University of Nottingham

G52GRP Final Group Report

gp14-sp-exo

AUDRI: An Automated Driver

Authors:
Lee Haines (lxh02u)
Madalina Stoica (mxs03u)
Syed Afiq Al Attas Syed Khaidzir (sas04u)
Samuel Armstrong (sha03u)
Qi Xing (qxx04u)
Oi Lam Pretty Ophelia Tsang (olt04u)
Jianan Dai (jxd14u)

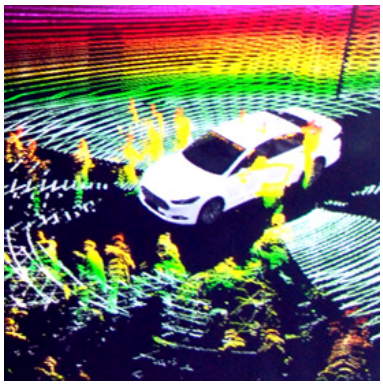
Supervisor:
Dr. Ender Özcan

March 27, 2015

Contents

1 Introduction

The goal of our project is to create an automated driving simulator. Its purpose is to demonstrate good overtaking practice when manoeuvring through traffic. The project involves first creating a game-like implementation followed by applying machine learning techniques to output data when people play the game. After using said techniques we will able to train the system using the data to understand the rules laid down in the specification.



For the machine learning aspect of our project, our supervisor (an expert in the field) strongly suggested the use of WEKA. WEKA is the worlds leading machine learning API written in Java. Initially we were going to use a Java game library to create the game-like implementation of the project, which would combine well with WEKA but would prove difficult to produce a user interface with. Instead we have decided to use the Unity game engine, with its dedicated support for game development, it is a better option for the project. Of course we had to research the possibility of integrating WEKA into a C# environment. After confirming that Unity would work with the machine learning platform further discussion with the supervisor led to an agreement that it was a more viable option.

To train the automated driver we will use the principle of Apprenticeship Learning - learning from an expert[2]. Instead of creating a reward function, we record an expert demonstrating the task that we want to learn to perform. In this context the reward function is successfully maintaining a left lane priority while avoiding collisions with other cars. We will observe each user who plays the game through the recording of a dataset at regular time intervals. This type of machine learning is especially useful in scenarios where the criteria for success are difficult to formulate into a specific set of instructions.Our AI, using reinforcement learning will try to recover this unknown reward function using a set of attributes that describe the context of the game. The algorithm may not recover the exact reward function that the expert was following. However the AI will be determined a success if it can generate behaviour close to that of the expert who created the dataset.

2 Background Information and Research

Our first task as a group involved introducing a set of principles to adhere to. Everyone agreed that we needed to follow an agile methodology. We decided to use Extreme Programming to help develop the code because most of us were already familiar with this idea from previous work. Another two theories we are going to apply are iterative development and rapid prototyping. Our implementation of iterative development included prioritising the requirements to produce the game aspects of the code first. This will allow us to test the

functionality of the game before having to worry about the machine learning (simulation) aspects. Once we are happy to progress we will go back and introduce said machine learning elements of the simulator. Hopefully this will reduce the number of changes to the core game play once WEKA has been introduced because each time the game is changed, a different AI model will need to be generated. We have set ourselves a number of targets on top of the fixed deadlines for the assignment e.g. we wanted a working version of the game mechanics by the deadline of this interim report. We are going to use a PERT chart as a planning tool to judge our progress towards these goals (Appendix A).

Initially when the group project was given, we were told to use Java as it was easy to merge with WEKA, a Java tool. However, the existing libraries and engines used to create games were not very well established. We found a lack of established documentation on majority of the libraries available and they did not have a very active community. The libraries being Slick 2D and libGDX to mention a few. However, researching alternatives we decided on Unity. Although Unity relies on us to learn C#, it provided a much better resource for us to proceed if anything were to occur. This being a well-documented API as well as video tutorials existing, making the coding process that much more secure in terms of murky waters.

2.1 Existing Systems

At the start of the project, it was very important for us to research similar real-world systems to help with the understanding of the task. For this project the key research areas involve machine learning and data collection. Probably the most famous example of artificial intelligence in driving is the Google driverless car. According to Madrigal (2014)[1], the Google team mapped each road that the car would travel by using their Google Map and Street View products and from this created virtual tracks of Mountain View in California. Then, in order to make the self-driving cars take right routes and avoid collision, the team modelled the behaviour of other cars or human by logging every single mile of driving. The lasers and cameras on the Google self-driving car could be used to record all the data of these dynamic objects during driving. Then they took the data and used a machine learning algorithm to classify how different types of objects act in different situations. That data was also used as the training sets for the algorithm of the self-driving car itself, to make it drive in a way that mimicked other cars. 700,000 miles of driving data contributed to the Google algorithm and help it understand how cars act.

Our project is obviously a much simpler task than that of the Google car. However we can learn from the way Google uses machine learning and generate the automated driver system in a similar way. First of all, we can collect large amount of data from the demonstrators when they are using the driving game-like implementation. Then using machine learning techniques, we can classify this data into sets which determine correct driving decisions from incorrect driving decisions which can then be used to create the simulation aspect of the project.

2.2 Machine Learning Software

During the initial meetings with our supervisor, he suggested a strong preference towards using WEKA as a tool for machine learning. However, as there are many open-source software for machine learning, we wanted to look at other machine learning software available and make comparisons between Weka and these software to find the most suitable tool for our project. According to KDnuggets (2014)[6], the two most popular machine learning and data mining software in 2014 were RapidMiner and R. Therefore, we decided to focus our research on these alternatives.

2.2.1 R

R is an open-source software environment and programming language for data analysis and statistic computing. R language can extend a large amount of user-created packages to complete a wide range of tasks, which includes machine learning packages. Therefore R is able to deal with machine learning tasks. The software provides a simple command line shell as its GUI which lacks user-friendliness and the input command has to be R language. As a result, when we tried to use this tool, we were not familiar with the R language, it was not obvious to find out the correct method for data analysis. It means that if we want to use R for machine learning in our project, we have to spend time learning the language first. Compared to Weka, R is more flexible and powerful. For example, using the plot function, R is able to draw graph for the given data in customisable way. In Weka, it will be much more difficult to do the same job. R can also import date in excel format directly, when Weka needs to use converter. However, R is still the more difficult one to learn and operate. Since machine learning in our project is not too complex and does not require too much advanced functions and algorithms for analysis, Weka is the better choice for starters in machine learning like us.

2.2.2 Rapidminer

Rapidminer is another popular data mining and machine learning software we researched, which is based on Java and developed by a German company. The earlier version of the software is open-source, and currently it provides starter version for free with some limitation. The GUI of Rapidminer is user-friendly and attractive thanks to its clear structure and understandable graphic button. In Rapidminer, all the jobs are put in a process. A process should also include some components called operators. Operators are a representation of data source and data mining grammar, such as the decision tree operator and k-Means operator. In process, we should connect the corresponding operators to construct the data flow. The process could then work to produce the result. Rapidminer is powerful for its extensions, including the R-language extension and Weka extension. Although Rapidminer has an appealing GUI and useful functions, we found that Weka is still easier to use. We found that Weka's explorer GUI is still easier to understand. We simply need to import a data set and choose a suitable algorithm to classify it quickly instead of spending too much time connecting operators and constructing a data flow. It means that the procedure to get the result of classification is not as convenient as Weka, and the functions and algorithms in Weka is powerful enough to meet projects requirements.

2.2.3 WEKA

WEKA is a popular piece of software for machine learning developed at the University of Waikato. It provides a rich set of techniques and tools for machine learning. We can use WEKA to produce an algorithm that makes safe driving decisions by using data obtained from previous demonstrators to generalise their driving patterns into a set of attributes.

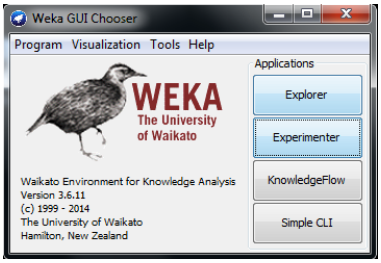


Figure 1: *WEKA provides multiple interfaces*

We need to extract a list of attribute that covers both the state of the environment and the actions of the user. These attributes should be noted at a consistent time interval. Whilst the user is playing the game all of these aspects should be saved into a comma separated text file. However because WEKA uses the ARFF (Attribute-Relation File Format) as input, we will also need to convert the text file using by adding the appropriate header dictating the list of attributes and data.

After reading the data into WEKA, we will select a classifier to produce an algorithm from our dataset. WEKA has plenty of different ways to classify data, we decided on the J48 Decision Tree algorithm to train the data. The advantage of using this decision tree is that the models of trees are simple to read and understand, and it can help determine the best choice of action as the environment changes. Before starting to classify the data, we will use WEKA to randomly split the data into two parts: about 70% for training the algorithm and 30% to test the correctness of the algorithm that we create. The result will display the decision tree model for the supplied data, and the accuracy of classified instances. The aim is to generate a reliable safe automated driving program, the correctness of the model should be about 85% to 95%, above 95% tends to be unrealistic. The best way to improve the correctness of the model is to repeat the training session with different sets of data. Finally we can use the model in conjunction with the game-like implementation to makes decisions for the automated driver.

