's policy on intellectual property (see <http://www.bath.ac.uk/ordinances/22.pdf>). This copy of the dissertation has been supplied on condition that anyone who consults it is understood to recognise that its copyright rests with its author and that no quotation from the dissertation and no information derived from it may be published without the prior written consent of the author.

Declaration

This dissertation is submitted to the University of Bath in accordance with the requirements of the degree of Bachelor of Science in the Department of Computer Science. No portion of the work in this dissertation has been submitted in support of an application for any other degree or qualification of this or any other university or institution of learning. Except where specifically acknowledged, it is the work of the author.

☐ Retain for Sample

Mark

 <p>UNIVERSITY OF BATH</p> <p>Department of Computer Science</p> <p>INDIVIDUAL COURSEWORK Submission Cover Sheet</p> <p>Please fill in both columns in BLOCK CAPITALS and post into the appropriate Coursework Submission Box.</p>	<p><i>For office use</i></p> <p>Date and time received</p> <p>This section will be retained by the Department Office as confirmation of hand-in</p>
<p>How to present your work</p> <ol style="list-style-type: none"> 1. Bind all pages of your assignment (including this submission sheet) so that all pages can be read by the marker without having to loosen or undo the binding. Ensure that the binding you use is secure. Missing pages cannot be marked. 2. If you are required to submit part of the work on a disk, place the disk in a sealed envelope and bind the envelope into the submission. <p>You must keep a copy of your assignment and disk. The original is retained by the Department for scrutiny by External Examiners.</p>	<p>Declaration</p> <p><i>I certify that I have read and understood the entry in the Department of Computer Science Student Handbook on Cheating and Plagiarism and that all material in this assignment is my own work, except where I have indicated with appropriate references. I agree that, in line with Regulation 15.3(e), if requested I will submit an electronic copy of this work for submission to a Plagiarism Detection Service for quality assurance purposes.</i></p>
<p>FAMILY NAME LAMBETH</p>	<p>FAMILY NAME LAMBETH</p>
<p>GIVEN NAME DAVID</p>	<p>GIVEN NAME DAVID</p>
<p>UNIT CODE CM30082</p>	<p>UNIT CODE CM30082</p>
<p>UNIT TITLE INDIVIDUAL PROJECT</p>	<p>UNIT TITLE INDIVIDUAL PROJECT</p>
<p>DEADLINE TIME AND DATE 04/05/2018 6pm</p>	<p>DEADLINE TIME AND DATE 04/05/2018 6pm</p>
<p>COURSEWORK PART (if applicable)</p>	<p>COURSEWORK PART (if applicable)</p>
<p>SIGNATURE</p>	<p>SIGNATURE</p>

Abstract

The presence of Artificial Intelligence (AI) in the Computing sector is rapidly increasing, resulting in a growing need for the new generation of programmers to have AI as a key skill within their skillset. This project aimed to provide a teaching aid for AI by producing a car racing game that can be controlled by an external AI using only HTTP requests.

The product created, Automobile, consists of a Java back end server that handles game logic, as well as a Dynamic HTML front end that can be used to view a race or participate as a human using a keyboard. The game can be completely controlled by an external AI client, that is intended to be written by students learning AI.

Further development of Automobile could include creating a drag and drop client to remove the need for textual programming language knowledge, allowing much younger children to learn AI principles.

Contents

List of Figures	viii
List of Tables	ix
List of Equations	x
Acknowledgements	xi
Chapter 1. Introduction	1
1.1. Goals	1
1.2. Motivation	1
Chapter 2. Literature and Technology Survey	3
2.1. Games and Education	3
2.2. Importance of Artificial Intelligence	3
2.3. Teaching Artificial Intelligence using Computer Games	3
2.4. Computer Games vs Simulations	3
2.5. Current Technologies	4
2.5.2. Computer Games and Simulators	4
2.5.3. Educational Games	6
2.5.4. Educational Simulators	6
2.5.5. Educational Simulators for Artificial Intelligence	6
2.6. Conclusion	7
Chapter 3. Requirements Analysis and Requirements Specification	9
3.1. Establishing Requirements	9
3.1.1. Questionnaire	9
3.1.2. Use Cases	11
3.2. Requirements	12
3.3. Functional Requirements	12
3.3.1. Creating a Race	12
3.3.2. Viewing a Race	13
3.3.3. Controlling a Car	14
3.4. Non-Functional Requirements	15
3.4.1. Front End	15
3.4.2. Back End	16
Chapter 4. Design	17
4.1. Technologies	17
4.1.1. Unity	17
4.1.2. Classic Web	17
4.1.3. Server Back End	18
4.2. Chosen Technologies	19
4.2.1. Front End	19

4.2.2.	Back End.....	19
4.2.3.	AI Client.....	19
4.3.	Version Control.....	19
4.4.	Development Methodology.....	20
Chapter 5.	Detailed Design.....	21
5.1.	Design Iterations	21
5.1.1.	Server	21
5.1.2.	Front End	21
5.1.3.	AI Client.....	22
5.2.	System Architecture.....	22
5.2.1.	Front End	23
5.2.2.	Server	23
5.3.	User Interface.....	23
5.3.1.	Menu Screen	23
5.3.2.	Create Race Screen	24
5.3.3.	Join Race Screen	24
5.3.4.	Race Screen.....	25
5.3.5.	Results Screen	26
5.3.6.	Controls Screen	26
5.3.7.	How To Screen.....	27
5.4.	Server-Side Design	27
5.4.1.	Physics	28
5.4.2.	In-game AI	32
5.5.	Communication.....	33
Chapter 6.	Implementation	35
6.1.	Server.....	35
6.1.1.	Timing.....	35
6.1.2.	HTTP Controller	35
6.1.3.	Waypoints	35
6.1.4.	In-game AI.....	36
6.2.	Front end	36
6.2.1.	Drawing.....	36
6.2.2.	Overview or Centred.....	36
6.3.	Server to Front end.....	36
6.4.	Implementation Challenges.....	37
6.4.1.	Server	37
6.4.2.	Front End	38
6.4.3.	Server to Front End	38
Chapter 7.	System Testing	39

7.1. Tick Rate	39
7.2. Request Performance	40
7.3. Physics Testing	44
Chapter 8. Conclusions	45
8.1. Retrospective.....	45
8.1.1. Technology Choices.....	45
8.1.2. Library Usage.....	45
8.1.3. Scope Creep	45
8.2. Further Development	45
8.2.1. Improving Playability.....	45
8.2.2. Creating an Extendable AI Client	46
8.3. Originality of product.....	46
8.4. Commercial Viability.....	46
Bibliography.....	47
Appendix.....	i

List of Figures

Figure 1 - Showing the difference between real world images and in game graphics (left side is real world, right is in game) (Alban, 2017).....	5
Figure 2 - A teacher creating a race	11
Figure 3 - A teacher searching for and viewing a race	11
Figure 4 - A student searching for and viewing a race	11
Figure 5 - An AI updating a car's controls	12
Figure 6 - System architecture diagram showing how the system components interact.....	22
Figure 7 - A mock up for the menu screen	23
Figure 8 - A mock up for the create race screen.....	24
Figure 9 - A mock up for the joining race screen	24
Figure 10 - A mock up for the race screen with a local car chosen	25
Figure 11 - A mock up for the race screen as an overview	25
Figure 12 - A mock up for the results screen	26
Figure 13 - A mock up for the controls screen	26
Figure 14 - A mock up for the how to screen	27
Figure 15 - Diagram showing longitudinal and lateral force directions	28
Figure 16 - A graph for finding torque from RPM, (Sarvaiya, 2018).....	30
Figure 17 - Illustration for how to calculate turning radius	31
Figure 18 - Illustration of the wheel base constant (L)	32
Figure 19 - Illustration of the angle to waypoint.....	36
Figure 20 - Graph for average response time for view request.....	41
Figure 21 - Graph for bytes transferred per second for view request	41
Figure 22 - Graph for average response time for vehicle update request	42
Figure 23 - bytes transferred per second for update vehicle request.....	42
Figure 24 - Graph for average response time for search request	43
Figure 25 - Graph for bytes transferred per second for search request	43

List of Tables

Table 1 - Requirements for front end creating a race.....	12
Table 2 - Showing requirements for back end creating a race	13
Table 3 - requirements for front end viewing a race	13
Table 4 - requirements for back end viewing a race	14
Table 5 - requirements for front end controlling a car	14
Table 6 - requirements for back end controlling a car	15

List of Equations

Equation 1 - Second iteration engine force	29
Equation 2 - Second iteration braking force	29
Equation 3 - Second iteration drag force	29
Equation 4 - Second iteration rolling resistance	29
Equation 5 - Second iteration total longitudinal forces	29
Equation 6 - Acceleration	29
Equation 7 - Velocity from old velocity and acceleration	29
Equation 8 - Second iteration rotation	29
Equation 9 - Third iteration engine force	30
Equation 10 - RPM	30
Equation 11 - Turning radius	30
Equation 12 - Angular velocity	31
Equation 13 - Lateral slip angle of the car	31
Equation 14 - Lateral slip angle of the front wheels	31
Equation 15 - Lateral slip angle of the rear wheels	31
Equation 16 - Front wheel's lateral force from friction	31
Equation 17 - Rear wheel's lateral force from friction	31
Equation 18 - Total lateral forces on the car from cornering	32
Equation 19 - Front wheel weight	32
Equation 20 - Rear wheel weight	32

Acknowledgements

I would like to thank my supervisor on this project, Dr. Tom Fincham Haines, for guiding me through the process of writing the longest piece of work I have ever completed.

To my parents and family, for all the continued support through my years of schooling and University.

Finally, to my puppy Cooper, your picture has been above my desk cheering me on the entire time.

Chapter 1. Introduction

Artificial Intelligence (AI) is a large field with far reaching applications in everyday life as well as in many specific industries. Its prevalence demonstrates the considerable need for incoming developers to be able to understand and utilise AI within their roles.

1.1. Goals

This project aims to create an online racing game that can be used as a simple tool to aid the teaching of AI. It will be a ready-made resource that teachers can set up within a lesson to give students access to an AI testing ground. It will be designed in a way that is very versatile, making sure that any technologies needed to run the tool are commonplace.

1.2. Motivation

Teaching without the proper tools is as difficult as building without the proper tools. Particularly when it comes to teaching concepts within Computer Science, as it can be difficult for students to link principles to real world scenarios.

Having taught Computing at an academy that designed software to fit the courses they wanted to teach, I understand first-hand that there is a severe lack of specific computing skills teaching aids for many topics. It cannot be assumed that all teachers will be able to devote time and attention to the building of the teaching aids that they require to teach, nor can it be assumed that they possess the necessary skills to do so.

Then Education Secretary Michael Gove claimed that the newly introduced curriculum changes were “designed to equip every child with the computing skills they need to succeed in the 21st century” (Department for Education, 2014). This claim and the changes do not specifically mention AI, but as the sector continues to grow, arguably there will be a need for more curriculum changes to include AI in schools (National curriculum in England: computing programmes of study, 2018).

Chapter 2. Literature and Technology Survey

The educational environment is full of competition, for instance with students attempting to get the highest grades. Competition can be used to create excitement which has the potential to engage students and increase focus that will lead to higher learning (Sadler, 2005).

This Literature review will primarily examine how computer games can be used for educational purposes with respect to teaching AI and how this activity can be analysed quantifiably.

2.1. Games and Education

Games have been used as learning aids for hundreds of years, with simple games such as chess and mah-jong. Modern technology advances have increased the potential for gamification of many areas in life, such as motivation to exercise and education (Koivisto and Hamari, 2014). Furthermore, research has found that games are an effective and viable vehicle to engage students and communicate new concepts (Byrne, 2016).

Games have already been developed that teach most topics including English language, Mathematics and Science, such as simple programming using KTurtle which helps to teach programming skills. These games bring about a personal component, where the avatar within the game is an embodiment of the player, allowing the player to move into their role inside the game as that avatar. This means that players can directly learn from the experiences of their avatar.

2.2. Importance of Artificial Intelligence

AI is used to assist in many industries, including gaming and banking.

AI is used in games to create a more realistic experience for the player, by adjusting parameters or using randomness to make every round of a game feel unique. AI can be of varying complexities and qualities, the better ones providing a greatly increased immersion for the player.

Banking utilises AI for mitigating fraudulent activities, building virtual advisors and catching mistakes in transfers (Wells Fargo Testing Bot For Messenger Featuring New Customer Service Experiences | Wells Fargo Online Newsroom, 2018), (Fraud Protection Solutions for Banks, 2018), (How BNY Mellon Became a Pioneer in Software Robots - Blue Prism, 2018).

AI of all sorts is becoming big business, being predicted by Constellation Research to surpass \$100 billion by 2025, which means that educating the next generation of AI workers is also significant (Morris, 2017).

2.3. Teaching Artificial Intelligence using Computer Games

In many cases, learning an abstract concept is difficult if you are unable to visualise it in a way that you can understand. Whether the difficulty lies in complicated mathematics or chemical reactions, finding a clear way to see the information is often the best way to form a coherent understanding of the subject area.

Bringing AI into the “real” virtual world with a computer game that is easily understood and shows the overall working of the AI can create an environment where the complexity is easier to comprehend. There is strong evidence that the use of games to engage students, in particularly in the topic of AI can improve the focus and therefore the learning of those students (Garris, Ahlers and Driskell, 2002) (Pozzer, Cesar & Karlsson, 2007).

2.4. Computer Games vs Simulations

So far computer games have been the main focus of the review, however, the difference between a computer game and a simulation has to be considered. The Oxford Dictionary defines a simulation as “The production of a computer model of something, especially for the purpose of study” (The Oxford English dictionary, n.d.). The main difference between a simulation and a

computer game is the reason behind the creation, with simulations generally used for study purposes and learning, whereas computer games are created for entertainment and fun.