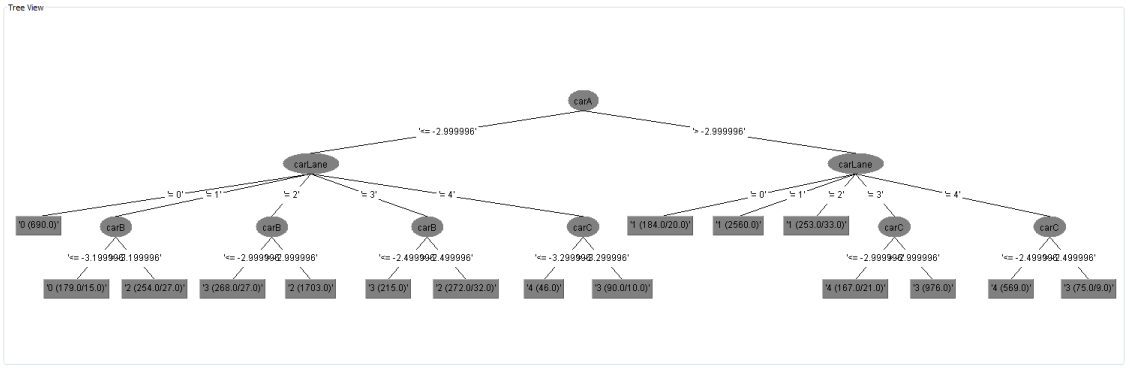


Figure 2: *WEKA provides the option to visualise a J48 tree after creation*

Finally we can input new data into the generated model and receive a classification of the data. The classification will determine an action that needs to be taken by the automated driver. Our project is using Unity to create the driving simulator and Unity uses C#, we need a way of linking the Java tool, WEKA into our C# project. IKVM[8], an implementation of Java for .NET, provides a .NET version of Java Virtual Machine and allows C# to call Java classes and use most of the standard Java API directly. It is Open Source software and is freely available. Most importantly, IKVM can convert a weka.jar file into a .NET dll. This allows us to convert the WEKA mechanics into a .NET library that C# can access like any other shared library.

2.3 C4.5 Algorithm

The J48 decision tree is a WEKA implementation of the C4.5 algorithm which itself is an extension of an earlier ID3 algorithm, both designed by Ross Quinlan.The ID3 algorithm[7] contains three basic ideas:

1. A node represents a choice concerning one of the attributes from the dataset(except from the class attribute). Each leaf contains a potential classification and will be a value of the class attribute. Because of this the class attribute must be a nominal attribute, i.e. a fixed set of choices to classify the data into.

2. Starting from the root node, each subsequent node should split the data based on the most informative attribute that hasnt been considered on this path from the root.

3. The most informative attribute is determined using the concept of Entropy[4]. At each node, we examine the information gain from each attribute and pick the attribute with the highest information gain to reduce the uncertainty of the end classification.Simply we can imagine a formula like:

$$\text{Information Gain} = \text{Entropy Before} - \text{Entropy After}$$

The C4.5 Algorithm updates this algorithm to include support for unavailable values, continuous data ranges and the pruning of decision trees using rule derivation. Recording the y-coordinate of the lowest car in each lane is an example of a continuous data range.

One of the major improvements in the C4.5 Algorithm is the ability to prune decision trees[3]. The ID3 algorithm will continue to grow a tree until all of the training data is classified corrected. This can lead to over-fitting which will reduce of the accuracy of the model on subsequent test data where the class attribute is unknown. C4.5 aims to produce a stronger model by pruning areas of the decision tree. Pruning of the tree is done by replacing a sub-tree with a single leaf node. If a decision rule determines that the expected error rate in the sub-tree is greater than in a single leaf then the sub-tree is pruned and replaced. WEKA allows us choose between pruned and unpruned trees as well as the confidence interval used for pruning.

3 Requirements Specification

For this project, the requirements will be gathered both from use cases and from the specification. These will be used to form a list of functional and non-functional requirements that will serve as a guide for the implementation of the program. The three main actors in this system will be:

- WEKA
- The User
- The Programming Team

The level of functionality has been split into three categories: summary, user and sub-function. These represent the detail that each use case is written to. The scope has also been split into three sections to represent the main objectives of the system: Driving Simulator, Machine Learning and User Interface. A select few use cases have been provided for context, the rest are available in appendix B.

3.1 Use Cases

Goal:A Play-through of the Game
Primary Actor: User
ID: 001
Description:

1. When the application is started, it must wait for conformation from the user before starting. When prompted the cars start to move.
2. The aim of the game is to last as long as possible without crashing the car. The player uses the left or right arrow keys, to move left and right respectively, to dodge other cars on the road.
3. When the players car gets to within a certain proximity of another car this counts as a crash and the game ends.
4. The cars stop moving and a game over screen appears.

Pre-Conditions: None
Scope: Driving Simulator
Level: Summary

Goal: Evaluation of User Skills
Primary Actors: User, WEKA
ID: 003
Description:

1. The system should help a user to evaluate their safe driving skills against AUDRI. When the user crashes the 'game over' screen should appear with some statistics.
2. The statistics should explain to the user how they got themselves into the situation of a 'bad state' that lead to the crash as well as points gained while playing and time.
3. The simulator could offer a replay of the same situation but with the automated driver controlling the users car to show them how they should of dealt with the situation.
4. The user will use this feedback to become better at the game and become a safer driver for the next time they play the game.

Pre-Conditions: 001, 002
Scope: Machine Learning
Level: Summary

Goal: Starting The Program
Primary Actor: User
ID: 004
Description:

1. When the user runs the program a window should appear.
2. In the window there should be a simple menu, detailing the different options available to the user.
3. The user should be able to start the game from the options menu.

Pre-Conditions: None
Scope: User Interface
Level: Sub-function

3.2 Functional Requirements

1. The system needs to involve implementation of C# based tool.
2. The system needs to be 2D based.
    2.1. The view of the road should be top down.
    2.2. The users car should stay near the bottom of the screen allowing enough time for the user to see what is coming.
    2.3. The users car should firstly appear at the bottom of the screen, in the middle lane of the road.

3. The simulator should be displayed as a car driving on a highway with three lanes.
    3.1. The left lane should always be priority for the users car.
    3.2. The GUI should display three lanes and sufficient space on either side for the car to go off-road.
    3.3. The car should be able to go off-road, turning the three lanes into 5 different options. ”
    3.4. The two extra lanes representing the off-road route should have the colour green, embodying grass.
4. A scoring system should be implemented by the following rules:
    4.1. The user gains 2 points with each second the user spends in the left lane.
    4.2. The user gains 1 point with each second the user spends in the middle lane (0 points for the right lane).
    4.3. The user loses 2 point with each second spent in either of the off-road lanes.
5. The 'enemy' cars should have a number of characteristics:
    5.1. All should have the same colour, this colour should be easy to distinguish from the user's car.
    5.2. The same exact dimensions and shape as the user's car.
    5.3. Should randomly appear at top of the screen.
    5.4. When they appear they should fall down in the direction of the users car, simulating overtaking, without changing the lane they initially appear on.
    5.5. Other cars should never crash into each other (all speeds will be fixed).
6. The simulator should recognise a crash.
    6.1. When two cars get within a proximity the system should register a crash (essentially when the objects touch).
    6.2. This should be visibly displayed on screen, so the user becomes aware of the crash.
7. A game over screen should appear when the user fails.
    7.1. The system should give statistics about the users run.
        7.1.1. The screen should show the length of the run.
        7.1.2. The screen should show the amount of points the user has gained while playing the game.
    7.2. The system should give the option of reviewing the run with the automated driver in charge.
    7.3. The user should be given the option of playing the game again.
8. The simulator will stop after a fixed period of time.

8.1. If the player has avoided crashing for three minutes the game ends.

9. The difficulty of the simulator should scale accordingly.

9.1. The number of enemy cars on the road should increase.

10. The system needs to integrate with WEKA.

10.1. The program should save the environment at regular intervals.

10.2. This data should be saved in a format that can be easily run through WEKA.

11. The user should be able to create and playback a model of their behaviour.

11.1. The program should prompt a name to be used as the name of the dataset.

11.2. From the options menu the user should be able to choose from a directory of models including any models created in the same play session.

3.3 Non-Functional Requirements

1. The program should be efficient enough to run smoothly and successfully on the lab computers.

2. The program should have a simple and clear GUI that should be able to be interpreted by all adults.

3. The program should give inexperienced users clear instructions about how to start and control the simulator.

4. The program should be stable as possible and if the program crashes it should do so gracefully while saving as much data as possible.

5. The difficulty of driving should be adjusted to a suitable situation in order to obtain useful data.

6. The program should respond to input from the keyboard/mouse.

6.1. The left and right arrows should be used to move the users car left and right.

6.2. The user should be able to quit the game early or pause the game.

6.3. The options can be manipulated using the keyboard or mouse.

7. The system should produce no sound of any kind (music/other system sounds).

4 Initial Design

Before implementing our program, we needed to consider the design. We were very conscious about the importance of this stage. We knew that rest of the project was going to be based on the decisions we made at this point. We wanted to provide a smooth transition when we added the machine learning algorithm into the game therefore we need to focus on a game design that would let us extract all the needed features for WEKA.

4.1 Paper Prototype

First of all, we had to consider the 2D design of a road crowded with cars that had to follow a set of rules.

1. There are supposed to be two types of cars visible on the screen: the users car and opposition cars.
2. There are supposed to be three main lanes, representing the road. Also, there are two additional ones, that would represent the off-road alternatives for the user's car.
3. The user's car will appear at the bottom of the screen, on the left lane of the main road and will only be able to be moved right or left, but never up or down.
4. The enemy cars will appear at the top of the screen, on the three main lanes. They are not allowed to appear on the off-road lanes and will only be able to go down, remaining on the lane they initially appear on.