This difference of purpose leads to other divergences, such as the level of realism and exact replication of real life that is built into a simulation. If a game is for entertainment, then the need for realism is greatly reduced, its purpose is to be fun rather than to create an experience which is exactly the same as real life.

2.5. Current Technologies

2.5.2. Computer Games and Simulators

Need for Speed

Need for Speed is a computer game series that centres around street racing, it is one of many games of this style and has been chosen for its popularity. The game design focuses heavily on entertainment and many of the game mechanics are clearly adapted for an enjoyable playing experience.

Driving Mechanics

The driving mechanics in Need for Speed are abstracted from real physics to enhance the level of enjoyment taken from playing the game.

The general driving mechanics, such as the car's slip and skid control are removed from reality, giving the driver a significant amount of control that for a real car is not possible.

Another feature is the nitrous oxide "boost" mechanic, which allows a car to accelerate at an increased rate. Though this is something that real-life cars sometimes use with the aim of increasing acceleration, the game avoids the consideration of extra wheel slip from this acceleration and the boost refills automatically, rather than having to physically refill a canister of gas.

Road Laws

The road laws in the Need for Speed games are in most cases almost completely removed. Though the other cars on the road stick to their side, there is no gain for the player from driving correctly.

Additionally, only in the main exploration mode is there even the possibility of driving properly. More often than not, in the races, there is no way to abide by road laws as there are blockades forcing the user to follow the defined track.

Graphics

The graphics within the game are realistic, giving the game a feel of realism within its own scope. Some images from Bryn Alban, who is the Vehicle Art Director for Ghost games, show a real-life image and the game's version of the same scene. In these images it is virtually impossible to determine which part is photography and which is an example of gameplay.



Figure 1 - Showing the difference between real world images and in game graphics (left side is real world, right is in game) (Alban, 2017)

Aspects considered

The driving mechanics are very game oriented, and this would be an interesting addition to the project, as it would give the chance to observe an AI trained to drive using realistic style mechanics trying to drive with game style mechanics.

The same logic applies to the road laws, it would be interesting to see the differences between enforcing road laws and not doing so.

Graphics will likely not play a big part in the project, largely due to the amount of time that would need to be spent to achieve a good result. In particular, the 3D aspect of the graphics would present difficulties in creating and training the AI effectively.

Truck Simulator

Truck Simulator is a simulation game based on long haul truck driving. Though it is a simulation, it is a game at heart and contains some gamification of the physics to allow a fun experience and to allow a personal computer to process it.

Driving Mechanics

The driving mechanics are designed to feel realistic, giving as close to a real-life driving experience as possible. Though there are very realistic driving details, there are also real-life issues missing, such as slipping whilst driving on ice, or hydroplaning.

Road Laws

The game does include road rules, such as red lights and speed traps. There is nothing that actively prevents poor driving, however like in the real world, there are penalties for each rule broken. The game does include police cars such that, if you commit a crime in their presence, they will fine you. However, the police take no action to stop the driver, but merely apply a penalty to the overall score and the player can carry on as if nothing happened.

Aspects considered

The driving mechanics of Truck Simulator are advanced and balanced to provide a realistic driving experience. This level of mechanical reality is not achievable in this project, especially as they often use peripheral devices to represent effects, such as steering wheel feedback. This said, the feel of a realistic driving experience will be strived for.

2.5.3. Educational Games

KTurtle

KTurtle is an educational programming environment that aims to make programming simple. The language used is loosely based on Logo and contains few simple commands that allow the turtle to move across the canvas, drawing a line behind it (Edu.kde.org, 2017).

How it is educational

Though KTurtle doesn't have inbuilt challenges, it is easy for a teacher to create challenges such as drawing a triangle, which can then be completed by students. As the solution will need to be written in code, the student will learn programming in this way.

Aspects considered

KTurtle is fairly old and so doesn't show cutting edge technology for the most part. However, the lack of challenges and scoring is very apparent when using it as it contrasts considerably with modern games.

2.5.4. Educational Simulators

VirtualHeroes

VirtualHeroes is a collection of simulations that are used to create a detailed training environment for many situations, including military and medical. It uses a virtual reality or augmented reality approach to immerse the user into a scenario that would normally be too dangerous or too complicated to create in the real world (Virtualheroes.com, 2017).

The software focuses heavily on representing real life as accurately as possible to provide effective training (Virtualheroes.com, 2017).

Graphics

The representation of real life starts with the impressive graphics in the simulations. "Virtual Heroes shows an astute eye in anatomical correctness of the subject and surrounding environments" (Virtualheroes.com, 2017), which implies that the detail in the simulations is intended to present an environment in sufficient clarity to trick a professional.

Aspects considered

VirtualHeroes is very detailed as it attempts to simulate a real situation to a close enough degree that a human can be trained as comprehensively as if they were in that real situation. This project will not be simulating to that level of detail as it would take years to develop, however the objective is to balance a suitable level of realism whilst keeping it a game.

2.5.5. Educational Simulators for Artificial Intelligence

OpenAI Universe

OpenAI's Universe is software that aims to create an environment to train a general intelligence by giving it access to as many games as possible (OpenAI Blog, 2017). It enables any program that can run on a computer be run within the Universe, with the AI running on it.

How does it work

It uses Docker containers, which hold a normal state with a program running inside, to train, test and compare reinforcement learning AIs on the program running.

Each game within Universe has a scoring mechanism that will allow an AI to know that it has begun to perform a positive action and that it should continue to perform positive actions. The AI that is external to the container but local to the machine will be requested 60 times a second to provide input using keyboard and mouse commands (OpenAI Blog, 2017).

One of the best features of Universe is the ability for a human to oversee the AI as it learns. This is possible because the AI runs by looking at the pixels on the screen, so showing the pixels is easy. It is also possible for a human to play the game, with potential for the AI to learn from the human.

Aspects Considered

Universe's design allows for a completely external AI to play the games that it has available and it does this by giving simple inputs and outputs that are external from the game. Some of these design principles will be employed when considering how to minimise the AI's inclusion in the game.

2.6. Conclusion

The largest contribution of ideas to this project from current technologies will be from Universe, particularly its separation between AI and game. This separation means that the functioning of the game is hidden from the AI and so updated versions can be released without AI writers having to update their code. Furthermore, it makes the writing of the AI more heavily focused on the AI and less on the game itself as most of the game's logic and variables are hidden.

Need for Speed's impressive graphics will not be used as a basis for graphics in this project, mostly due to time and processing constraints, but also for design preferences, giving a more simplistic aesthetic to the game.

The level of detail in the simulating of real-world physics will find a balance between the arcade style of Need for Speed and the simulator style of VirtualHeroes, in order to create a realistic feel whilst keeping the processing to a minimum to maintain high frame rates.

Chapter 3. Requirements Analysis and Requirements Specification

3.1. Establishing Requirements

There was no stakeholder used for this project, instead requirements were gathered from various sources. Existing solutions and use cases were the largest influencers of requirements, and a questionnaire was also used.

3.1.1. Questionnaire

See Appendix A.1. The questionnaire was designed to survey people who had previously been taught AI, hoping to obtain information on the existing tools and methods teachers are currently using. Teachers and students who have used other solutions are suitable sources for requirements as they have already formed opinions on the solutions that they have available.

The results of the questionnaire (Appendix A.2) provided few suggestions for what should and shouldn't be included, although it did show that the educational sector appears to use tools that are only used by a single person at a time, on their own machine.

There were some results that were not particularly useful, as the respondents had not been a part of any AI courses and did not play any driving games. These respondents were removed from the results. Existing Solutions

Top Down Racing Games

Some existing solutions were found through the questionnaire, and these were then played to get a feel of their features. Other solutions were ones already known and these were analysed in the same way.

Key features or attributes that most or all these solutions had in common were:

- Countdown to start
- Music during setup
- Music during race
- Finish screen, showing the times of each car
- Position indicator, showing what place you are in
- Keyboard controls were roughly the same between games
- The number of cars in the race was not variable

Some features won't be included, such as the number of cars being static, as this doesn't fit with the motivation of having a classroom situation, where there could be any number of students.

Features such as a countdown will be in the requirements as these are necessary for the function of the game.

Web Browser Multiplayer Games

The games that were analysed for this project are LazerSharks.io and Littlewargame.

LazerSharks.io

LazerSharks.io is a simple game where the user plays as a shark with a laser attached to its head (LazerSharks.io | Play LazerSharks.io for free on Iogames.space!, 2018). The aim of the game is to eat the little fish that live around the game world whilst avoiding being killed by other sharks. The main takeaway from this analysis is how quickly the game is joined. There is no lengthy setup or signup process; the user simply chooses a nickname and a colour, and then can begin.

Littlewargame

Littlewargame is a strategy game where the user can build, upgrade and battle with others or with computer players (Littlewargame, 2018). Again, the amount of time needed to join a game that has already been set up is minimal, though setting one up for yourself takes slightly longer to

allow for customisation. The features taken from this are the customisation, where the user can choose what map they want to play on, and the option of having computers to play against.

Educational AI tools

LEGO Mindstorms NXT

The LEGO Mindstorms NXT kit is a buildable and programmable robot generally used for educational purposes (Home - LEGO.com, 2018). The kit includes a programmable brick that can be used to control the robot in its movements.

As the robot is programmable and comes with sensors to perceive the environment, it provides an excellent platform for teaching AI, giving a clear display of how AI acts in a real-world scenario.

There are already courses that use the LEGO Mindstorms NXT kit to assist in teaching AI or testing knowledge. One of these is at the University of Bath for final year students, assigning a single robot to a pair for a coursework (Programme & Unit Catalogues - University of Bath, 2018).

Another course is at FunTech, which is an extracurricular academy teaching computing, whose course uses the robots to teach about path-finding (Lego NXT | Lego Robot Programming For Kids | Lego Holiday Camps, 2018). The course ends with a maze solving tournament between students, and this enables them to witness the advantages and disadvantages of different approaches.

Both courses use the robots to give a visible indication of how the AI is performing. Requirements taken from this are the visual and real-world applications, and the need for a solution that allows many students to participate together at the end.

External AI tools

OpenAI Universe

OpenAI's Universe project allows an AI to play any number of games that have been ported into the project. Each game is held within a Docker container that has a standard interface for the AI to get the output video from the game and to give the inputs chosen (Docker, 2018).

This solution gives the AI the ability to play many games using the same interface, making it possible to create an AI that is capable of playing lots of different games. The requirements gathered from this all relate to the externalised nature of the AI. Having AI that is part of a larger system, such as during developing a game and adding in AI for the enemies, means that you have create that larger system, or add in the AI after. Having a system that is already created and allows for you to make a small AI that communicates with it, such as using a Docker container, removes the need to be making something larger. The requirements gathered from Universe are relating to the external nature of the AI. Instead of the AI being built into the original project, using a Docker container means that the AI can be written after with no access to source code. This gives both a better game design and requires a more general AI, rather than one specifically designed for that game code.

3.1.2. Use Cases

Using the questionnaire and existing solutions, use cases were created. The users are a teacher, many students and an external AI.

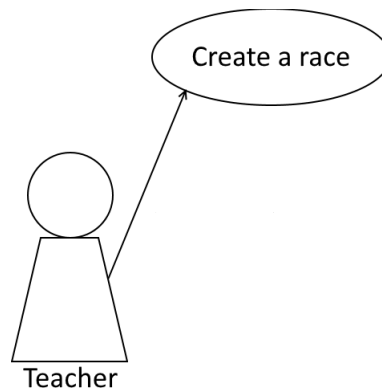


Figure 2 - A teacher creating a race

It is assumed that the user in the teacher position will be creating the race, including all setup.

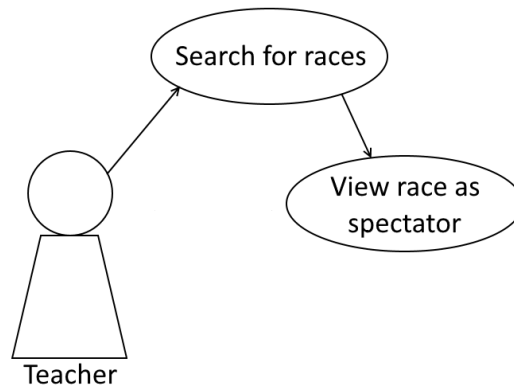


Figure 3 - A teacher searching for and viewing a race

The teacher will then be able to view the race as a spectator, which will allow the race to be shown on the main screen for all to see.

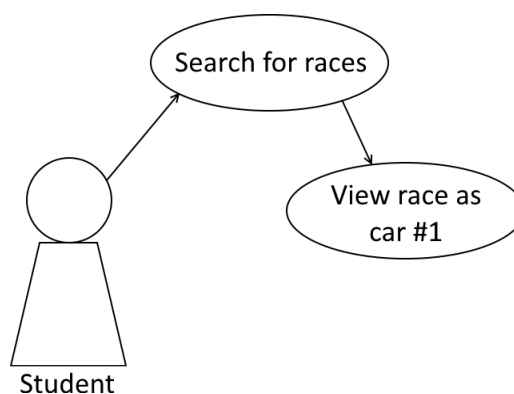


Figure 4 - A student searching for and viewing a race

The student will be able to view the race from the perspective of a single car, with the camera following the car around the racetrack. At this point the student could drive the car using a keyboard, or just watch as the external AI races.

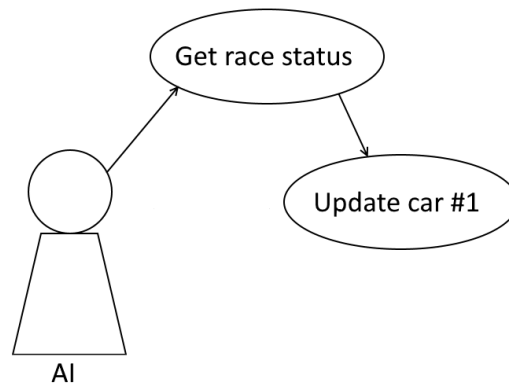


Figure 5 - An AI updating a car's controls

This most important case, the external AI being able to update the car's controls to drive it around. The AI must be able to get the status of the race, including all the information about the car's location and where it is going.