One of the first design choices we had to make involved the movement of the cars. The specification states that the users car has a fixed speed higher than all of the other cars. This simplifies the driving simulator so it focuses on overtaking. We decided that instead of having all the cars moving at once, we could keep the users car stationary at the bottom of the screen. Other cars appear at the top of the screen and drop towards the car. This provides the illusion that the users car is the fastest and is catching up to the others.

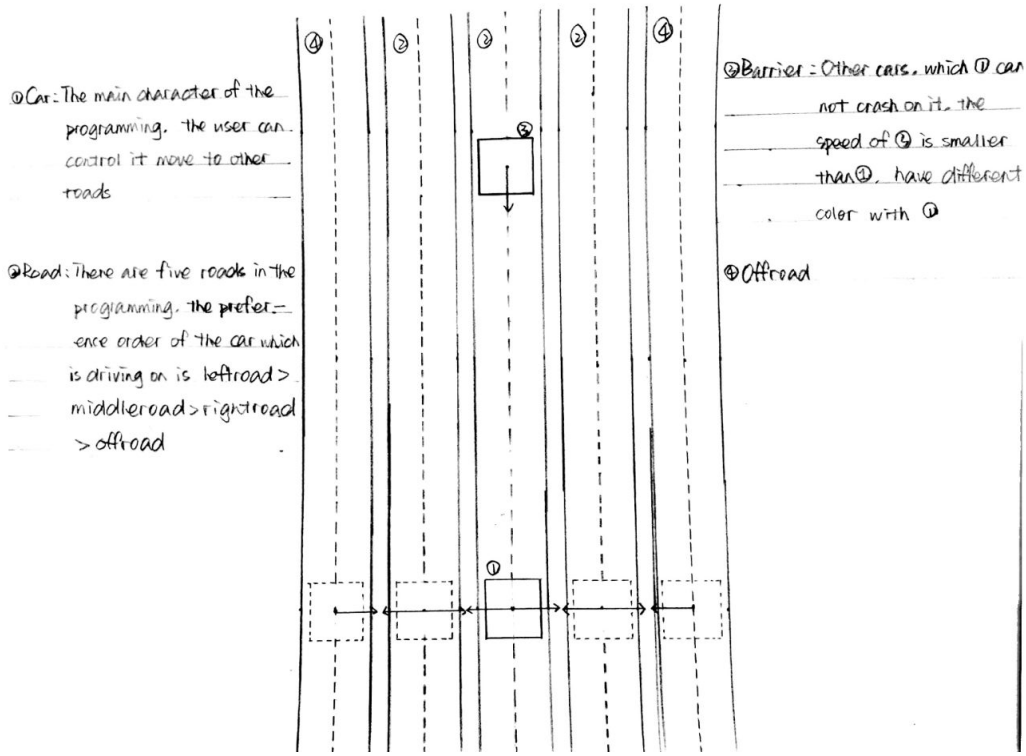


Figure 3: *First Paper Prototype*

Figure 4: *Prototype of Potential Scoreboard*

Another key area of discussion during the design phase revolved around how we can get the user to follow safe driving practices. We decided the best way to implement this was through a scoring system. The scoreboard will give the user a visual aid to correct their driving decisions. It will act as a deterrent to prevent drivers from using the off-road lanes too frequently and reward drivers who stay in the left lane. The scoring system will also act as an evaluation tool, allowing the user to check their performance against previous results and measure improvement.

The biggest showcase for our project will be the open day with lots of people playing the simulator for the first time. Therefore we need to make the system accessible for new people. We decided on keeping all the cars the same size and shape. To design with this in mind, we wanted to simplify a lot of the colours to effectively differentiate objects. As more cars get introduced we wanted the user to be able to pinpoint their car and incoming threats.

Figure 5: *Sophisticated examples of the graphic style*

However at that point in the project we didnt want to focus our attention on designing a beautiful set of sprites and graphics when a lot of the features were likely to change. Therefore we decided to go with a very simple blocky style. The cars are represented as small squares (blue for the player and red for the enemies and the lanes as rectangles stretching from the top of the screen to the bottom. To identify the lanes they were coloured appropriately with grey for the road lanes and brown for the off-road lanes.

4.2 Software Prototype

The next stage of design was to take our theoretical paper prototypes and translate them into the unity engine. We quickly created a mock example of the game screen including the five lanes, the users car and an example of a enemy car spawning at the top of the screen. This prototype was very useful in helping us flesh out the logistics of the game, such as the car size, fall speed of the cars and the vertical/horizontal length of the lanes.

This example was very much used to help us associative ourselves with C# and test some of the new features of Unity that we were unfamiliar with.

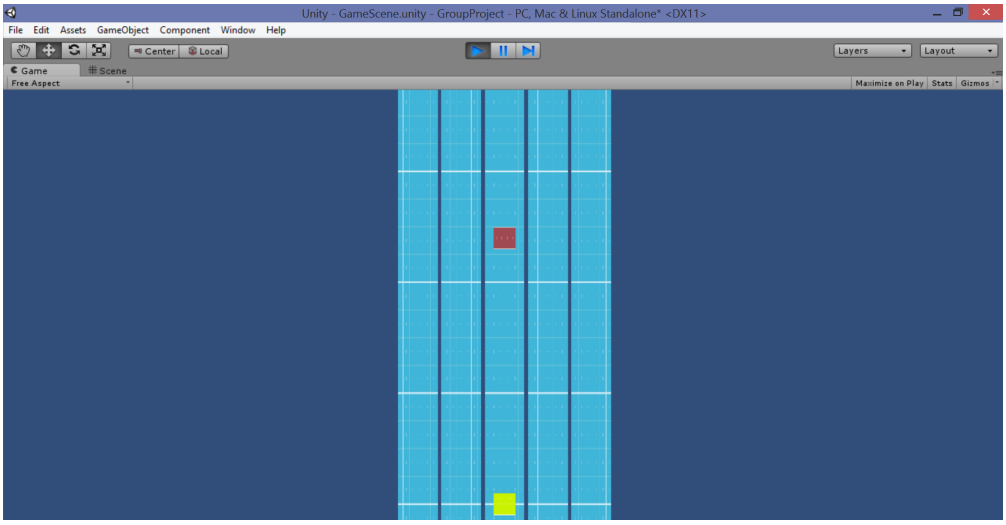


Figure 6: *Software Prototype*

4.3 Prototyping The Simulator

After designing the game mechanics, the group decided it would be beneficial to start thinking about the AI design. After deciding on WEKA and the J48 decision tree as our preferred tools we had to design the system that connected these into the C# project. We begin by drawing an Activity UML Diagram which depicted the potential flow of control of the finished project.

Using the diagram we managed to extract 4 steps needed to introduce the machine learning technology.

1. Play the Game to create a dataset
2. If you want the simulator to be in control. Build a J48 Decision Tree over the dataset by calling WEKA directly, when the game is started.
3. Before logging the attribute set, feed the attribute set (minus the class attribute) into the decision tree
4. Take the result and use it to determine the movement of the car

Finally we decided to try and develop the game with an idea of the attributes we would want to extract. Without a logging system it was hard to test our theories but we predicated a dataset might contain some of the following features.

- The current lane of the car
- the y-coordinates of the lowest enemy car in each lane
- the class attribute: decision made by the user: left, right or stay