One of the things considered for the use cases is the need for separate creation and joining of a race. A single teacher will create the race and then many students will join to watch, and many AI will join to control the cars. Keeping the joining separate from the creation will make this much easier to complete.

3.2. Requirements

Each requirement has been given a number that shows its section and exact identity, this will be used as the GitHub commit name so that it is easy to see what requirement is being worked on.

The requirements will be given a description and either must, should or could, to show the importance of the requirement on the system. Any requirement with must, has to be included and working for the system to be complete.

3.3. Functional Requirements

Functional requirements describe the actual operation of the system, detailing what the system must, should and could do.

3.3.1. Creating a Race

Front End

Table 1 - Requirements for front end creating a race

ID	Priority	Summary
R.1	MUST	Create a race from a web browser
R.2	MUST	Choose options for the race
R.3	SHOULD	Have choice of track
R.4	COULD	Have choice of music/audio
R.5	SHOULD	Have choice of lap number
R.6	SHOULD	Have choice of number of cars
R.7	MUST	Able to create multiple races
R.8	MUST	Give clear message for confirming a race has been created

Back End

Table 2 - Showing requirements for back end creating a race

ID	Priority	Summary
R.9	MUST	Have endpoint to create a race
R.10	MUST	Create race in a single endpoint
R.11	MUST	Be able to create multiple races
R.12	SHOULD	Create race without affecting other races
R.13	SHOULD	Be able to create multiple races at the same time
R.14	MUST	Store race details in a list of races

3.3.2. Viewing a Race

Front End

Table 3 - requirements for front end viewing a race

ID	Priority	Summary
R.15	MUST	Be able to view a race as an overview
R.16	MUST	Show the order of cars (their positions)
R.17	MUST	Have an overview mode
R.17.1	MUST	Show entire track
R.17.2	MUST	Show every car
R.17.3	MUST	Still show order of cars (their positions)
R.17.4	MUST	Not show any individual car's information, other than position
R.18	MUST	Show when the race has started
R.19	MUST	Show a starting countdown before the race starts
R.20	MUST	Show a finish screen
R.20.1	MUST	Show position of each car
R.20.2	MUST	Show finish times of each car
R.20.3	MUST	Have button to take user back to menu page
R.21	MUST	Have centred view
R.21.1	MUST	Centre race on a chosen car
R.21.2	MUST	The track should not rotate
R.21.3	MUST	Show individual car's information
R.21.3.1	MUST	Show steering wheel angle
R.21.3.2	MUST	Show accelerator pedal depth
R.21.3.3	MUST	Show brake pedal depth
R.21.3.4	MUST	Show which gear the car is in
R.21.3.5	MUST	Show speed of the car
R.21.3.5.1	COULD	Show speed on a speedometer with a rotating dial

Back End

Table 4 - requirements for back end viewing a race

ID	Priority	Summary
R.22	MUST	Have an endpoint for viewing the race
R.22.1	MUST	Be able to serve multiple requests simultaneously
R.22.2	MUST	Have a countdown to the start of the race
R.22.2.1	MUST	Give this countdown to all requests from front end or AI
R.22.3	MUST	A finished race must give final information
R.22.3.1	MUST	Give each car's position
R.22.3.2	MUST	Give each car's finish time

3.3.3. Controlling a Car

Front End

Table 5 - requirements for front end controlling a car

ID	Priority	Summary
R.23	MUST	Register key presses
R.23.1	MUST	Update key presses at each front-end tick
R.24	MUST	Register up arrow key as increasing accelerator pedal depth
R.24.1	MUST	Have minimum accelerator pedal depth (0)
R.24.2	MUST	Have maximum accelerator pedal depth (100)
R.25	MUST	Register down arrow key as increasing brake pedal depth
R.25.1	MUST	Have minimum brake pedal depth (0)
R.25.2	MUST	Have maximum brake pedal depth (100)
R.26	MUST	Register left arrow key as decreasing the steering wheel angle (turning left)
R.27	MUST	Register right arrow key as increasing the steering wheel angle (turning right)
R.27.1	MUST	Have maximum steer angle
R.27.2	MUST	Have minimum steer angle
R.28	COULD	Register space bar as applying handbrake
R.29	COULD	Allow changing controls from the menu page
R.30	MUST	Send updates of vehicle's control information

Back End

Table 6 - requirements for back end controlling a car

ID	Priority	Summary
R.31	MUST	Accelerator causes forward motion for the vehicle
R.31.1	MUST	Acceleration is proportional to accelerator pedal depth
R.32	MUST	Brake causes deceleration
R.32.1	MUST	Deceleration is proportional to the brake pedal depth
R.32.2	MUST	Brake does not cause reversing (backwards motion)
R.33	MUST	Steering wheel angle causes turning
R.34	COULD	Turning at high speeds cause skidding
R.35	SHOULD	If handbrake is applied, skidding is caused
R.36	SHOULD	When skidding, the car slides
R.37	SHOULD	When skidding, the car's traction forces are decreased
R.38	SHOULD	When skidding the car's braking forces are decreased
R.39	MUST	Vehicles should collide with each other
R.39.1	MUST	Vehicles should bounce when colliding with another vehicle
R.40	MUST	Vehicles should collide with the edges of the track
R.41	MUST	Vehicles should collide with any obstacles on the track

3.4. Non-Functional Requirements

Non-functional requirements describe criteria for checking a system's operation, instead of describing exactly what the system does.

3.4.1. Front End

ID	Summary
NF.1	The minimum frame rate must be 25 frames per second
NF.2	The general frame rate should be 30 frames per second
NF.3	Should have a real enough looking track
NF.4	Should have a real enough looking car model
NF.5	Should have easy to understand controls
NF.6	Should have easy to understand displays for all individual car information (e.g. speed)
NF.7	Must run in a web browser

3.4.2. Back End

ID	Summary
NF.8	The minimum frame rate must be 50 frames per second
NF.9	The general frame rate should be 100 frames per second
NF.10	The largest size of a single response should be 5KB
NF.11	Response time for a request should be such that it doesn't affect the frame rate of the front end (<~40ms)
NF.12	Should be able to handle multiple players

Chapter 4. Design

4.1. Technologies

This section discusses the technologies that were considered for this project. All considerations take into account ease of use, flexibility, prior experience and industry standards.

4.1.1. Unity

Unity is a game engine that works cross-platform, used to create 2D and 3D games (Unity, 2018). Many big productions have been created in Unity in the 12 years since its release, such as *Escape from Tarkov*; *In the Valley of Gods*; and *Rick and Morty: Virtual Rick-ality*.

As the project will be required to run in a web browser, there are two options available from Unity, Unity Web Player and Unity WebGL Build.

Unity Web Player

A Netscape Plugin Application Programming Interface (NPAPI) based plugin that runs in web browsers, Unity Web Player works in a similar way to Adobe Flash, giving a web player access to running code on the client machine that allows for more flexible and powerful games.

The Unity Web player has been deprecated as of March 2016 (Echterhoff, 2018). This deprecation is due to many web browsers blocking all NPAPI plugins by default (Echterhoff, 2018). Google states that the speed, security and stability of its Chrome browser will be improved by this removal (Schuh, 2018).

Unity WebGL Build

Unity WebGL is a build option that converts Unity projects into JavaScript content using HTML 5 technologies to run Unity content in a browser (Unity - Manual: Getting started with WebGL development, 2018). It compiles the Unity C# into Common Intermediate Language which is then converted using IL2CPP into C++, which is finally converted into asm.js (an optimised version of JavaScript) (Unity - Manual: IL2CPP, 2018) (asm.js - frequently asked questions, 2018).

This long conversion process is currently the only viable way for a Unity project to be playable in a web browser without changing browser settings. An issue with Unity WebGL is that the increased number of steps between writing code and the final code execution can lead to problems that are invisible in the written code but break the final running code.

Advantages

Unity has a very large following and so finding help with issues is simpler as there are more expert users. There is cross-browser support, so any browser can be used.

Disadvantages

The largest disadvantage is the lengthy process of turning C# code into running JavaScript. Though the process is contained within a single action in Unity, the debugging from JavaScript is much more difficult as the line numbers do not match up and the running code is not the same as what is written.

4.1.2. Classic Web

Classic web elements are the fundamental blocks that general webpages are built from. Every modern web browser can display HTML which is styled by CSS and has JavaScript (JS) run on it. Using classic web would mean using JavaScript to make requests to the server, these requests would be made using Representational State Transfer (REST), a standard for HTTP web services that uses GET, POST, PUT and DELETE requests to change the state of the server (Fielding, 2000).

HTML

HTML (Hyper Text Mark-up Language) is the standard language for webpages and so creating any sort of webpage will use HTML (W3C HTML, 2018). There are various IDEs that help writing HTML, giving the option to click and drag to make a webpage. However, these solutions are designed for creating large responsive webpages, whereas the one that would be used for this project would be incredibly simple, giving a few options with buttons and a canvas to draw the game. This means there is little reason to use a large IDE to produce the HTML, a simple text editor would be used instead.

CSS

CSS (Cascading Style Sheets) are used to implement uniform styling across a webpage (CSS: Cascading Style Sheets, 2018). The use of CSS within this project would be even more minimal than the HTML, as the main aim is to create the functionality, rather than making it pretty. Additionally, the CSS will only improve the aesthetic of the HTML, most of the front end will be drawn by JavaScript and so CSS will have no effect.

JavaScript

Using pure JavaScript with no added technologies such as JQuery, Node.js or AngularJS gives the advantage of not having the large bloat that these technologies include. With a project such as this where the main functionality will be server side and not be running in JavaScript at all, the need for such technologies is small. However, they were still considered in case their inclusion would have improved the project.

JQuery

JQuery is a Javascript library that adds functionality to assist in event handling, HTML traversal and HTML manipulation. It is a standard used by many people creating webpages and is supported by most browsers (jQuery, 2018).

JQuery makes RESTful HTTP requests very simple, using the AJAX requests built in to JQuery. Though this would be useful, there are only a few requests that need to be included in the front end, so the amount of time saved would be small.

Node.js

Node.js is an open source JavaScript runtime that runs server-side, it aims to be an incredibly scalable solution for incredibly large numbers of users concurrently (Node.js, 2018). As with JQuery, there is an amazing amount of functionality that would be wasted on a simple JavaScript application, meaning it would just be bloat.

AngularJS

AngularJS is a JavaScript framework (AngularJS — Superheroic JavaScript MVW Framework, 2018). It “lets you use HTML as your template language and lets you extend HTML's syntax to express your application's components” (AngularJS, 2018). It does this by using data binding and dependency injection.

AngularJS focuses on CRUD (Create, Read, Update and Delete) webpages, where the front end is managing a database in the back end. As there will be no need for persistent data, AngularJS doesn't fit perfectly with the project

4.1.3. Server Back End

Java

Java is a programming language that can be used to create executable applications (What is Java and why do I need it?, 2018). To create a server that can handle HTTP requests, Java needs to be supplemented with another technology.

Java is my preferred language and this project will not be the first web server created with Java so will most likely be the chosen back end language.

Spring

Spring is a framework for Java that enables dependency injection and when using a Spring Boot application, it is very easy to create a web server that will respond to HTTP requests (Spring Framework, 2018). Spring is another framework that I have worked with before and so will likely be chosen.

Other Languages

There are many languages that can be used to create a web server, some are detailed below.

Rust

Created as a thread and memory safe systems programming language, Rust can create a web server, however, like Java, it needs some help (The Rust Programming Language, 2018). Iron is a framework much like Spring that can be used to build a web application (iron/iron, 2018). Currently Iron is not being updated and so should be avoided, but there are many other frameworks that are similar.

Python

Python, like Java and Rust, has many frameworks and libraries that can be added into the pure Python to add more functionality in terms of web applications (Welcome to Python.org, 2018).

4.2. Chosen Technologies

4.2.1. Front End

For the front end, HTML, JS and CSS were chosen to give complete flexibility. Though other frameworks and libraries can be added to reduce the amount of code needed to write each part, the initial design has very little on the front end, with the clear majority of logic being in the back end. The front end will only have logic for tracking key presses, sending simple HTTP requests and drawing the game. As this is all that is needed from the front end there is no need for a large library to be added in to do this.

4.2.2. Back End

The back end will use Java and Spring Boot as it will need to be a server that responds to HTTP requests. The majority of the logic will be in the back end and so this is the most important part of the project for it to be optimised in terms of outputting a decent tick rate for the game.

The dependencies and build of the back end were controlled by Maven (Porter, Zyl and Lamy, 2018). Maven gives a simple way to define exactly what dependencies are needed for the project, as well as how the project should be built into an executable file.

4.2.3. AI Client

The AI client is to be designed and created entirely by the students that are making use of the product built in this project. Because the server will require HTTP requests to control the vehicles, the AI client can be written using many different technologies, the only requirement for it is that it sends HTTP requests. It will be discussed in the Detailed Design from the perspective of making sure that the functionality of the server would allow the AI client to function.

4.3. Version Control

For version control, as the University provides a free Github account, Github will be used for version control. The requirement IDs from Table 1 to Table 6 will be used as branch and commit names.

GitHub is a cloud backed up layer on top of original Git which makes using Git much easier (Build software better, together, 2018) (Git, 2018).

4.4. Development Methodology

The methodology chosen was Extreme Programming as this allows for very short iterations with fast changing requirements (What is Extreme Programming (XP)?, 2018). Changing requirements are expected in this project as the game's feel will provide feedback on the requirements already gathered.

Being able to react to new requirements quickly by creating a new short cycle will mean that the game will continue to feel correct (in terms of the physics etc.) during all stages of the development.

Chapter 5. Detailed Design

The design and implementation are split up into the separate modules: back end server, front end web. Each module had three full iterations where the modules were usable together as a full working product. This meant that if there was not enough time to complete a single iteration, there would be a fully working product from the previous iteration to fall back on. An AI client will not be implemented in this project, though the functionality of it will be tested by manually sending HTTP requests to the server.