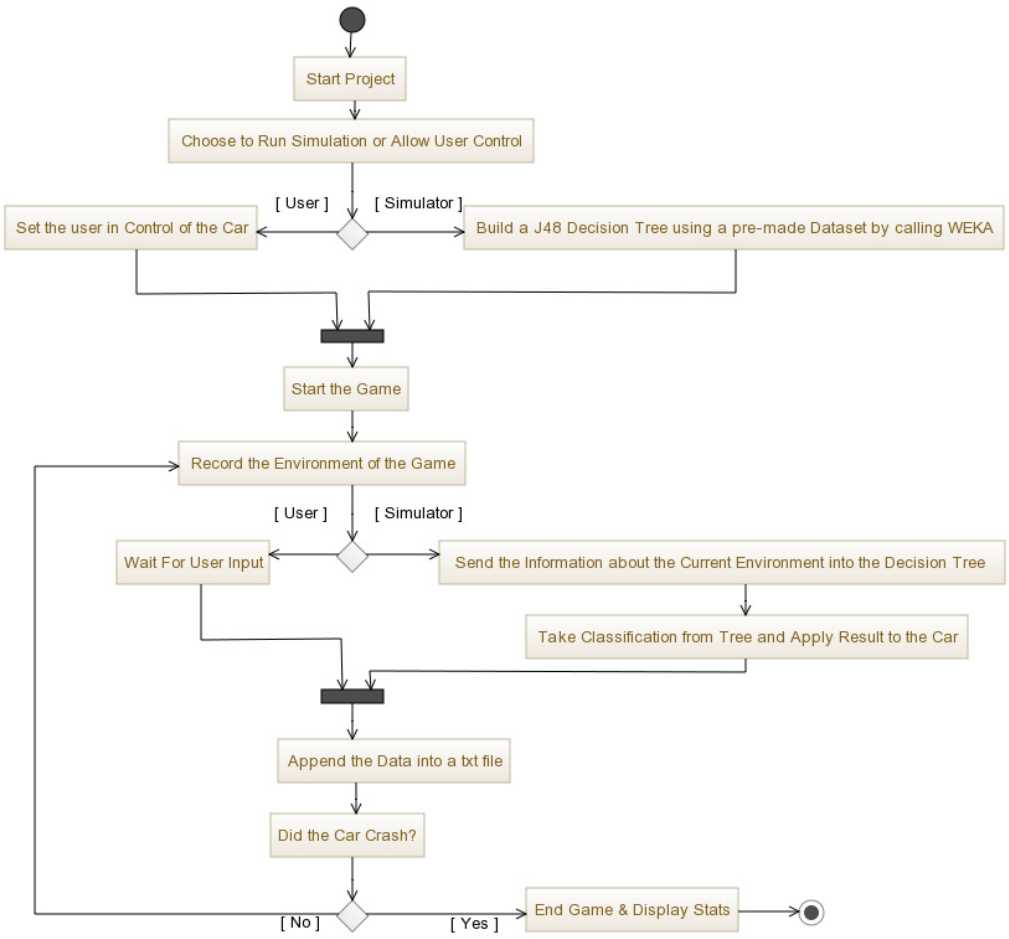


Figure 7: *UML Activity Diagram*

5 Key Implementation Decisions

5.1 Programming Language

Choosing the programming language was an important decision, but one which we had little control over. Initially the plan was to use Java, as previously stated, however after deciding to use Unity instead to handle the game aspect of the project this forced our hand to use C# and Javascript as our main languages. Unfortunately the most commonly used version of WEKA is the Java implementation and so this decision to use C# although helpful when designing the interfaces came with the cost of possible issues arising with the machine learning later on.

Late on in development we made the switch to Unity 5. The main reason why we changed to Unity 5 is the fact that it provided a standard platform as well as it fixes some bugs we had with Unity initially. Some of these problem we had were caused by team members working on different versions (4.6, 4.0). Unity 5 fixed these bugs as well as providing a more stable

environment and a standard that all of our programmers can use. This is also imperative for backwards compatibility as our previous code will not run on Unity 5 if kept the design based on the Unity 4 settings.

5.2 Code Structure

For the project the code was split into a number of scripts, each contained within its associated object. It was important from the start that we try to adhere to the Unity good coding practices[10] to ensure our code was of good quality. The main practice we followed was Single Responsibility, which ensures that each class should ideally be responsible for one task.

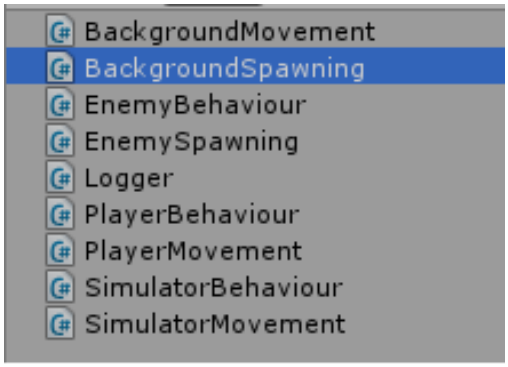


Figure 8: *Break down of the scripts used to implement the system*

Firstly we split the game into its associated objects, with each object containing all its associated scripts. For scripts which it wouldnt make sense for it to be with the object itself we also created a GameManager object. For example for the Enemy and Background prefabs they are designed to be spawned in at the top of the screen and de-spawn at the bottom. In this case it would not make logical sense that the script that spawns such prefabs to be contained within the objects themselves, thus it was attached to the GameManager object. In regards to the Player and Simulator objects we split their code into two scripts each; movement and behaviour. Although these scripts could have perhaps been split down into more, it would have made the code more awkward to understand and longer to read. The reason that we used two scripts instead of attempting to push them into one was not only to adhere to the Single Responsibility principle, but also that we needed multiple Update functions (of which you can only have one per script). Although this could have been handled through co-routines again it would have made the script more complicated than it needed to be.

5.3 Point System

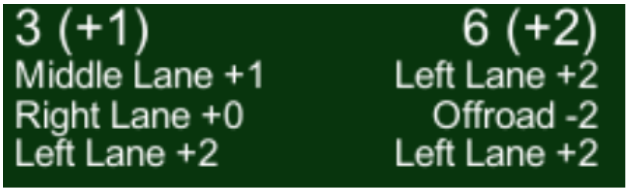


Figure 9: *New scoreboard set up for player + simulator*

Another important decision we had to make was how the games point mechanic would work. Although we were constructing a simulator we needed a system for rewarding what we considered good behaviour. For the beta version of the project we decided on a simple point system whereby points are awarded or negated every second. If the user was in the left lane, the desirable lane, then they would be awarded 2 points per interval.

If the user was in the centre lane then they would be awarded 1 points per interval and 0 points for the right lane. If the user was in either of the outside lanes, the off road lanes, then they would be deducted 2 points for each time interval they remained there.

5.4 Difficulty and Progression

Another decision we faced was how we would approach difficulty in the game. After multiple discussions we decided that the difficulty would be controlled by two factors. One such factor being an increase in the spawn rate of enemy cars. Initially the design was to implement uniform speeds on each lane and increase them as the difficulty progressed in the game. This was thought up when a demo was shown at the first meeting. However to imitate a realistic road situation we decided the left most lane to spawn faster to imitate a slow lane, the middle being a moderate speed and the right being the fast lane with the slowest speed.

5.5 Graphics

Once the majority of the projects background functionality had been completed we agreed that it was time to update the graphics from the basic prototype blocks we had been using. The first decision was which graphic style we wanted to use. With many choices including cell-shaded, neon (Tron-like) and high resolution amongst others it was a fairly difficult decision. However we agreed that the main focus of the project was on the simulator and the machine learning and for that reason we wanted a simple and clean graphics style. We decided that the 16-bit style suited the gameplay well and we would be able to achieve good quality sprites using it. It avoided complications such as the high detail required to make cars look realistic and allowed us to spend more of our time with the functionality.

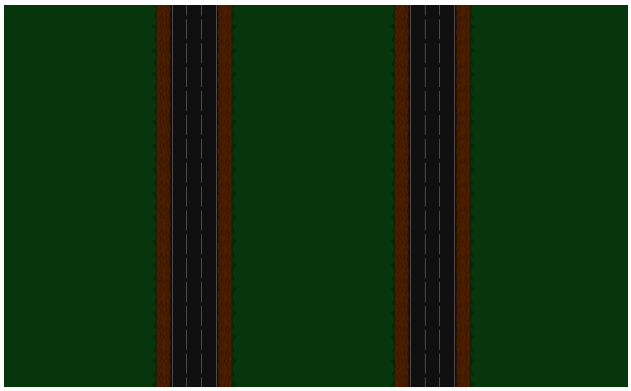


Figure 10: *The animated background representing two sets of lanes*

Once we had decided on the graphics style the first object to create was the generic car (as this would then go on to determine the road size). To do this we used a simple top-down view template of an actual car and designed a small sprite from it. This sprite was then increased in size beforehand with smooth scaling disabled to maintain the pixel style rather than scaling it in unity which caused blurring. Once increased in size we were able to very easily change the colour of the car to reflect its role by changing its HSV value. Of course retaining the blue as the players car, the classic red for an enemy object (as inspired from many games) and orange for the simulator. After the cars were designed we were able to use them as a template to draw the roads and get the width right. Instead of the old system whereby each lane was a separate sprite, which would have

Figure 11: *Graphics representation of the user, simulator and enemy cars from right to left*

been much more inefficient when it came to animating a moving background, we instead decided to combine all the background sprites into one. The usual colours were chosen to reflect their real life counterparts with roads retaining their dark grey, off-road tracks having a mud brown colour and the grass having its usual green.

5.6 User Interface

The User Interface was split into three menus representing the start screen, game over screen and the pause menu. In the beginning, we started with different scenes for the Start Menu and for the Game Over situation, as well as for the main game. Using UI buttons, we gave the user the options to start a new game or quit the existing one. A pause function was then implemented, using transparent UI text and buttons, that would be activated when pressing either 'p' or 'P' on the keyboard. Of course, the existence of additional scenes made linking possible events to appropriate actions easier for us. For example, adding a new scene for the situation in which the user looses the game, meant we would only have to link the collision to it. Then, a different image would pop up. Here, the user would receive some statistics detailing their behaviour such as: points gained, time spent in certain lanes and time spent playing. All this would be done without us having to worry about what happens to the game next, as the game scene would no longer be seen and would no longer affect the user's experience.

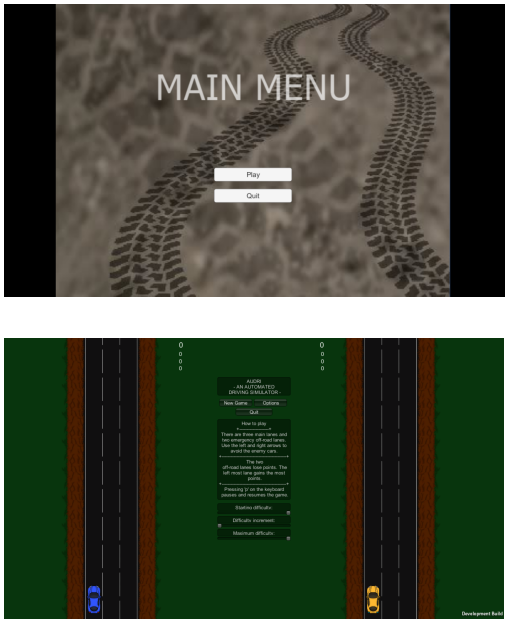


Figure 12: *top: the old main menu, bottom: the new main menu to include the difficulty sliders*

However, we wanted the user interface to be clean and consistent. So after implementing the correct functionality a switch was made from incoherent menus, each with different graphics, to a uniform look with transparent boxes. We came to the conclusion that giving up scenes that were not needed would be a good idea. The game over scene was replaced with transparent UI text and buttons, that would offer statistics and the possibility of either restarting or quitting the game. Our goal became being as close as possible to having only one main scene and so we managed to transform all the Menus and the Options scenes into transparent boxes and buttons that would appear in the main game scene.