5.1. Design Iterations

5.1.1. Server

The focus of the server's iterations was to get the endpoints working for the REST HTTP as soon as possible, then to add in the physics for the race afterwards. Having the endpoints created first will mean that the server's functionality can be checked without the front end, by manually sending HTTP requests.

Iteration 1

The first iteration of the server was to have endpoints for creating and viewing a race and updating the vehicles within that race. There was to be no logic implemented for the race to have any physics. The decision to have no logic was so that all the endpoints could be designed and implemented without running into issues with the physics.

Iteration 2

The second iteration will implement simple physics and include the design of a single track, with the option to add more tracks in future. Being able to extend the final product from this project is important, so having a way of adding more tracks and the potential to add other vehicles is important.

Iteration 3

The third iteration was to implement more sophisticated physics, including advanced collisions, RPM and realistic turning with skidding.

5.1.2. Front End

The iterations of the front end aimed to get the race information from the server and have it drawn as early on as possible. This means that the race logic on the server can be tested from the front end and that the link between the two modules is checked from the start.

The speed of communication and drawing was checked at each iteration to make sure that the minimum frame rate and update tick rate could be achieved.

Iteration 1

The first iteration was to create the foundation for the communications needed to play the game. An XMLHttpRequest object from JavaScript was to be used to send the HTTP Requests and deal with the responses. Each request type (POST, PUT, GET etc.) was going to be added at this point and have simple code to output to the console the request and response. This would mean that each request could be checked and confirmed at this stage, so that debugging could almost completely ignore the possibility of broken communication.

Iteration 2

The second iteration would add functionality to create and draw the races, with no other inputs. At this point there was no intention of making the race playable for a human or being able to search for a race that has already been created. This iteration would allow an AI controller to play the game externally and for it to be shown in a web browser window.

This iteration was designed to include the vast majority of drawing elements, as the whole race would have to be drawn at this stage. Each vehicle would need to be drawn separately.

Iteration 3

Iteration 3 would implement the front-end features required for a human to play the game. This would also include various additions such as speedometer and gear indicator.

5.1.3. AI Client

Though the AI client will not be developed in this project each iteration of server and front end will have effects on the AI client which will be detailed here. These stages will be tested using Postman, an API development tool which allows sending HTTP requests and shows the response, including any headers (Postman, 2018). This tool is useful whilst developing APIs to see exactly what the response is from the API.

Iteration 1

The AI client will be able to send requests to the server and will receive responses. However, the responses will be placeholders and will have no particular meaning at this point.

Iteration 2

In this iteration the AI client will be able to send requests to the server and will get back the game's current state. At this stage the game will be playable for an AI and viewable in the front-end.

Iteration 3

An AI client should be able to connect to a race that has been created in the front-end by a human controller and should get information about the superior physics, such as RPM.

5.2. System Architecture

Each section of the system architecture is kept separate to show the encapsulation of the system. All the requests and responses are labelled and, other than between the users and their front ends, all arrows show HTTP requests or responses.

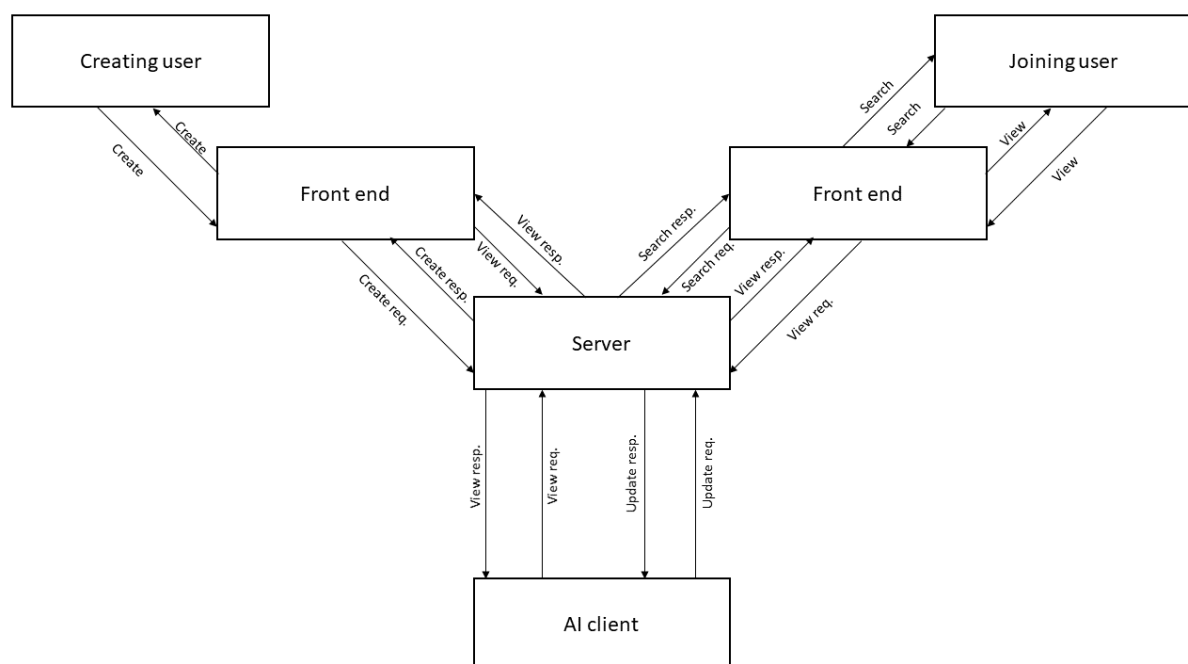


Figure 6 - System architecture diagram showing how the system components interact

5.2.1. Front End

The creating user's front end and the joining user's front end will be the same but from different perspectives. The creating user is whichever user is creating a new race, whereas the joining user will be searching for and joining a previously created one.

The creating user will input details about the race to be created and then a button press will send an HTTP request with these details to the server. The response from the server will be the object for the new race, with only the details that the creating user will need. This includes the ID of the race and the number of cars etc.

At this point the creating user becomes a joining user or goes on to create another race.

The joining user is any user that is using the front end for searching for and joining races. The joining user will request for a search of created races and the server will respond with a list of all races that can be joined. Since all races should be able to have any number of viewing users, all running races should be shown.

5.2.2. Server

The server will run a timer operation for game ticks, which will be done at a frequency of 100 ticks per second. These game ticks will update every vehicle in every race in terms of vehicle physics and collisions. The tick timer will be threaded separately from the threads that will respond to requests so that the number of ticks per second should remain roughly the same throughout, other than an occasional drop if there is extreme load on the server.

5.3. User Interface

The user interface was designed with functionality in mind. The design is minimal and will be implemented initially just to achieve the necessary functionality and then updated to improve the visuals and include more images and animations rather than simple HTML objects.

Other than the race pages, all the screens will have the same layout, with the page's title in the top left to clearly show what can be done on that page. The pages will mostly be built using HTML with some CSS.

5.3.1. Menu Screen

The first screen that users will go to is the menu screen. This will show the game title at the top and buttons to take them to the create a race, join a race, controls and how to pages.



Figure 7 - A mock up for the menu screen

The finished menu screen should look similar to this, preferably with an image from the game as the background. In addition, the finished screen could incorporate some animations, such as cars driving around a track behind the options.

5.3.2. Create Race Screen

The create race screen will allow users to enter details about creating a race, including the host IP address, race name, chosen track and number of cars and laps.

Clicking on the create button will prompt a confirmation message and then take the user back to the menu page. The decision for returning to the menu page instead of joining straight away was taken because the joining of a race could be completely separate from the creation and splitting it gives a common process for the joining of a race.

The mockup for the 'Create Race' screen includes a title bar labeled 'Create Race'. Below the title bar, there are five input fields arranged vertically. The first is 'Host IP address' with a text box. The second is 'Race Name' with a text box. The third is 'Number of Cars' with a range of 1 to 8. The fourth is 'Number of Laps' with a range of 1 to 4. The fifth is 'Track Number' with two radio buttons labeled 1 and 2. At the bottom center, there is a 'Create!' button.

Figure 8 – A mock up for the create race screen

5.3.3. Join Race Screen

The race joining screen will have a host IP address text box and clicking search will populate the list of races. When a race is selected from the first list, the second list will populate with the cars available to view as a local car. Choosing a local car will give a view centred on that vehicle. If a local car isn't chosen, then an overview of the race is shown.

The mockup for the 'Join Race' screen includes a title bar labeled 'Join Race'. Below the title bar, there are two input fields: 'Host IP address' and 'Search'. To the right of the 'Search' button is a list of options (Option 1 to Option 6). Below the 'Search' button is a 'Local car' text box and another list of options (Option 1 to Option 6). At the bottom center, there is a 'Join!' button.

Figure 9 - A mock up for the joining race screen

5.3.4. Race Screen

The race screen will have two variations, centred on a single car, or with the track centred.

If there is a single car chosen, then the status of that car will be shown on the right-hand side, the steering wheel's angle, the gear that the car is in and a speedometer. Along the top will be the positions of all the cars, showing who is in the lead.

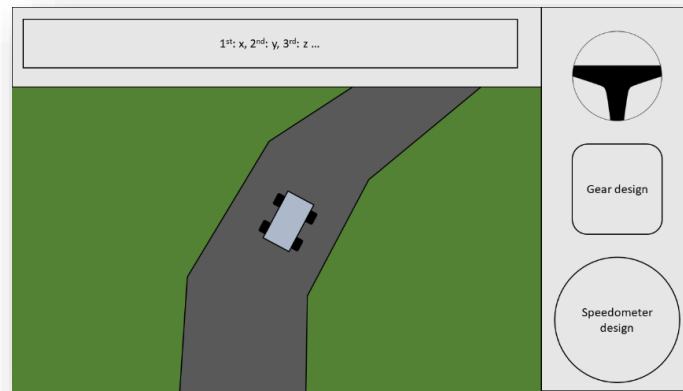


Figure 10 - A mock up for the race screen with a local car chosen

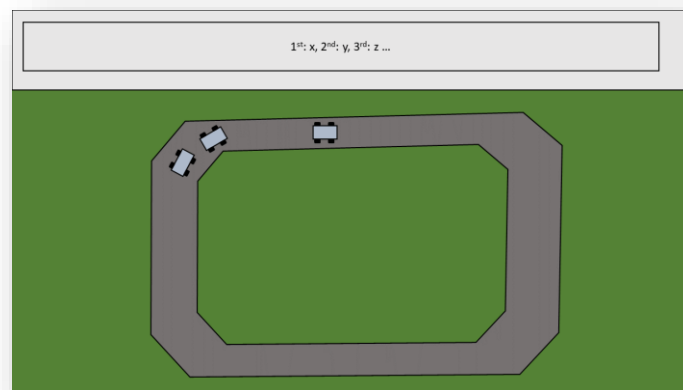


Figure 11 - A mock up for the race screen as an overview

5.3.5. Results Screen

The results screen will show after every vehicle has finished the race. It will detail the order of finishing and the times that each vehicle finished. It will show until the user presses the back button, which will take them back to the menu screen.

The results screen is possibly the most useful screen for users, as they are trying to create an AI that plays the game well, so finishing first or in less time than their last run is important. However, it will not be the most complex page, as it is just displaying the finish times of each vehicle, which will come from the server.



Figure 12 - A mock up for the results screen

5.3.6. Controls Screen

The controls screen is designed to allow users change which keypresses will control the game. These controls will only be used during a race when there is a local car selected.

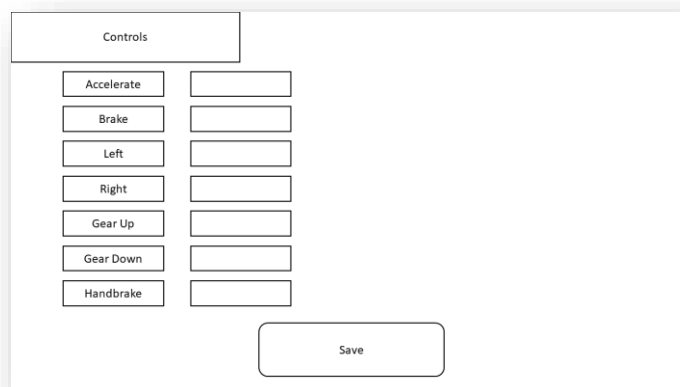


Figure 13 - A mock up for the controls screen

5.3.7. How To Screen

The how to screen will show some information about the game, such as how to create and join a race and how to send updates to the race. It will also explain the difference between an overview race and driving a local car.



Figure 14 - A mock up for the how to screen

5.4. Server-Side Design

The server's design will focus on fast response time to all requests and fast processing of any physics.

The server will be multithreaded to meet these needs for fast processing. The multithreading of the HTTP controller is handled by Spring's RestController class, giving each request a new thread so requests are dealt with concurrently. As almost every request will be a GET and therefore a simple read, the only race conditions that need to be properly evaluated are for updating a vehicle and creating a race. In the case of updating a vehicle, there should only be a single AI updating each vehicle at any time, however the most recent write will simply overwrite the previous and cause no issue other than the previous request's commands will be ignored. The high tick rate will mean that missing a single tick's worth of movement control will make minimal difference in terms of control lost. Along with this, the most recently received request will be left controlling the vehicle, for instance if the steering wheel is turned fully to the left and then no more requests are received, the wheel will stay fully turned to the left. In this way the AI's ability to send requests in a timely manner is also tested. The second race condition of creating a race is even simpler to control. As the creation of a race will be fully completed before adding it to the list of races, only the iterating over the list of races will need to be locked.

5.4.1. Physics

All the physics for the races will be held on the server. This will mean that no user can update their car's position to improve their chances of winning, they will only be able to send updates to the server with vehicle attributes such as accelerator pedal depth.