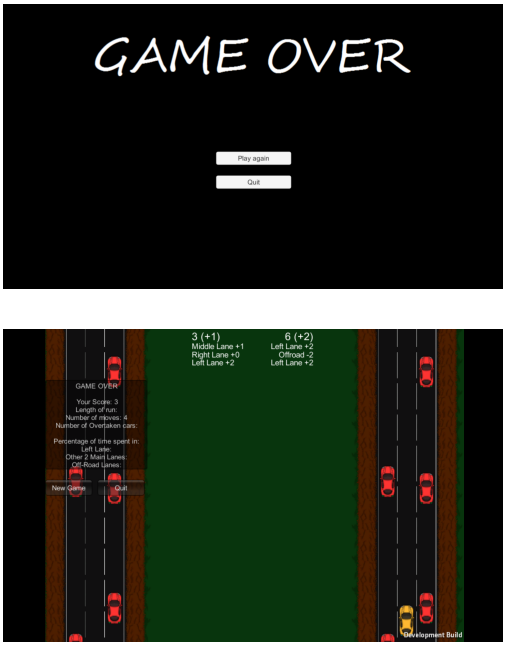


Figure 13: *top: the old game over screen, bottom: the new game over screen including the statistics*

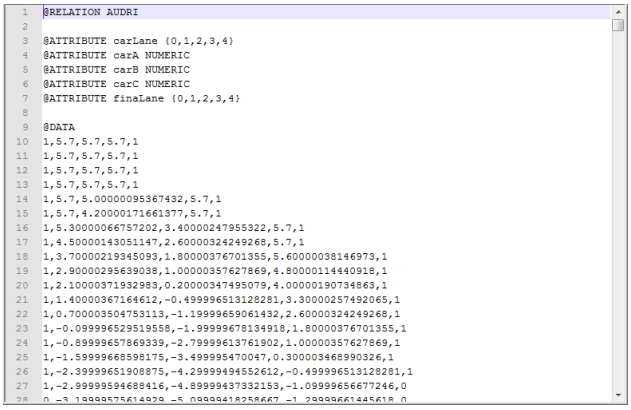
The Pause Menu suffered no changes throughout the creation process. However, we decided to change the initial Start Menu, which in the beginning only had two buttons ('New Game' and 'Quit') and added an 'Options' section, where the user can choose the difficulty of the game before starting it.

5.7 Simulator AI

A key part of the project shows how we can use machine learning tools to model user behaviour.The first step in achieving this was to introduce a logging system to record a users behaviour. This would then be used as a dataset to build the classifier. To implement a logging system we needed to decide on a set of attributes to record. Initially we focused on the attribute set outlined in the design section.

After playing back some of the models created by this dataset we found that WEKA struggled to classify the relative decision of left, right or stay.Therefore an abstraction of the relative decision making progress was made. We modified the class attribute to feedback a absolute decision, instead of a response to change lane if necessary the class attribute now represents a preferred lane to be in {0,1,2,3,4}. A full explanation of the changes made can be found in the testing documentation.
Finally after further testing we developed a new set that modelled the users behaviour successfully:

- the current lane of the car {0,1,2,3,4}
- the Y-coordinates of the lowest car in each lane
- the class attribute: the final lane of the user {0,1,2,3,4}



```
@RELATION AUDRI

@ATTRIBUTE carLane {0,1,2,3,4}
@ATTRIBUTE carA NUMERIC
@ATTRIBUTE carB NUMERIC
@ATTRIBUTE carC NUMERIC
@ATTRIBUTE finaLane {0,1,2,3,4}

@DATA
1,5.7,5.7,5.7,1
1,5.7,5.7,5.7,1
1,5.7,5.7,5.7,1
1,5.7,5.7,5.7,1
1,5.7,5.00000095367432,5.7,1
1,5.7,4.20000171661377,5.7,1
1,5.300000667572021,3.40000247955322,5.7,1
1,4.50000143051147,2.60000324249268,5.7,1
1,3.700002193450928,1.80000376701355,5.60000038146973,1
1,2.90000295639038,1.00000357627869,4.80000114440918,1
1,2.10000371932983,0.20000347495079,4.00000190734863,1
1,1.40000367164612,-0.4999965131282812,3.30000257492065,1
1,0.7000035047531128,-1.19999659061432,2.60000324249268,1
1,-0.0999965295791626,-1.999996781349182,1.80000376701355,1
1,-0.8999965786933899,-2.79999613761902,1.00000357627869,1
1,-1.59999668598175,-3.499995470047,0.3000034689903259,1
1,-2.39999651908875,-4.29999494552612,-0.4999965131282812,1
1,-2.99999594688416,-4.899994373321533,-1.09999656677246,0
0,-3.19999575614929,-5.09999418258667,-1.29999661445618,0
```

Figure 14: *The Output of the Logging System*

However we also wanted to train a model that would establish the ideal driving practices. These practices include maintaining a priority for the left lane while avoiding all the other cars on the road and using the off-road lanes as little as possible. This could be used as an evaluation tool by the user to judge their own performance. Using the basis of apprenticeship learning this meant we needed an expert who could demonstrate this behaviour. In addition to this we wanted this model to be virtually unbeatable or at least better than most of the users playing the game.

Our first efforts to implement this included recording one human playing the game trying to abide by the practices outlined as best they could. Reinforcement learning suggests that the model can't be better than the expert who trained it. So we tried to refine this idea further by combining the data from multiple users playing the game. Another suggested improvement involved selectively removing data from runs where the user scored particularly low. Neither of these ideas worked and we were stuck with two major issues with the AI:

1. At the lower difficulty settings, the user only needs to switch between the left and middle lanes to avoid most collisions. At higher difficulties when use of all lanes is required to avoid collisions, a human player often fails before collecting enough reliable data.

2. Every time a human user crashes, they have to restart the game and replay the lower difficulty portion again. Because of this 90% of the data focused on the left lane creating a very unbalanced data set.

```
private void bespokeAI() {
    float[] positions = getPositions ();
    float lowestLeftLaneEnemy = positions [0];
    float lowestMiddleLaneEnemy = positions [1];
    float lowestRightLaneEnemy = positions [2];
    float currentLane = positions [3];

    // Lane prioritisation.
    if (currentLane == -2) {
        // Check if lane -1 is free (want to move right, into left lane).
        if (lowestLeftLaneEnemy > dangerZone) {
            // Able to move right.
            simulatorObject.NewMove ("right");
        }
    } else if (currentLane == -1) {
        // Don't need to prioritise, in the best lane position (don't want to move).
    } else if (currentLane == 0) {
        // Check if lane -1 is free (want to move left, into left lane).
        if (lowestLeftLaneEnemy > dangerZone) {
            // Able to move left.
            simulatorObject.NewMove ("left");
```

Figure 15: *part of the bespoke AI*

Eventually we decided on a change of tactic and tried to implement a very simple AI using if statements that would capture the behaviour we wanted.The code was drawn from the driving practices we wanted to implement. This rudimentary AI was then run in place of a human user to build a dataset. In addition to this to help balance the dataset we fixed the initial difficulty setting to a similar setting that most human users would struggle with. This managed to solve both of our initial problems and now we have an expert that can demonstrate the behaviour we wanted to imitate.To further improve on this we combined this with our earlier policy of selecting runs with points above a certain threshold.

5.8 Evaluation Tools

We wanted the user to be able to evaluate their performance against AUDRI. Our initial plan was to allow the user to see both scenes, just not at the same time. They would firstly have to play the game themselves and only after that they would be able to compare their race with AUDRI, learning where they could have made better decisions in order to gain more points and avoid more enemies. In time, after implementing our idea and testing it, we found out that our approach had a couple of flaws. First of all, the user would only have to compare the two races from their memory, which would make it harder to understand their mistakes. Also, the simulator would not always be put in the same situations as the player at the same time due to the random arrangement of cars. Which would make it harder to associate the two. In this case, our project would practically fail to achieve it's goal, so creating the AUDRI driver would be slightly irrelevant.

At this point, we decided the way to solve this problem was showing both the game and the automated driver's races in parallel. We then merged the two scenes together and placed two roads on the screen at the same time. One of them, the one shown on the left side of the screen, would be the player's one, while the one on the right, would be AUDRI. We considered that allowing the user to play their own way and giving them the possibility of seeing the alternative provided by the trained driver at the same time would make comparing the results easier. Therefore, it would help the user see the better solution and so make the better choice in less time in the future. We also decided that both roads should look the same, which would be another way of facilitating the process of comparison for the person playing.

Additionally we wanted to include a set of statistics on the game over screen. These statistics allow the user to break down their performance in more detail. The data we decided to present the user include: score, length of run, overtakes, number of lane changes and the percentage of time spent in each lane. A similar set are available for the simulator. Hopefully this provides a more quantifiable solution for the user to evaluate their success.