The iterations detailed below will start at the second iteration, as the first iteration will have no physics implemented. The physics will be split up into longitudinal motion and lateral motion, with longitudinal motion being in the x direction of the vehicle and lateral motion being in the y direction (Illustrated in Figure 15). After the total forces have been calculated for each vehicle in its own reference space, these forces are then converted into the game world's reference space, with x direction being from left to right and y direction being from bottom to top.

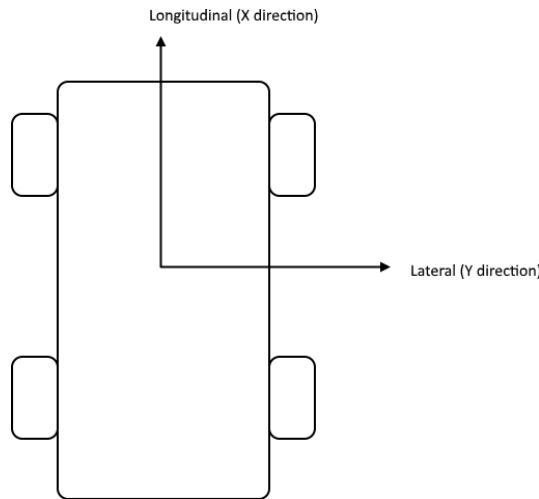


Figure 15 - Diagram showing longitudinal and lateral force directions

For all equations, the following constants and variables are used.

P_a = Variable accelerator pedal depth ($0 \leq P_a \leq 100$)

P_b = Variable brake pedal depth ($0 \leq P_b \leq 100$)

A_{steer} = The angle of the steering wheel ($-\frac{\pi}{6} \leq A_{steer} \leq \frac{\pi}{6}$)

ω = Angular velocity ($\text{rad} \cdot \text{s}^{-1}$)

C_e = Constant for maximum force put out from the engine (N)

C_b = Constant for maximum force from braking (N)

C_{drag} = Constant for drag

C_{rr} = Constant for rolling resistance

m = Mass of vehicle (kg)

Second Iteration

The second iteration's physics will be very simple and are only to show that the car can move around the track and that the rest of the race features are working correctly. There is no reverse feature as this will be added with gears.

Longitudinal Motion

Engine force and braking force at this point are assumed to act directly on the body of the car, rather than on the wheels, this is to simplify the physics. They are both in the direction of the car (x direction).

Equation 1 - Second iteration engine force

$$\bar{F}_{engine} = P_a \times C_e$$

Equation 2 - Second iteration braking force

$$\bar{F}_{brakes} = \begin{cases} P_b \times C_b & \text{if } \bar{v} > 0 \\ 0 & \text{if } \bar{v} = 0 \end{cases}$$

Drag and rolling resistance forces use the vehicle's current velocity, so are not necessarily purely in the longitudinal direction, therefore total force is also not purely in the longitudinal direction for this section.

Equation 3 - Second iteration drag force

$$\bar{F}_{drag} = -C_{drag} \times \bar{v} \times |\bar{v}|$$

Equation 4 - Second iteration rolling resistance

$$\bar{F}_{rr} = -C_{rr} \times \bar{v}$$

Equation 5 - Second iteration total longitudinal forces

$$\bar{F}_{total} = \bar{F}_{engine} + \bar{F}_{brakes} + \bar{F}_{drag} + \bar{F}_{rr}$$

The forces are then converted into world reference space and used to calculate acceleration, velocity and new position of the vehicle.

Equation 6 - Acceleration

$$\bar{a} = \frac{\bar{F}}{m}$$

Equation 7 - Velocity from old velocity and acceleration

$$\bar{v} = \bar{u} + \bar{a} \times dt$$

Where \bar{u} is previous velocity and dt is the time per tick (100 ticks per second is 10ms per tick).

Lateral Motion

The second iteration will have very simple turning mechanisms. The angle of the steering wheel is the same as the angle of the wheels and then this is the rotation of the vehicle. This gives a maximum rotation per second of roughly $\frac{\pi}{2}$ radians. This gives an over-responsive feel to the vehicles, where they may be rotating nearly on the spot, and this is addressed in the third iteration.

Equation 8 - Second iteration rotation

$$rotation = \begin{cases} A_{steer}/100 & \text{if } \bar{v} > 0 \\ 0 & \text{if } \bar{v} = 0 \end{cases}$$

Collisions

There will be no collision detection or resolution in this iteration. This is due to the complexities that collisions present. This iteration is to show that the simple physics work and any confusion from collisions will make this harder to confirm.

Third Iteration

This iteration added in gears, more sophisticated turning and more realistic engine physics.

Longitudinal Motion

The force from the engine in this iteration includes gears and torque. The gear ratios are hard coded and step up in each gear, it also includes a reverse gear that is a negative gear ratio.

Torque is taken from a graph that has maximum torque for RPM values.

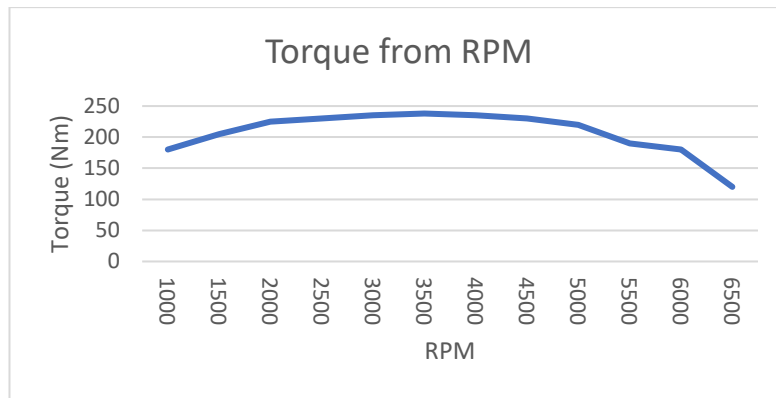


Figure 16 - A graph for finding torque from RPM, (Sarvaiya, 2018)

Once maximum torque has been found for the current RPM of the engine by taking the value for the closest RPM with a point on the graph, engine force is calculated.

Equation 9 - Third iteration engine force

$$\bar{F}_{engine} = \frac{P_a \times T_{engine} \times G}{r_{wheel}}$$

Where T_{engine} is the maximum torque of the engine for that RPM, G is the gear ratio for the current gear and r_{wheel} is the radius of the wheel.

The current RPM is calculated after everything else as it uses the values calculated on the current tick for use in the next tick.

Equation 10 - RPM

$$RPM = \frac{|\vec{v}|}{r_{wheel}} \times G \times \frac{60}{2\pi}$$

Lateral Motion

The lateral motion is calculated from the lateral forces caused by the front wheels being at an angle to the car's direction. If the car has no longitudinal velocity, then there is no lateral motion calculated.

The turning radius of the vehicle is calculated in Equation 11 below, and Figure 17 illustrates it.

Equation 11 - Turning radius

$$R = \frac{L}{\sin \theta}$$

Where L is the wheel base constant (the distance between the two axles).

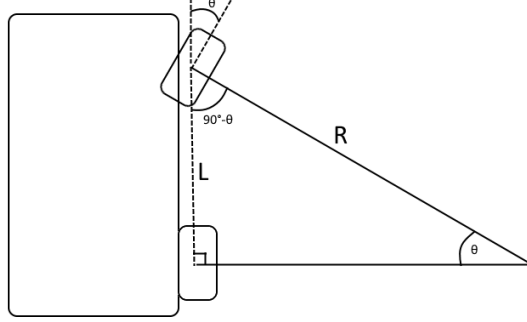


Figure 17 - Illustration for how to calculate turning radius

The radius of the turning circle is used with the vehicle's velocity to find the angular velocity (ω) in radians.

Equation 12 - Angular velocity

$$\omega = \frac{\bar{v}}{R}$$

However, this will only work for slow cornering, as the vehicle will follow the wheels perfectly. For faster turning the sideways slip of the wheels will need to be calculated. Lateral slip is the amount of lateral movement of a body away from its longitudinal direction. The slip angle of a body is the angle between the direction of movement of the body and its facing direction.

The lateral slip angle for the car is shown in Equation 13.

Equation 13 - Lateral slip angle of the car

$$S_{car} = \tan^{-1} \left(\frac{\bar{v}_y}{\bar{v}_x} \right)$$

The front and back wheel's slip angles are shown here.

Equation 14 - Lateral slip angle of the front wheels

$$S_f = \tan^{-1} \left(\frac{v_{lat} + \omega L_f}{|v_{long}|} \right) - \theta \text{sign}(v_{long})$$

Equation 15 - Lateral slip angle of the rear wheels

$$S_r = \tan^{-1} \left(\frac{v_{lat} - \omega L_r}{|v_{long}|} \right)$$

Calculating the lateral forces applied to the wheels by friction with the road is done as below.

Equation 16 - Front wheel's lateral force from friction

$$F_{f-lat} = S_f \times C_c \times W_f$$

Where F_{f-lat} is the lateral force on the front wheels, C_c is the cornering stiffness and W_f is the weight on the front wheels.

Equation 17 - Rear wheel's lateral force from friction

$$F_{r-lat} = S_r \times C_c \times W_r$$

The total lateral force applied to the car is then:

Equation 18 - Total lateral forces on the car from cornering

$$F_{lat} = F_{r-lat} + \cos \theta \times F_{f-lat}$$

Vertical Motion

As the game is in two dimensions, there won't be any tangible vertical motion. However, an important part of realistic car physics is weight transfer, where as a car accelerates, more weight is put on the rear axle than the front one. This in turn creates more friction on those wheels and so decreases the slip. This will also affect the steering, as the front wheels will have less weight on them during acceleration and so less friction to make the car turn.

Equation 19 - Front wheel weight

$$W_f = \frac{L_r m - h F_r}{L}$$

Equation 20 - Rear wheel weight

$$W_r = \frac{L_f m - h F_r}{L}$$

Where W_f is weight on the front wheels, W_r is weight on the rear wheels, L_f is the distance from centre of gravity to the front wheels, L_r is the distance from the centre of gravity to the rear wheels, m is the mass of the car, F_r is the driving force of the rear wheels (there is not F_f as the car is rear-wheel drive) and L is the wheel base distance ($L = L_f + L_r$).

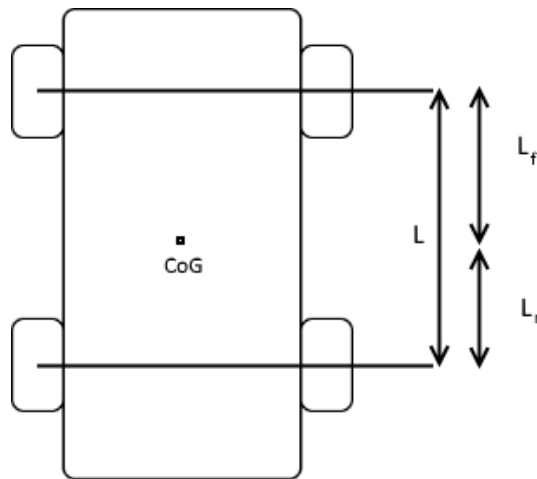


Figure 18 - Illustration of the wheel base constant (L)

Collisions

Collisions will be included in this iteration if there is time to do so.

Collision detection will be done using the separating axis theorem, which finds whether there is any line that separates two objects and if there is then there is no collision or penetration (Chong, 2018).

Collision resolution will be done following the work of Hecker, due to the complex physics involved in rotating colliding polygons (Hecker, 2018).

5.4.2. In-game AI

Though the aim of this project is to create a tool that can be used to aid teaching AI, there is still some need for a built in AI that will be able to race against students who potentially don't have any other students to race against. This AI will simply follow waypoints around the track and will

only be added during the third iteration of the server as an optional extra, as it was not in the original requirements.

5.5. Communication

The communication between the server and the front end or AI client will be done using JavaScript Object Notation (JSON). JSON is easily converted into JavaScript objects using pure JavaScript and can be converted into Java objects using the Spring Controller.

The structure of each JSON request/response is detailed in Appendix A.3, any repeated response has been omitted, as several responses are the same.

Chapter 6. Implementation

This chapter will discuss the implementation and challenges faced whilst implementing the solution described in the previous chapter.

6.1. Server

The server was the first section to be programmed for each iteration, as it can run independently of a front end with ease, using Postman to send requests that control the flow and manually confirming the response is correct.

6.1.1. Timing

The timing of the server was the first issue that needed to be addressed. It was required to run game ticks at a frequency of 50 ticks per second or more, aiming for 100 ticks per second. The more game ticks that were in a second the more precise the physics would be and reduce the amount of stutter that the game would have. Aiming for 100 ticks per second meant having a single tick's total time to be 10ms or less.

Setting up a timer that would start every $1000 \div \text{ticks per second}$ ms would give a static tick rate that can be updated if needed.

Each tick, every vehicle within every race has the physics updated as described in the detailed design. The position of each vehicle is calculated and then collision detection/resolution is applied.

The updating of the vehicle's control variables, such as steering wheel angle, was independent of the tick rate because the update requests were dealt with in a separate thread with no timing restrictions.

6.1.2. HTTP Controller

As the testing was done from Chrome with a file (file://) rather than a web server, Cross-Origin Resource Sharing (CORS) needed to be enabled on the server to allow the front end to access the server's different domain (localhost).

Each method in the controller is strictly one or two lines of code, for a log message and a service call. In this way the controller itself does not handle anything other than passing on the service's response. This is made more useful by calling the service interface's methods, allowing the service class to be removed and replaced very easily and, as the new class must have the same methods as the interface, the replacement will work correctly.

The Spring Restful HTTP Controller handles all multithreading for the HTTP responses, so there was no need to overcomplicate the controller with threadpools. Though there is still the same need to make sure that data races and locks are analysed correctly to keep the game from deadlocking.

6.1.3. Waypoints

The waypoint system was implemented as a cyclic list of locations around the track, each position being at a corner or along a long straight section of track. By checking whether a vehicle is within a certain distance from the next waypoint, the vehicle can progress through them. Once the vehicle is in the waypoint area for the zeroth waypoint, the lap number is increased. If the vehicle gets closer to the previous waypoint than the current one or next one, the waypoint number is decreased.