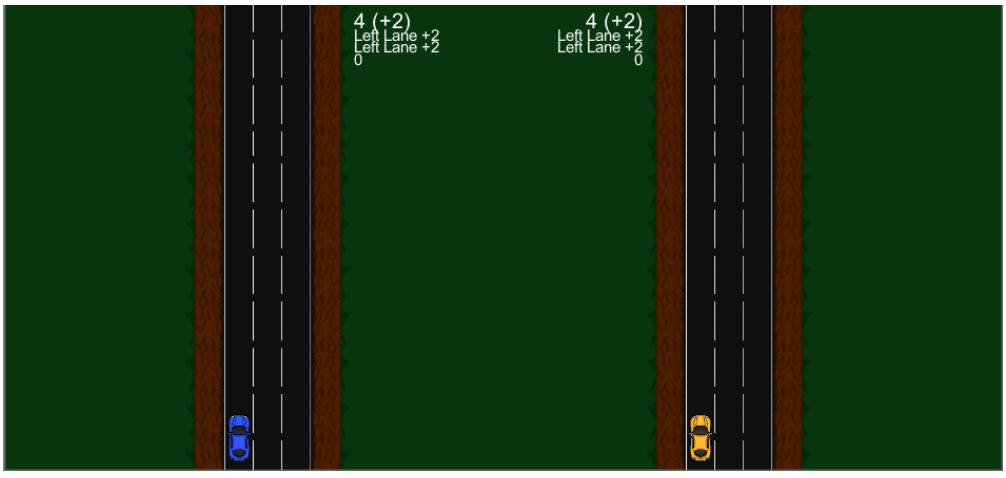


Figure 16: *Depiction of the new dual lane set up*

6 Problems Encountered

Our project specification does not detail any hardware requirements. Therefore most of the problems we have face come in the form of software problems. The two main areas of conflict have been using the lab computers in conjunction with Unity and learning the SVN.

6.1 Lab Computers

Our plan was to spend most of the time programming the game together in the labs to allow everyone to communicate effectively. However, when we tried to access Unity on the lab computers the program crashed shortly after opening. This problem limited us to personal devices during the early stages of development. Eventually we found the cause of the issue, an access problem when a Project was created. The issue was solved by using a different directory as the file source. This was time consuming because we needed the lab PC's to continue development and hours that should have been spent programming were instead spent debugging this hardware problem.

During the development phase a reoccurring problem materialised when certain functionalities were added to the system causing it to crash. We believe that the problem arises when interacting between multiple game scenes in Unity. This issue only occurs when using Unity on the lab computers. Hence we assumed the problem is caused by the older versions of Unity (Lab PCs). This caused problems for people who were using lab equipment for development. This also brought up an issue for Open Day as we will need to make alternative plans to avoid using the lab computers. This however was considered a minor setback as when compounded with the other issues caused by using the lab computers we decided to use our own equipment. We have added a help file into our project to help avoid these issues.

Another ongoing issue caused at times the scripts written in Unity to be inaccessible and we would have to re-import all over again. This issue became somewhat of a setback due to the repetition of reimports as the problem was not solvable since we do not have administrative access.

6.2 Subversion

During the prototyping phase, an issue uploading a file occurred. As we coded the simulator a group member uploaded a recently updated version of the simulator. However, when a second member of the team tried to upload their version, the update failed and the second member lost their progress on the project. This issue might be due to an inexperienced user on SVN.
Most of the group only had experience programming in C and Java. The new language provided a slight delay as the programming team had to teach themselves the language using online sources. Work was a bit more difficult in a sense that we are coding a somewhat complicated simulator with very little knowledge on C#. Unity itself was quite a daunting task as it also contains its own set of libraries, environments and API.

7 Evaluation

7.1 Technical Perspective

To evaluate the success of our project from a technical standpoint, we wanted to individually evaluate the three main components of our project. A set of black box tests designed to document the functionality of the simulation. T-tests to determine the effectiveness of the machine learning algorithms and usability testing to collect qualitative data about the user interface.

7.1.1 Test Plan

To perform the black box testing and test the functionality of the game against the initial requirements we created a testing plan. We referred back to the requirement specification to specify a number of test cases. At regular intervals the testing team would produce new versions of the testing plan updating the outcome of each test case. This plan was used during meetings to measure our progress towards the goal as well as being a useful evaluation tool. A full version of the testing plan is available in the appendix.

33. The car should only drive within the lane and off road.	Pass.	Car is limited to movement within the 5 lanes, e.g. pressing left in the left-most lane does nothing.
34. The Scoring System should look more like the prototype on the website. i.e. Users should be able to see +2 for being in the left lane. Ignore the score values used in that image	Fail.	Scoreboard covers the basic functionality, but lacks the information to the user (+2 left lane, -1 offroad)
35. The users car should spawn in the left lane, to promote the safe driving behaviour	Pass.	Car now correctly spawns in the left lane.

Figure 17: *A example of the testing documentation*

The testing plan shows that there a number of requirements that we didn't implement. One such test suggests that the system should allow the user to "review their run with the automated driver in charge". This requirement was designed as a potential way of helping the user to check their driving performance. During implementation it was decided that this idea would be unnecessarily complicated to implement. This is when we brainstormed the idea of the dual road system. Now that the dual roads allows the user and the automated driver to both run through the same layout of cars the requirement was discarded. Therefore we replaced that requirement with this:

1. The game window should be split to represent two different roads.
    1.1. The car on the left road will represent the user.
    1.2. The car on the right road will be controlled by the simulator.
    1.3. The cars should be differentiated for ease of use.
    1.4. The layout of enemy cars should be identical for both roads.

Another test that failed was designed to fulfil the requirement: "The user should be able to create and playback a model of their behaviour". This requirement would allow the user to play the game, generate a dataset and view the model created from their dataset. This functionality hasn't been implemented mainly because of time restraints. This functionality

doesn't represent a core mechanic and this is why this feature was pushed back compared to others. However we still hope to have this implemented in time for the presentation and open day.

7.1.2 T-Tests

A t-test can be used to determine if two sets of data are significantly different. It is a hypothesis test that checks if the mean of two groups of data are reliability different based on a p-value. Here it is relevant to use a paired t-test[9] because a sample from one observation can be linked to an observation form the second sample.

Firstly we decided to test the capabilities of the system to build new models that can copy a users behaviour. We wanted to train a model different to AUDRI so the characteristics chosen included maintaining a right lane priority and never using the outside lane on the right. The next step was to introduce a null hypothesis[5].

$H_0 : \mu_0 = \mu_1$

where
μ_0 = the mean of the users score
μ_1 = the mean of the simulators score

We will aim to try and disprove this hypothesis through the t-tests. However if we can't disprove it then we have to accept that there is no significant difference between the samples. If the outcome of the t-test finds a value greater than the p-value then we cannot conclude that there is a significant difference. For this experiment we decided to use the generally accepted p-value of 0.05

To collect the data the system was set to a low difficulty so the player wouldn't have trouble crashing. This was to hopefully limit the number of variables introduced in the test. Data was collected for approximately 2 minutes and 30 seconds. AUDRI (on the right road) was replaced by this new model. This time the user played the game, using the same behaviour, while the model was also running. Running both samples of the same layout of cars was another attempt to reduce the variables. After a minute of running the points totals of both the player and the model are recorded and the game restarted. This was repeated 10 times but twice the user crashing during that minute. Therefore we will ignore these as outliers and compare the remaining 8. We decided to use two tailed tests because we couldn't correctly predict whether the simulator or user would score more points. The points aren't reflective of the behaviour we are enforcing but instead offer a comparison of the lanes used.

The test shows that we cannot prove a significant difference between the samples. The result tells us that there is a 15% chance that the samples come from the same group. This is an adequate result for 2 and a half minutes of training. However we wanted to take the model further to see if we could see an improvement from the AI. So the initial dataset was doubled by letting the user play for an additional 2.5 minutes. The outcome shows a massive improvement again with a 43% chance of the data belonging to the same set.

	Sim	User			
	35	29			
	24	29			
	35	32		t-test =	0.146488743
	41	41			
	31	27			
	44	42			
	24	23			
	46	41			
After 2:30 more of adding to the dataset					
	User	Sim			
	28	27			
	20	18			
	36	35			
	24	26		t-test =	0.4304161722
	27	30			
	30	31			
	42	40			
	35	27			

Figure 18: *The data collected and the results of the t-tests*

In our introduction we indicated that if the machine learning tools could generate behaviour close to that of the expert who created the dataset, it would be evaluated a success. From these experiments we have been unable to reject our null hypothesis. Therefore, it is reasonable to state that there is no significant different between the two. Now we can determine that we successfully completed the objective set out in the introduction.

7.1.3 User Testing

Its important to test the intuitiveness of the game-play and the accessibility of the menus before releasing the program to the public on the open day. To test the user interface we gathered some qualitative feedback from people unfamiliar with the game. The testing mainly involved watching outsiders to our project interact with the program. We found it hard to find random people who wanted to spend time testing software for us. So along with a few other groups we all spent time playing with each others projects. One reoccurring criticism brought up included the speed of the enemy cars. Having fixed speeds for all the enemy cars takes some of the challenge and realism away from our project. This personified internal fears we had during development. We tried to identify ways we could increase the variety of the simulator to better reflect real world decisions. Eventually, we adopted the idea of a slow lane that we discovered from a similar research project. This would allow us to change the speed of the cars based on what lane they spawned in but avoid other cars colliding with each other.