6.1.4. In-game AI

The AI built into the game works by finding the angle difference between its current heading and the direction it needs to go to get to the next waypoint (Illustrated in Figure 19). Once it has this angle, it decides whether to turn left or right. It keeps the accelerator pressed at all times and does not apply brakes or change gears.

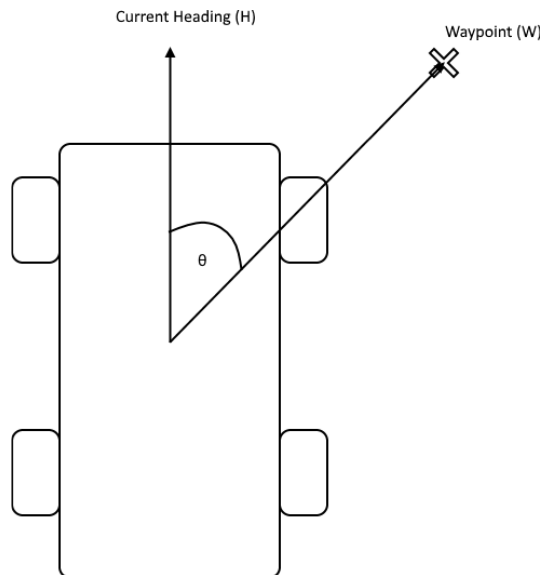


Figure 19 - Illustration of the angle to waypoint

Though not that intelligent, it is able to finish a race by itself and so fulfils the purpose of having something to race against if there are no other players.

6.2. Front end

6.2.1. Drawing

Drawing in the front end was initially handled with JavaScript's two-dimensional canvas-context objects. Shortly after the first stage of programming was completed for the drawing, it was noted that the game would look significantly better and give much more flexibility if images were used instead of objects. This meant that all that was needed were images for each of the parts to be drawn, rotation and scaling.

6.2.2. Overview or Centred

The overview was the first race type created as it involved no zooming or centring of anything in the frame, everything was visible on a single page.

The centred view was much more complex, as it involved zooming in and centring the race around a single vehicle. It also brought in the need for dials and visual representations of the keys that were being pressed.

Zooming was implemented as a variable, so that the zoom could be adjusted if it was found to be zooming too far or not far enough.

Centring the race was done by subtracting the vehicle's position and then adding half the width/height to everything to be drawn. The race was kept with North as the top, removing the need to rotate everything about the local car at the centre of the screen.

6.3. Server to Front end

As the server's endpoints were created and tested for correct form before the front end was built, the front end could assume correctness of the server's responses. The front end was implemented

to visually show the information from the server, with as little as possible processing on the front end. This meant that there was only one place to add and check functionality.

6.4. Implementation Challenges

There were a number of challenges that were not foreseen when designing the project. The challenges and their resolutions are detailed in this section.

6.4.1. Server

Collisions

Collisions in two dimensions with polygons, rather than circles, have very complex physics. The collisions are more difficult to detect, as well as to resolve. During the implementation of the collisions, detecting the collision was effective using the separating axis theorem, however the resolution of those collisions was much more difficult.

The race was designed to continually move forward at 100 ticks per second, but with collision resolution, the exact time where the two objects collide is needed to calculate the proper physics, so there is a need to be able to find an object's exact state at time t . Not having a way to find this information made collisions very difficult to implement. Instead of using correct physics, as they repeatedly did not function, a method of bouncing was devised. Within this method if there is a collision, the centre point of each object is pushed apart by a factor of the amount of intersection there is between the two objects.

Tick rate

The tick rate was set at 100 ticks per second, however the tick counter was within the tick method, so if the number of ticks was lower than 100 per second due to stress on the system, it would still log that there were 100 ticks per second. What it was indicating was that there were 100 ticks per cycle of 100 ticks.

Once another timer was included that ran every second in a separate thread counting the ticks, it showed that the actual tick rate was much lower than 100, sometimes as low as 40. The cause of this was found to be the size of the packets being sent from the server to the front end, discussed in 0 and the locking of the race list in 0.

Locking of the race list

As the race list was being accessed concurrently and Java's ArrayList does not allow concurrent modification (it causes a ConcurrentModificationException), the list of races was locked during iterations.

The lock removed the possibility of a ConcurrentModificationException, however it did cause performance issues as it meant that each race had to wait for all previous races' physics to be completed. This only became a problem after five races were created and being played. This was not tested with multiple front ends for each race, though this would have increased the impact of each race on the performance.

Though not implemented, the proposed solution to this issue would be to have a single thread for each race, which would be kicked off at the start of each tick separate from the list. This way the list could stay unlocked whilst iterating over the list as the object would be separate from the copy of it in the list.

Vehicle Physics

The final iteration of vehicle physics brought about issues with the turning mechanics and reversing. Prior to this the reversing and turning was very simple. In this stage the reversing was implemented as a zeroth gear with a negative gear ratio. The car did reverse correctly when in this gear, so long as the car was not turning at the time. When the vehicle attempted to reverse and turn at the same time, the vehicle would move sideways and occasionally spin wildly and then disappear, as the location became NAN (Not A Number).

To solve this, all variables that were being saved in the vehicle's object were normalised with a maximum and minimum. They were also rounded to 0 if they were $-0.01 < x < 0.01$, which removed any turning movement when the vehicle should have been at a standstill.

6.4.2. Front End

Scale

During the initial drawing phase, where objects were drawn rather than using images, the size of the vehicles was matched to the size of the track, making them look roughly the right size. This strategy required modification when it was noticed that the scale of the vehicles, track and images were all completely different.

To correct this, the images were all set to the same scale, with each pixel being set as 5 pixels per metre. This meant the size of most images needed to be changed, which delayed the next task, however it made the physics feel much more realistic.

6.4.3. Server to Front End

Difference in axis (y is up/down)

The server assumed that the coordinate system was in the orientation of a graph, with the x axis being left to right and the y axis being from the bottom to the top. Unfortunately, this is not how coordinate systems for images work, instead they have y axis being top to bottom.

Network usage

During a stage of testing where the speed of transmission was being assessed, it was noted that the network usage was incredibly high, roughly 138KB/s constant. This was found with only a single race running and with only a single front end. This issue would have scaled up with more races and more front ends requesting information.

The problem was with the size of the response from the server when a front end requested the race information. There was a huge amount of data being passed back that was not used by the front end and in some places, could be abused by someone writing an AI to help them win.

DTOs (Data Transfer Objects) were implemented with a mapper to map the response to a smaller DTO class that only had the important information. This reduced the network usage to 3.8KB/s for the same test.

Chapter 7. System Testing

7.1. Tick Rate

The maximum load was tested for the server and the front end. For the server, the tick rate was required to be a minimum of 50 ticks per second (NF.8) with the aim of being 100 ticks per second (NF.9). For the front end, the minimum ticks per second was 25 (NF.1) with the aim of being 30 (NF.2).

The tests in this section can be used to roughly show the impact of multiple races or multiple front ends on the server and front end's tick rates. The server's tick rate roughly equates to the processing speed of the physics and the front end's tick rate roughly equates to the request/response and drawing time.

Each test was set up from a completely fresh state and with the server being run from the command line rather than an IDE. The front ends that joined the races were in incognito windows and any creation or starting requests were made using Postman.

The results were gathered from the server with command line output with the number of ticks in the last roughly 1000ms, with the exact number of milliseconds being printed as well. The server tick rate numbers quoted in this section will include ± 2 ticks and ± 10 ms, therefore a tick rate of 100 per second is between 98 ticks per 990ms and 102 ticks per 1100ms.

The results from the front end were based on the [Violation] 'setInterval' handler took XXXms error, which is reported when a setInterval method takes longer than it is assigned. As it is assigned 33ms, this gives the 30 frames per second.

Test 1 Tick rate test for many running races

- 1) Start the server fresh
- 2) Send a Create Simulation request
- 3) Send a Start Simulation request
- 4) Check the tick rate
- 5) Repeat steps 2-4

The tick rate of the server stayed at 100 ticks per second for the first 60 races created. After this the test was no longer relevant, 60 races worth of physics were running perfectly and so no more were added.

Test 2 Tick rate test for many running races with a single front end each

- 1) Start the server fresh
- 2) Send a Create Simulation request
- 3) Send a Start Simulation request
- 4) Join with one front end
- 5) Check the tick rate
- 6) Repeat steps 2-5

Test 2 was tested up to 10 races with a single front end in each.

The server stayed above 100 ticks per second throughout.

None of the front ends showed a violation, so their tick rate was kept above 30 ticks per second. However, the CPU usage from the browser was increasing significantly, which is acceptable as there should only be one front end running on a single computer.

Test 3 Tick rate test for a single running race with many front end

- 1) Start the server fresh
- 2) Send a Create Simulation request
- 3) Send a Start Simulation request
- 4) Join with one front end
- 5) Check the tick rate
- 6) Repeat steps 4, 5

0 had up to 30 front ends pulling from the same race information. This was specifically to check whether the lock around the race list would influence the front end's ability to request if there were many of them.

The server kept up at 100 ticks per second, but the front ends started to have interval violations above the 25 races mark. These violations were not worrying as there were only 2 of them across all the races and they were not constant. The cause of these is not particularly important as they were not frequent.

Test 4 Tick rate test for many running races with many front ends

- 1) Start a server fresh
- 2) Send a Create Simulation Request
- 3) Send a Start Simulation request
- 4) Join with 6 front ends
- 5) Check the tick rate
- 6) Repeat steps 2-5

2 races with 6 front ends requesting from each worked properly but adding a third race and 6 more front ends caused some interval violations on the front ends. These violations averaged to 14 in 1000 requests per front end, representing 1.4% of the requests taking longer than 33ms. This is a small percentage but would likely be increased drastically by adding a fourth or even fifth front end.

7.2. Request Performance

The network usage and request performance were tested using SoapUI (The World's Most Popular API Testing Tool | SoapUI, 2018). It can be configured to send each of the requests a certain number of times and measure total time taken or send a certain number of requests a second and check that it keeps up. These tests will show that the server can handle the expected demand of 30 requests per second from several clients and that it can exceed these expectations by using the total number of requests settings.

All the tests were run on a laptop, with the server and load testing software on localhost. This will heavily impact the response times of the server due to internet speed, however the trends will be roughly the same. The hardware used was an Intel® Core™ i7-3537U CPU @ 2.00GHz with 8.00GB of installed RAM running Windows 10.

Test 5 Network usage test for View Simulation request

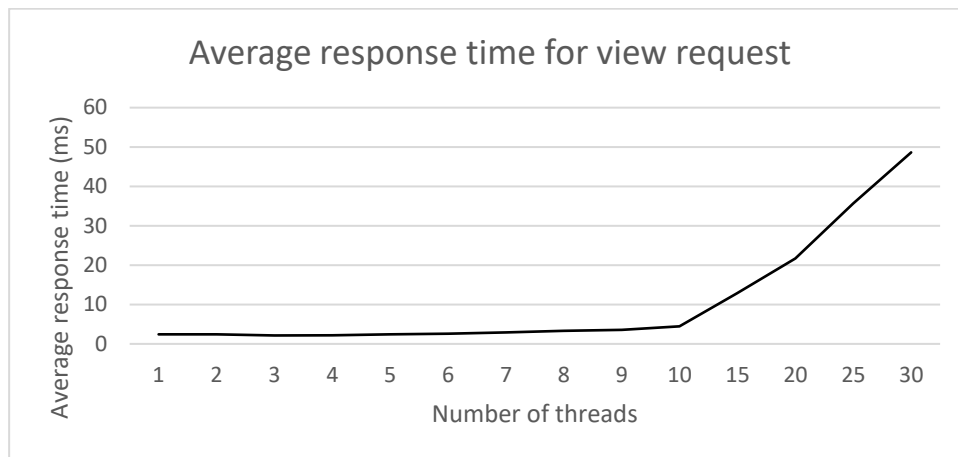


Figure 20 - Graph for average response time for view request

The average response time for view requests kept below 10ms until there were between 10 and 15 threads simultaneously requesting at 30 times per second with a random offset of 50%. The response time is enough to handle up to 25 simultaneous threads without passing the 30ms boundary where the front ends may begin to have a noticeable impact from the slower times.

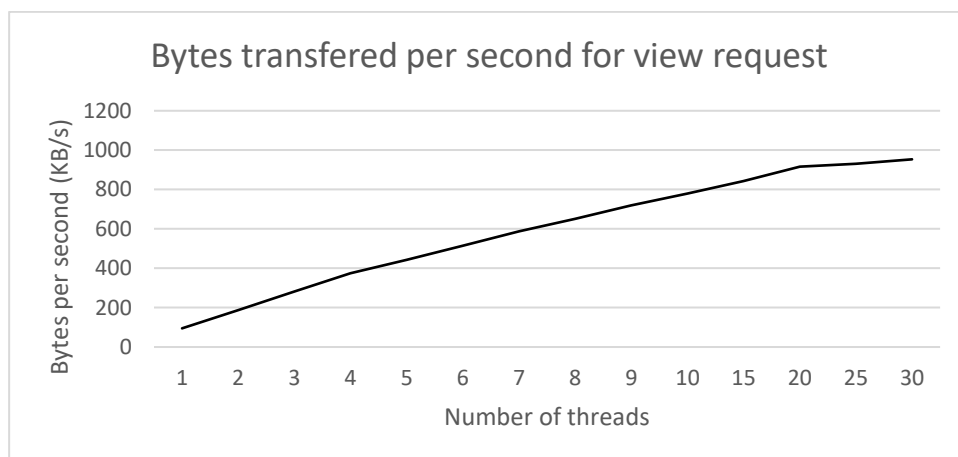


Figure 21 - Graph for bytes transferred per second for view request

The bytes transferred by requesting the race's view is expected to be the largest, and was with even 2 threads being over 200KB/s. However, this value is vastly improved from an earlier iteration where the bytes per second was 4.14MB/s with a single thread. This was due to variables that were not necessary to pass on the view being passed, it is detailed more in 0.

Test 6 Network usage test for Update Vehicle request

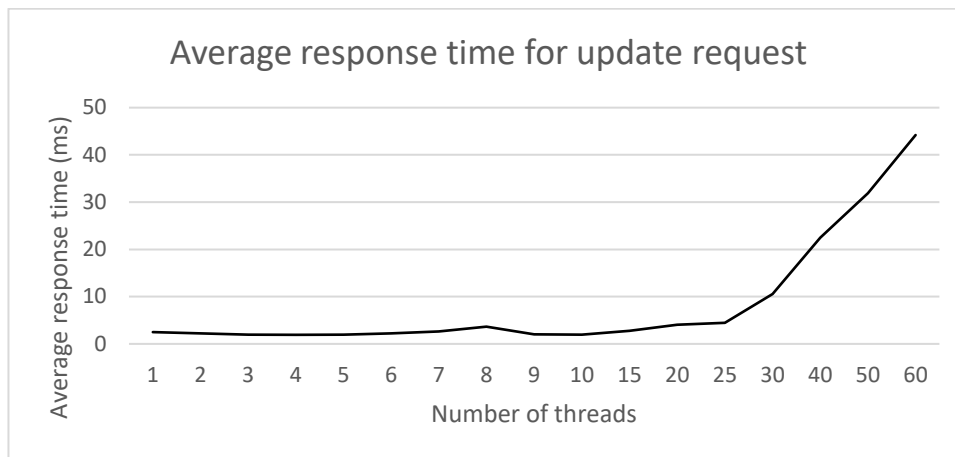


Figure 22 - Graph for average response time for vehicle update request

As expected, the response time for updating a vehicle was significantly less than viewing a race. The amount of work that the server has to do for this request is very small as it is simply updating a few variables before returning. Up to 50 threads could send requests to update vehicles 30 times per second without going over the threshold of 33ms. This means that there could be 5 races with 6 vehicles in each before this would become a bottleneck.

Additionally, the number of update requests from the front end per second is only 10, rather than 30. Though this is not enforced for an AI client.

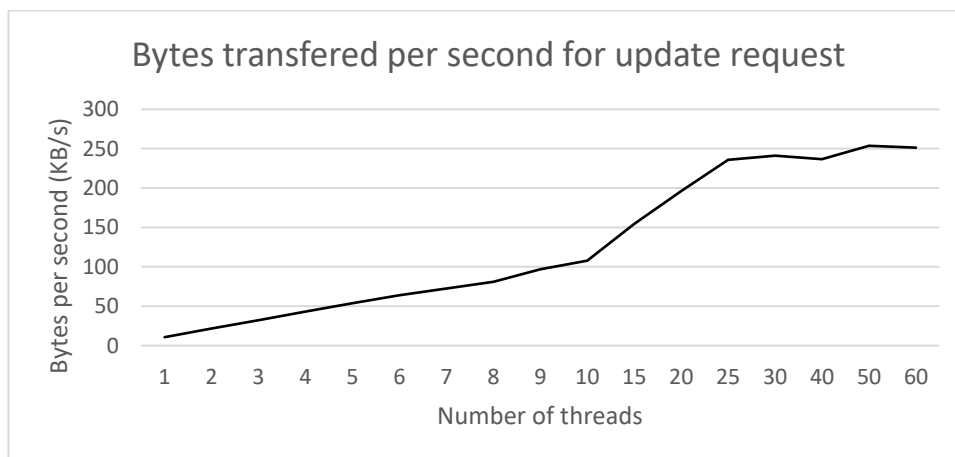


Figure 23 - bytes transferred per second for update vehicle request

Figure 23 shows that for update requests at 60 threads there is less data transferred per second than 3 threads for the view request, as the update request is very small, and the response is also minimal this was predicted. As previously stated, the update requests would be much less frequent than this.

Test 7 Network usage test for Search Simulations request

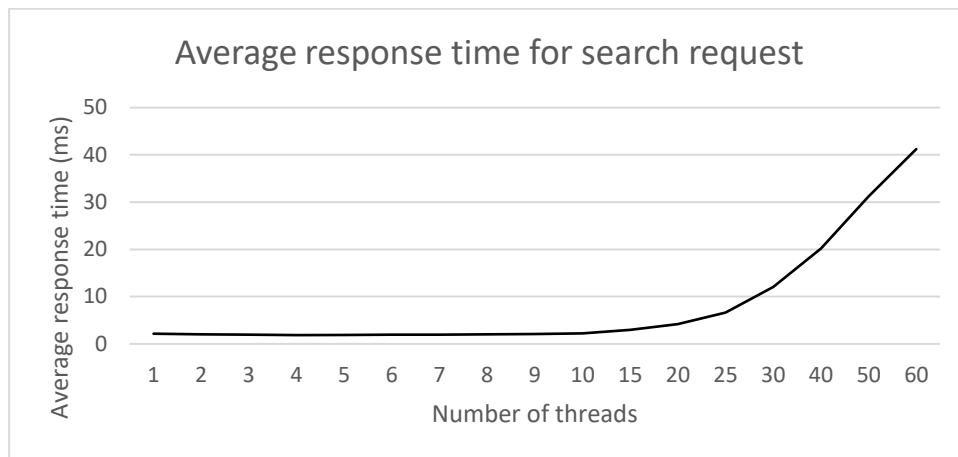


Figure 24 - Graph for average response time for search request

The average response times for the search request were predictably low, as the server only has to cycle through the list of simulations to build the response. With a large list of simulations, this may become a bottleneck, however the number of simulations that would need to be created is so large that it is improbable.

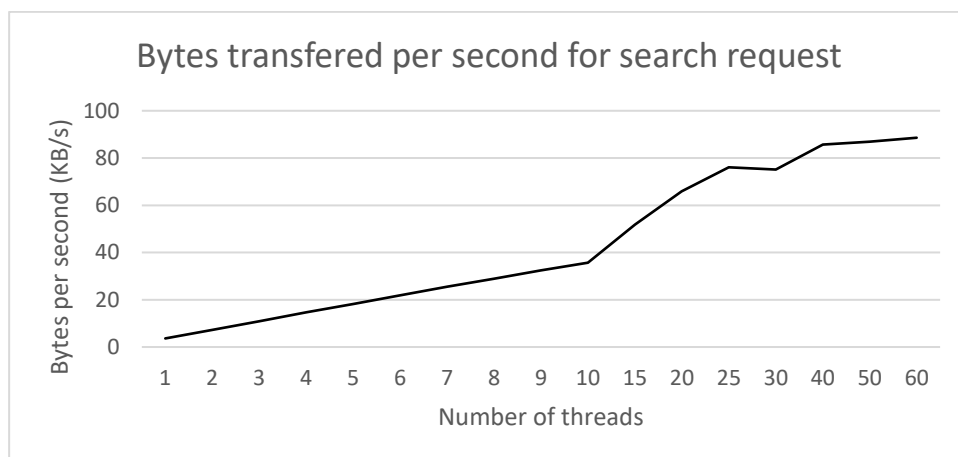


Figure 25 - Graph for bytes transferred per second for search request

The bytes per second for the search request were the smallest of any of the requests tested, and this is due to it having the smallest request and response size for a single simulation. If there were many simulations then the bytes per second for search request would eventually become more than the others, as they will only ever respond based on a single simulation.

7.3. Physics Testing

For testing the physics, each equation from 0 were tested separately using JUnit (JUnit 5, 2018). After this a simple example was worked through by hand and checked against the coded version to make sure that they were very roughly the same. There was a relatively small difference between the two, as was expected due to the result worked out by hand rounding most of the steps. This difference was accepted without further testing because the game does not need to be exactly correct with its calculations, so long as it is generally the same as the design and that the game feels realistic and fun.

Chapter 8. Conclusions

This section will discuss the choices made in the project, the further development that should be done and the viability of the project in the future.

8.1. Retrospective

The retrospective will look at any choices or assumptions that were made during the project that, in hindsight, were not necessarily the best option.

8.1.1. Technology Choices

The technologies chosen for this project were decided heavily for their use in industry, as using these technologies will provide important experience with them. However, choosing Unity to create the game would have reduced the workload considerably. Instead of building a server and front end separately, Unity would allow creating the entire game and physics within a single project.

Reducing the amount of work needed to achieve a working product is important in industry, though for this project was much less important than the experience and learning gained.

After choosing to build the front end using JavaScript, HTML and CSS, there was the decision to not use node.js. Once again this would have reduced the workload a small amount and would have been slightly more useful in future, as node.js is often used in industry.

8.1.2. Library Usage

There were very few libraries used for the server outside of the pure Java libraries, only Spring, log4j and JUnit were added (Spring Framework, 2018), (Log4j – Apache Log4j 2 - Apache Log4j 2, 2018), (JUnit 5, 2018). Adding more libraries to handle the physics and collisions would have removed most of the time that was spent on these sections of the server. For example, dyn4j is a library that handles collision detection, collision resolution and other physics (dyn4j – Java Collision Detection and Physics Engine, 2018).

Using libraries instead of writing the code yourself is a necessity in a commercial setting but would have removed some of the challenge from the project.

8.1.3. Scope Creep

Scope creep is the term for when the scope of the project gradually increases as more is done. It is a difficult problem, especially when there is only a single person designing, developing and testing a product.

This project had a relatively small amount of scope creep. For example, adding a feature such as the server-side AI because the game did not feel complete without it. This sort of feature should really have been added in a future version instead of being built into the product at this point.

8.2. Further Development

Further development on this project is very likely, and the key areas are discussed here.

8.2.1. Improving Playability

Though the game is playable, there is a definite lack of excitement whilst playing. This is most probably due to some features not being completed, such as a start screen with a countdown and a finish screen with the times of each vehicle. Features that were not in the requirements, for example vehicle noises for skidding and the engine revving, would also add a large amount of user engagement.

8.2.2. Creating an Extendable AI Client

In the product's current state there is no AI client, meaning that anyone planning on writing one must start at the very beginning, including finding a way of sending HTTP requests to the server. If there was a template AI client that handles the HTTP and abstracts it into a simple form, the user would be able to start writing code for an AI immediately. In a classroom scenario, it is extremely important to get to the essence of the topic quickly, in this case writing the AI, as there is always a limited amount of time in a lesson.

Extending this idea to a client that uses drag and drop, instead of code, would bring the age and skill requirements of students down significantly. This technique has been successful in recent years, with examples such as code.org, a non-profit organisation attempting to expand access to computer science in schools (Code.org: Anybody can Learn, 2018).

8.3. Originality of product

Though there are many other games used as teaching aids, and several of them designed to teach AI, this project appears to be the first AI teaching aid that does not require building the AI into the original game to any degree.

The University of Bath uses a Unity game that implements Parallel-rooted, Ordered Slip-stack Hierarchical (POSH) dynamic plans for a coursework on an AI unit (Theodorou, 2018). This is a final year unit and so requires programming, however the new AI code is written directly into the game. Because of this there are many variables and methods that can be exploited in an unfair way, even if they were perfectly encapsulated there would still be ways to implement code that plays the game with more advantage than if a human were to play using a keyboard and mouse.

Automobile keeps a large separation between the AI and the game physics, meaning there is no difference between an AI client playing and a human playing, other than the AI being able to have a perfect playing style.

8.4. Commercial Viability

This project sits well within the educational games sector and specifically in the sub-sector of teaching aids for programming. Following successes such as Scratch, there is a clear market for this type of programme (Scratch - Imagine, Program, Share, 2018). Scratch takes programming and simplifies it as far as possible, giving a programming language that can be used by programming beginners of any age. Taking the project and creating a much less complex front end and an extendable AI client as discussed in 8.2.2 would make this a product that could be used in classrooms to teach AI, without the need for any code to be written. Removing the writing of the AI client is the biggest step in this process but would allow students of any age to create an AI that they can see working visually in the game.

Bibliography

Anon, 2018. *LaserSharks.io / Play LaserSharks.io for free on Iogames.space!*. [online] Iogames.space. Available from: <http://iogames.space/lasersharks-io> [Accessed 11 Mar. 2018].

Anon, 2018. *Littlewargame*. [online] Littlewargame.com. Available from: <http://www.littlewargame.com/> [Accessed 11 Mar. 2018].

Anon, 2018. *Home - LEGO.com*. [online] Lego.com. Available from: <https://www.lego.com/en-gb/mindstorms> [Accessed 11 Mar. 2018].

Anon, 2018. *Lego NXT / Lego Robot Programming For Kids / Lego Holiday Camps*. [online] Funtechsummercamps.com. Available from: <https://funtechsummercamps.com/course-descriptions/lego-robotics-level-1> [Accessed 11 Mar. 2018].

Anon, 2018. *Programme & Unit Catalogues - University of Bath*. [online] Bath.ac.uk. Available from: <http://www.bath.ac.uk/catalogues/2011-2012/cm/CM30229.htm> [Accessed 11 Mar. 2018].

Anon, 2018. *Universe*. [online] OpenAI Blog. Available from: <https://blog.openai.com/universe/> [Accessed 11 Mar. 2018].

Anon, 2018. *Docker*. [online] Docker. Available from: <https://www.docker.com/> [Accessed 11 Mar. 2018].

Anon, 2018. *Unity*. [online] Unity. Available from: <https://unity3d.com/> [Accessed 27 Mar. 2018].

Echterhoff, J., 2018. *Unity Web Player Roadmap – Unity Blog*. [online] Unity Technologies Blog. Available from: <https://blogs.unity3d.com/2015/10/08/unity-web-player-roadmap/> [Accessed 27 Mar. 2018].

Schuh, J., 2018. *The Final Countdown for NPAPI*. [online] Chromium Blog. Available from: <https://blog.chromium.org/2014/11/the-final-countdown-for-npapi.html> [Accessed 27 Mar. 2018].