7.2 Management Perspective

7.2.1 Time Plan

When we started the project we placed an importance on time management as a way of maximising our development time while keeping work load manageable for everyone with different commitments. We decided the best way to structure our time was through the use

of a PERT chart. A PERT chart allows us to measure the critical path of the project. This is important for meeting deadlines and allowing us to compensate for potential problems we encounter during development.We created an initial PERT Chart (Appendix A) under the principle that we would have a working implementation of the game mechanics before the interim report was due. We believed this was a good stopping point to write the report before we started implementing the machine learning aspect of the project. We anticipated that the game would be fairly easy to implement as we were using an engine with an intuitive API and a focus on game development. The hard part of the project would be integrating WEKA with the simulator.

Unfortunately we quickly fell behind schedule during the implementation of the game mechanics. This can be attributed to some of the problems detailed in the previous section and also to an inexperienced team trying to be too ambitious to get the project working. Therefore we pushed the back the completion of the game mechanics until mid January. The extension of this deadline meant that the testing of the game mechanics was fairly limited before we started integrating the AI.

The lack of testing did create issues later on, specifically when changing the coordinates of the cars to allow the user and simulator to play simultaneously. All previous data collected in the game had to be abandoned because the coordinates of the cars were an attribute collected for WEKA. Therefore we had to spend extra time retraining WEKA when any substantial changes to the mechanics were made. Luckily the overall effect of the changes to the time plan didn't impact progress any further and we managed to have a working AI by the deadline.

7.2.2 Methodology and Communication

The team needed a medium to be able to communicate efficiently as each of us had different responsibility that occupied most of our day. To better communicate and to coordinate meetings we decided upon whatsApp which is a social media application that allows a group chat function that seems invaluable to transfer pictures, messages and files of sorts to proceed with the project.For an alternative to SVN google drive allowed us to share resources of file natured. This is due to the SVN mainly used for code as well as the website being focused on specific content. Google Drive had tools such as Word and Spreadsheets that was portable by just using a browser that was very convenient.

Although the extensive size of the project. the programming component seemed to work better when in components and segmented into parts (Game, AI, UI). This provided the group to team up in Extreme Programming which proceeded quite well on the fact that each person had strengths and weaknesses each person can help out. The prototyping was broken into 3 phases. First one being the paper prototype that was designed using the set specifications and when that prototype satisfied said specification it was rapidly converted to a software prototype. This prototype did not take long and was successful and provided enough evidence of the constraint and functionalities of the would be software. This then allowed a blueprint to design the simulation prototype that imitates a would be AI and

machine learning component of the software.

7.3 Reflective Comments

From a technical perspective we have shown the relative success of our project. We believe this achievement is down to a number of key decisions. One choice involved our switch from Java to Unity. This was a hard decision because we took a risk, not knowing if this would overcomplicate things further in development. Another positive note was the value we put into our meetings. These meetings were highly attended throughout the year by everyone. A full list of minutes from these meetings are available in the appendix This allowed everyone to stay in contact and connected to the project. Finally the realisation to adjust unattainable requirements to more realistic goals showed a maturity in the decision making process. This allowed us to ensure all core implementation features were successfully completed on time.

However we should also look at our shortcomings. One of the weakest parts of our project is the user interface. We can attribute this to a lack of design work put towards the UI. We reasoned this by suggesting it made sense to implement the game mechanics before the UI. This led to a lot of confusion about the user interface and eventually we had to scrap some of the features we were planning. One feature includes the ability for layers to build their own models. Most of the WEKA code for this feature was implemented and the lack of a competent UI held us back. We should have placed a higher focus on the UI during the design meetings. Another mistake we made during the design was limiting the simulator to one collision. In reflection the ability for multiple collisions opens up more possibilities when training a model. It would eliminate some issues with unbalanced datasets. If we were to repeat this project, we would add collision detection to the scoring system but allow the player/simulator to continue after a collision is detected. Although we were happy with our progress during the informal meetings. We believe that the formal meetings should have been utilised better. This could be improved upon by discussing, during our informal meetings, exactly what we needed guidance on before meeting with our supervisor.

8 Bibliography

References

[1] Madrigal A. The trick that makes google's self-driving cars work the atlantic. Online, May 2014. Viewed 13 Nov 2014.

[2] P Abbeel. Apprenticeship learning via inverse reinforcement learning. Journal, 2004.

[3] Sergio A. Alvarez. Decision tree pruning based on confidence intervals (as in c4.5). Online, Sept 2008.

[4] Girja Sharma Dr. Neeraj Bhargava. Decision tree analysis on j48 algorithm for data mining. Journal, Jun 2013.

[5] Dr. David Stone/Jon Ellis. Evaluation of means for small samples - the t-test. Online, August 2006.

[6] Abernethy G. Piatetsky. 15th annual analytics, data mining, data science software poll: Rapidminer continues to lead. Online, 2014. Viewed 14 March 2015.

[7] Giorgio Ingargiola. Building classification models: Id3 and c4.5. Online, Oct 1997.

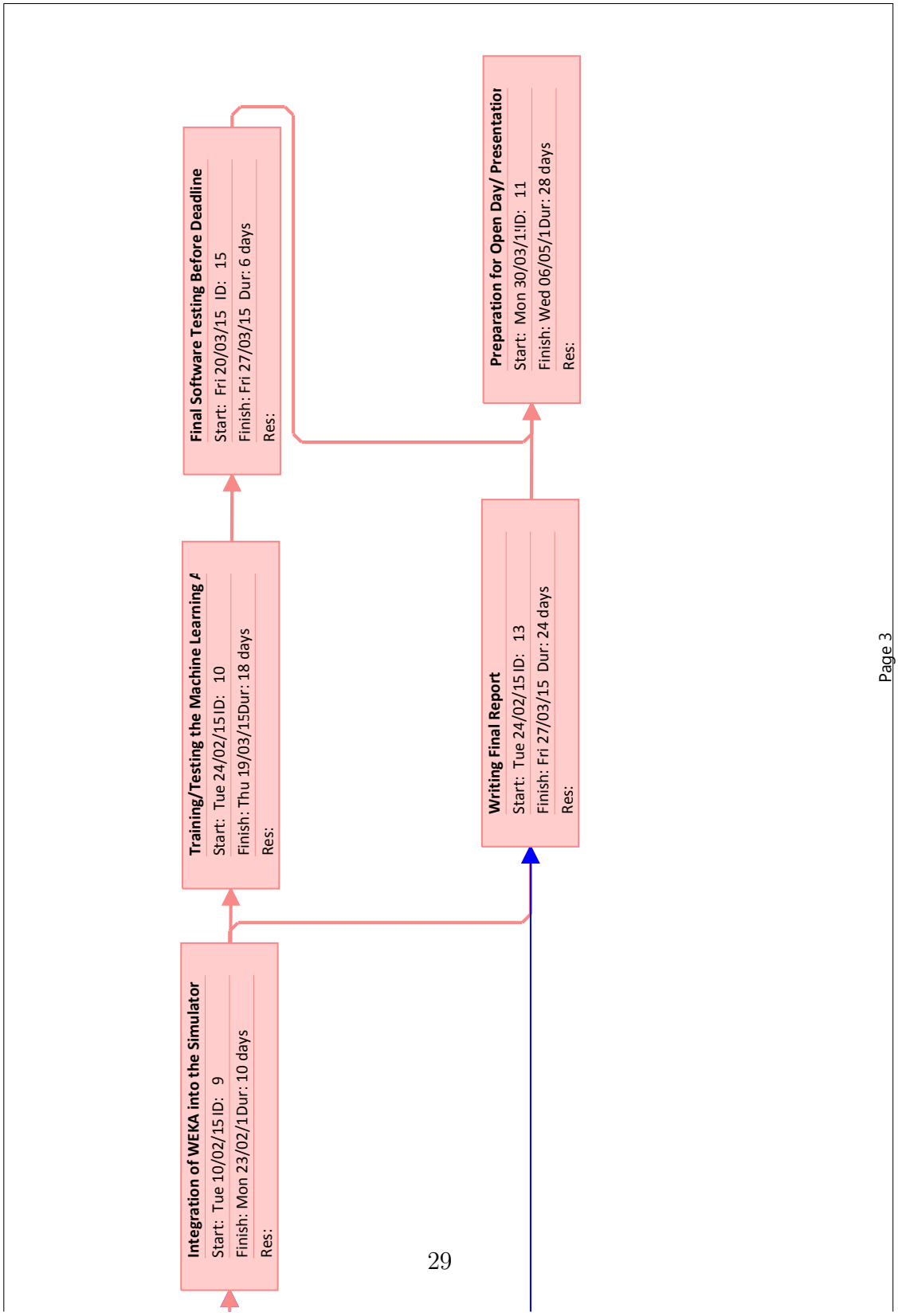
[8] The University of Waikato. Ikvm with weka tutorial. Online, Sep 2012.