Lian, A., 2018. *Unity 5.3: All-new features and more platforms – Unity Blog*. [online] Unity Technologies Blog. Available from: <https://blogs.unity3d.com/2015/12/08/unity-5-3-all-new-features-and-more-platforms/> [Accessed 27 Mar. 2018].

Anon, 2018. *Unity - Manual: Getting started with WebGL development*. [online] Docs.unity3d.com. Available from: <https://docs.unity3d.com/Manual/webgl-gettingstarted.html> [Accessed 27 Mar. 2018].

Anon, 2018. *asm.js - frequently asked questions*. [online] Asmjs.org. Available from: <http://asmjs.org/faq.html> [Accessed 27 Mar. 2018].

Anon, 2018. *Unity - Manual: IL2CPP*. [online] Docs.unity3d.com. Available from: <https://docs.unity3d.com/Manual/IL2CPP.html> [Accessed 27 Mar. 2018].

Virtualheroes.com. (2017). *Virtual Heroes / ARA / Serious Games / Game-based Training / VR Learning*. [online] Available at: <http://www.virtualheroes.com/> [Accessed 4 Nov. 2017].

Morris, T. (2017). 16 Startling Statistics Forecasting the Future of Artificial Intelligence. [online] Cloudblogs.microsoft.com. Available at: <https://cloudblogs.microsoft.com/dynamics365/2017/02/16/16-startling-statistics-that-forecast-the-future-of-artificial-intelligence/> [Accessed 1 Nov. 2017].

Alban, B. (2017). Creating The Cars Of Need for Speed - Speedhunters. [online] Speedhunters. Available at: http://www.speedhunters.com/2015/07/creating-the-cars-of-need-for-speed/#_presentation-251581 [Accessed 12 Nov. 2017].

OpenAI Blog. (2017). Universe. [online] Available at: <https://blog.openai.com/universe/> [Accessed 16 Nov. 2017].

Anon, 2018. *jQuery*. [online] JQuery.com. Available from: <https://jquery.com/> [Accessed 3 Apr. 2018].

Anon, 2018. *W3C HTML*. [online] W3.org. Available from: <https://www.w3.org/html/> [Accessed 3 Apr. 2018].

Anon, 2018. *CSS: Cascading Style Sheets*. [online] MDN Web Docs. Available from: <https://developer.mozilla.org/en-US/docs/Web/CSS> [Accessed 3 Apr. 2018].

Anon, 2018. *AngularJS — Superheroic JavaScript MVW Framework*. [online] Angularjs.org. Available from: <https://angularjs.org/> [Accessed 3 Apr. 2018].

Anon, 2018. *Node.js*. [online] Node.js. Available from: <https://nodejs.org/en/> [Accessed 3 Apr. 2018].

Anon, 2018. *AngularJS*. [online] Docs.angularjs.org. Available from: <https://docs.angularjs.org/guide/introduction> [Accessed 3 Apr. 2018].

Anon, 2018. *What is Java and why do I need it?*. [online] Java.com. Available from: https://java.com/en/download/faq/whatis_java.xml [Accessed 3 Apr. 2018].

Anon, 2018. *Spring Framework*. [online] Projects.spring.io. Available from: <https://projects.spring.io/spring-framework/> [Accessed 3 Apr. 2018].

Anon, 2018. *iron/iron*. [online] GitHub. Available from: <https://github.com/iron/iron> [Accessed 3 Apr. 2018].

Anon, 2018. *The Rust Programming Language*. [online] Rust-lang.org. Available from: <https://www.rust-lang.org/en-US/> [Accessed 3 Apr. 2018].

Anon, 2018. *Build software better, together*. [online] GitHub. Available from: <https://github.com/> [Accessed 3 Apr. 2018].

Anon, 2018. *Git*. [online] Git-scm.com. Available from: <https://git-scm.com/> [Accessed 3 Apr. 2018].

Anon, 2018. *Welcome to Python.org*. [online] Python.org. Available from: <https://www.python.org/> [Accessed 3 Apr. 2018].

Anon, 2018. *Postman*. [online] Postman. Available from: <https://www.getpostman.com/> [Accessed 13 Apr. 2018].

Sarvaiya, D., 2018. *What is the relation between engine RPM and engine torque?*. [online] Available from: <https://www.quora.com/What-is-the-relation-between-engine-RPM-and-engine-torque> [Accessed 16 Apr. 2018].

Koivisto, J. and Hamari, J., 2014. Demographic differences in perceived benefits from gamification. *Computers in Human Behavior*, 35, pp.179-188.

Anon, 2018. *How BNY Mellon Became a Pioneer in Software Robots - Blue Prism*. [online] Blue Prism. Available from: <https://www.blueprism.com/news/automation/bny-mellon-became-pioneer-software-robots> [Accessed 19 Apr. 2018].

Anon, 2018. *Fraud Protection Solutions for Banks*. [online] Feedzai. Available from: <https://feedzai.com/issuers/> [Accessed 19 Apr. 2018].

Anon, 2018. *Wells Fargo Testing Bot For Messenger Featuring New Customer Service Experiences / Wells Fargo Online Newsroom*. [online] Newsroom.wf.com. Available from: <https://newsroom.wf.com/press-release/community-banking-and-small-business/wells-fargo-testing-bot-messenger-featuring-new> [Accessed 19 Apr. 2018].

Anon, n.d. *The Oxford English dictionary*. [S.l.]: [s.n.].

Porter, B., Zyl, J. and Lamy, O., 2018. *Maven – Welcome to Apache Maven*. [online] Maven.apache.org. Available from: <https://maven.apache.org/> [Accessed 19 Apr. 2018].

Chong, K., 2018. *Collision Detection Using the Separating Axis Theorem*. [online] Game Development Envato Tuts+. Available from: <https://gamedevelopment.tutsplus.com/tutorials/collision-detection-using-the-separating-axis-theorem--gamedev-169> [Accessed 19 Apr. 2018].

Hecker, C., 2018. Physics, Part 3: Collision Response. *Game Developer*. [online] Available from: <http://chrishecker.com/images/e/e7/Gdmphys3.pdf> [Accessed 19 Apr. 2018].

Anon, 2018. *JUnit 5*. [online] Junit.org. Available from: <https://junit.org/junit5/> [Accessed 22 Apr. 2018].

Anon, 2018. *Scratch - Imagine, Program, Share*. [online] Scratch.mit.edu. Available from: <https://scratch.mit.edu/> [Accessed 23 Apr. 2018].

Anon, 2018. *Log4j – Apache Log4j 2 - Apache Log4j 2*. [online] Logging.apache.org. Available from: <https://logging.apache.org/log4j/2.x/> [Accessed 24 Apr. 2018].

Anon, 2018. *dyn4j – Java Collision Detection and Physics Engine*. [online] Dyn4j.org. Available from: <http://www.dyn4j.org/> [Accessed 24 Apr. 2018].

Anon, 2018. *Code.org: Anybody can Learn*. [online] Code.org. Available from: <https://code.org/> [Accessed 24 Apr. 2018].

Anon, 2018. *National curriculum in England: computing programmes of study*. [online] GOV.UK. Available from: <https://www.gov.uk/government/publications/national-curriculum-in-england-computing-programmes-of-study> [Accessed 24 Apr. 2018].

Department for Education, 2014. *Michael Gove speaks about computing and education technology*.

Anon, 2018. *The World's Most Popular API Testing Tool / SoapUI*. [online] Soapui.org. Available from: <https://www.soapui.org/> [Accessed 24 Apr. 2018].

Anon, 2018. *What is Extreme Programming (XP)?*. [online] Agile Alliance. Available from: <https://www.agilealliance.org/glossary/xp> [Accessed 28 Apr. 2018].

Fielding, R., 2000. *Architectural Styles and the Design of Network-based Software Architectures – Chapter 5*. Ph.D. University of California.

- Byrne, M., 2016. Using Games to Engage Students in Inquiry. *PRIMUS*, 27(2), pp.271-280.
- Pozzer, Cesar & Karlsson, Börje. 2007. Teaching AI Concepts by Using Casual Games: A Case Study.. 135-138.
- Garris, R., Ahlers, R. and Driskell, J., 2002. Games, Motivation, and Learning: A Research and Practice Model. *Simulation & Gaming*, 33(4), pp.441-467.
- Sadler, D., 2005. Interpretations of criteria-based assessment and grading in higher education. *Assessment & Evaluation in Higher Education*, 30(2), pp.175-194.
- Theodorou, A., 2018. *RecklessCoding/BOD-UNity-Game*. [online] GitHub. Available from: <https://github.com/RecklessCoding/BOD-UNity-Game> [Accessed 29 Apr. 2018].

Appendix

A.1. Questionnaire

Educational Racing Games Questionnaire

What is the Questionnaire about?

This questionnaire has been designed for use in a final year project, the project is to create a driving game that can be used to assist in teaching AI to students.

How will my answers be used?

Your answers will provide a guide for requirements of the system, though not everything is necessarily going to be used or included.

What about my personal information?

This questionnaire will collect no personally identifying information.

***Required**

1. I identify my gender as...? *

Mark only one oval.

- ☐ Female
- ☐ Male
- ☐ Prefer not to say
- ☐ Other: _____

2. How old are you? *

Mark only one oval.

- ☐ Less than 13
- ☐ 13-20
- ☐ 21-30
- ☐ 31-40
- ☐ 41-50
- ☐ 50+

Skip to question 3.

Top down racing games

3. 1) Have you ever played a top down racing game, with a bird's eye view of the car

Mark only one oval.

- ☐ Yes
- ☐ No
- ☐ Maybe

4. 1a) If yes, please note down as many as you can remember

5. 2) What features would you say make racing games or games in general fun?

Programming and AI teaching

6. 1) Have you ever attended a lesson where you were taught programming or AI?

Mark only one oval.

- ☐ Yes
☐ No
☐ Maybe

7. 1a) If yes, how did the teacher test your knowledge?

8. 1b) How would you rate this technique?

Mark only one oval.

- 1 2 3 4 5
 Not at all effective ☐ ☐ ☐ ☐ ☐ Very effective

9. 1c) Is there anything you would change with this technique?

Powered by
 Google Forms

A.2. Questionnaire Results

Gender	Age	1)	1a)	2)	1)	1a)	1b)	1c)
Female	21-30	Yes	Mario kart	the boxes with items in them	No			
Male	21-30	Yes	Grand theft auto	crashing into other cars	No			
Male	21-30	Yes	Cant think of any top down ones, but Need for speed	Races, winning	No			
Female	21-30	No			Yes	There was a unity game that we added code to. And had an XML plan	4	Some people used exact coordinates rather than relative positions, made it unfair

Male	21-30	Yes	Micro Machines!	Speed of the cars, they need to be fast, but not so fast that its too hard to control them	Yes	We made a chess game that you played against an AI	5	I learned a lot about AI, it just took a while to write the chess game first, could have been given the game to start with and just added in the ai
------	-------	-----	-----------------	--	-----	--	---	---

A.3. JSON Requests

a. Create Simulation Request

```

1. {
2.   "name": "sim_name",
3.   "trackNumber": 0,
4.   "numberOfLaps": 3,
5.   "vehiclesToCreate": [{
6.     "vehicleType": "CAR",
7.     "computer": 1
8.   }]
9. }
```

b. Simulation Response

```

1. {
2.   "id": 0,
3.   "name": "sim_name",
4.   "vehicles": [{
5.     "id": 0,
6.     "location": {
7.       "x": 150,
8.       "y": 225,
9.       "magnitude": 270.4163456597992
10.    },
11.    "directionOfTravel": 3.141592653589793,
12.    "steeringWheelDirection": 0,
13.    "acceleratorPedalDepth": 0,
14.    "brakePedalDepth": 0,
15.    "gear": 1,
16.    "waypointNumber": 0,
17.    "lap": 0,
18.    "finished": false,
19.    "position": 1,
20.    "rpm": 1000,
21.    "speed": 0,
22.    "computer": true
23.  }],
24.   "trackNumber": 0,
25.   "course": {
26.     "rectangles": [{
27.       "x": 1145,
28.       "y": 735,
29.       "width": 859,
30.       "length": 297,
31.       "rotation": 0,
32.       "hexColour": "#323232",
33.       "solid": false
34.     }],
35.     "arcs": [{
36.       "x": 965,
```

```

37.         "y": 630,
38.         "radius": 30,
39.         "startAngle": 3.141592653589793,
40.         "endAngle": 4.71238898038469,
41.         "counterClockwise": false,
42.         "rotation": 0
43.     }]
44. },
45.     "running": false,
46.     "waypoints": [{
47.         "position": {
48.             "x": 180,
49.             "y": 100,
50.             "magnitude": 205.91260281974002
51.         }
52.     }],
53.     "numberOfLaps": 3
54. }

```

c. Update Vehicle Request

```

1. {
2.     "id": 0,
3.     "steeringWheelOrientation": 0,
4.     "acceleratorPedalDepth": 100,
5.     "brakePedalDepth": 0
6. }

```

d. Update Vehicle Response

```

1. {
2.     "id": 0,
3.     "location": {
4.         "x": 150,
5.         "y": 225,
6.         "magnitude": 270.4163456597992
7.     },
8.     "directionOfTravel": 3.141592653589793,
9.     "steeringWheelDirection": 0,
10.    "acceleratorPedalDepth": 100,
11.    "brakePedalDepth": 0,
12.    "gear": 0,
13.    "waypointNumber": 0,
14.    "lap": 0,
15.    "finished": false,
16.    "position": 1,
17.    "rpm": 1000,
18.    "speed": 0,
19.    "computer": true
20. }

```

e. Simulation Search Response

```

1. [{
2.     "id": 0,
3.     "name": "Sim Name",
4.     "trackNumber": 1,
5.     "numberOfLaps": 3,
6.     "vehicles": [{
7.         "id": 0,
8.         "computer": true
9.     }]
10. }]

```