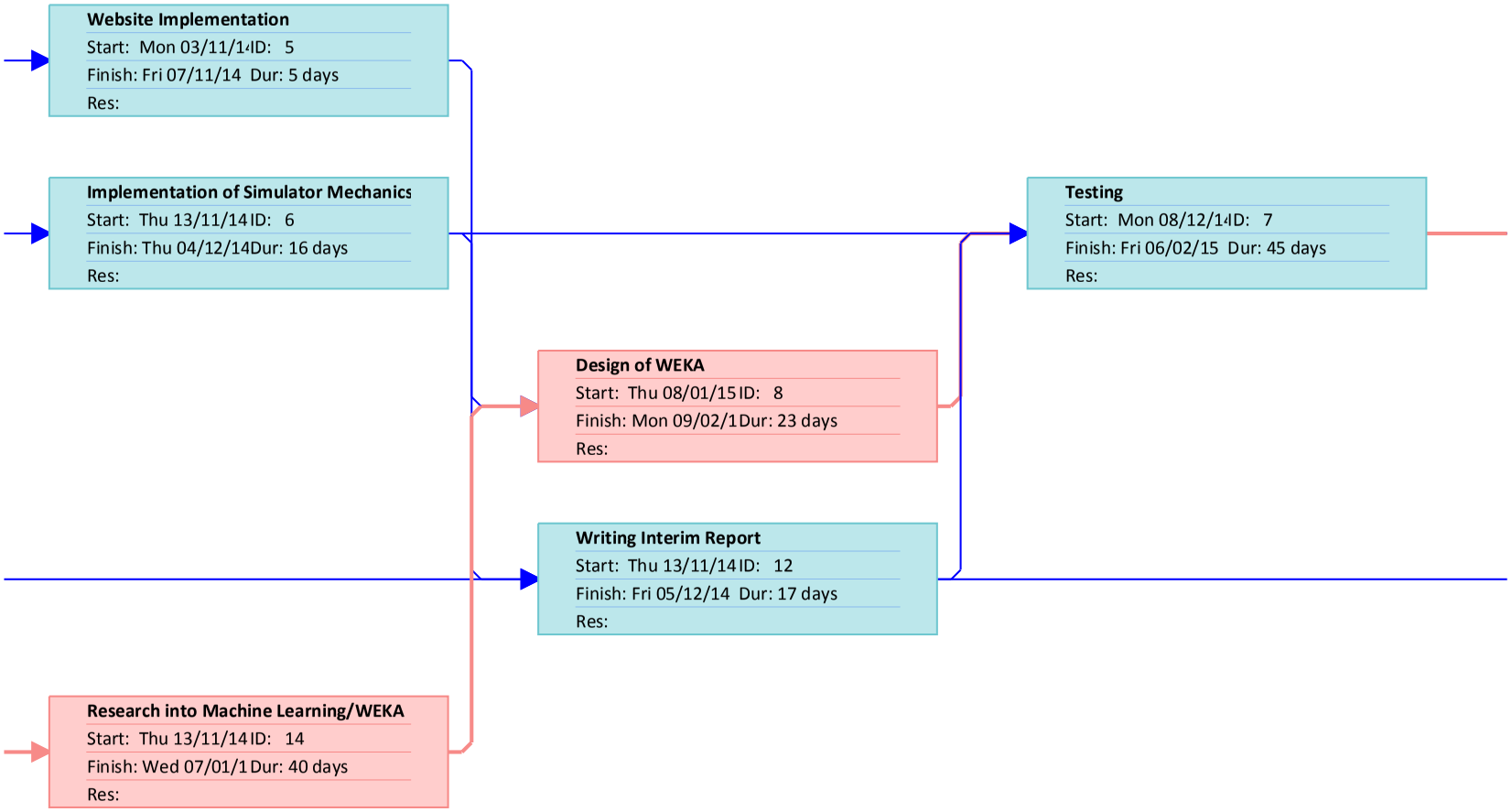
[9] Rosie Shier. Statistics: 1.1 paired t-tests. online, 2004.

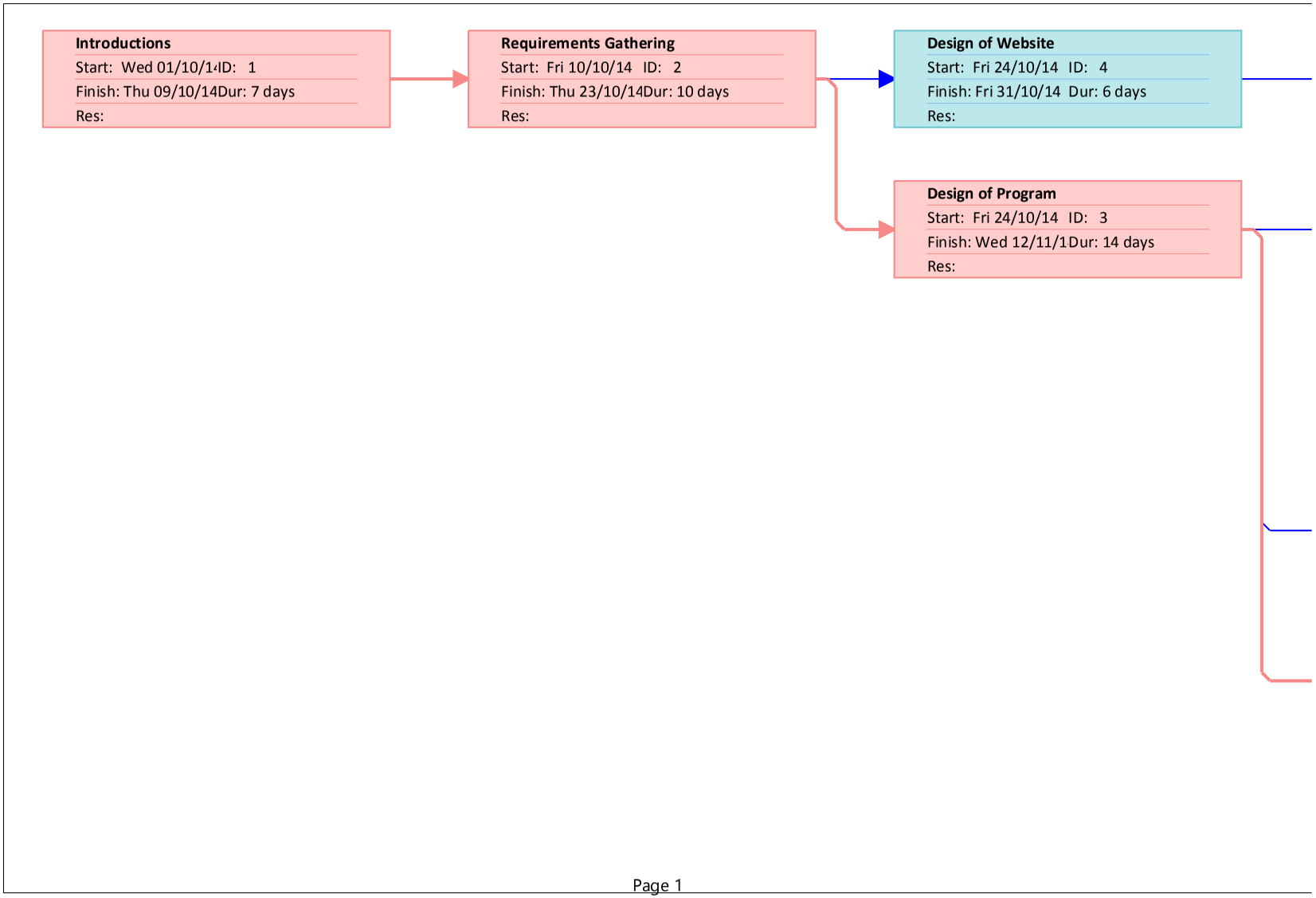
[10] Alan Stagner. Good coding practices in unity. Online, Oct 2014.

9 Appendix

9.1 Appendix A - PERT Chart




Website Implementation
Start: Mon 03/11/14 ID: 5
Finish: Fri 07/11/14 Dur: 5 days
Res:
Implementation of Simulator Mechanics
Start: Thu 13/11/14 ID: 6
Finish: Thu 04/12/14 Dur: 16 days
Res:
Testing
Start: Mon 08/12/14 ID: 7
Finish: Fri 06/02/15 Dur: 45 days
Res:
Design of WEKA
Start: Thu 08/01/15 ID: 8
Finish: Mon 09/02/1 Dur: 23 days
Res:
Writing Interim Report
Start: Thu 13/11/14 ID: 12
Finish: Fri 05/12/14 Dur: 17 days
Res:
Research into Machine Learning/WEKA
Start: Thu 13/11/14 ID: 14
Finish: Wed 07/01/1 Dur: 40 days
Res:


Introductions
Start: Wed 01/10/14 ID: 1
Finish: Thu 09/10/14 Dur: 7 days
Res:
Requirements Gathering
Start: Fri 10/10/14 ID: 2
Finish: Thu 23/10/14 Dur: 10 days
Res:
Design of Website
Start: Fri 24/10/14 ID: 4
Finish: Fri 31/10/14 Dur: 6 days
Res:
Design of Program
Start: Fri 24/10/14 ID: 3
Finish: Wed 12/11/1 Dur: 14 days
Res:

9.2 Appendix B - Use Cases

Goal: Self Learning System
Primary Actors: WEKA, Programming Team
ID: 002
Description:

1. A user plays the game.
2. The environment of the game is recorded periodically.
3. The data is split into training and test data and put through the chosen learning algorithm.
4. WEKA starts by copying a single users good behaviour. Eventually WEKA will be able to recognise the difference between a bad state, that will lead to a crash, and a good state that is a safe driving decision.
5. Once the system has received input from multiple users over a good amount of time it should be able to sufficiency classify new possibilities into good and bad states and chose for itself which option should be taken (stay in same lane, move left, move right etc).

Pre-Conditions: 001
Scope: Machine Learning
Level: Summary

Goal: Exiting The Simulator Early
Primary Actor: User
ID: 005
Description:

1. The user starts the simulator.
2. For some reason the user doesn't want to continue playing the driving simulator.
3. The user pauses the game and can quit the game from a menu.
4. Any useable data is saved by the system.
5. The system quits the current run of the simulation and returns to the starting arrangement.

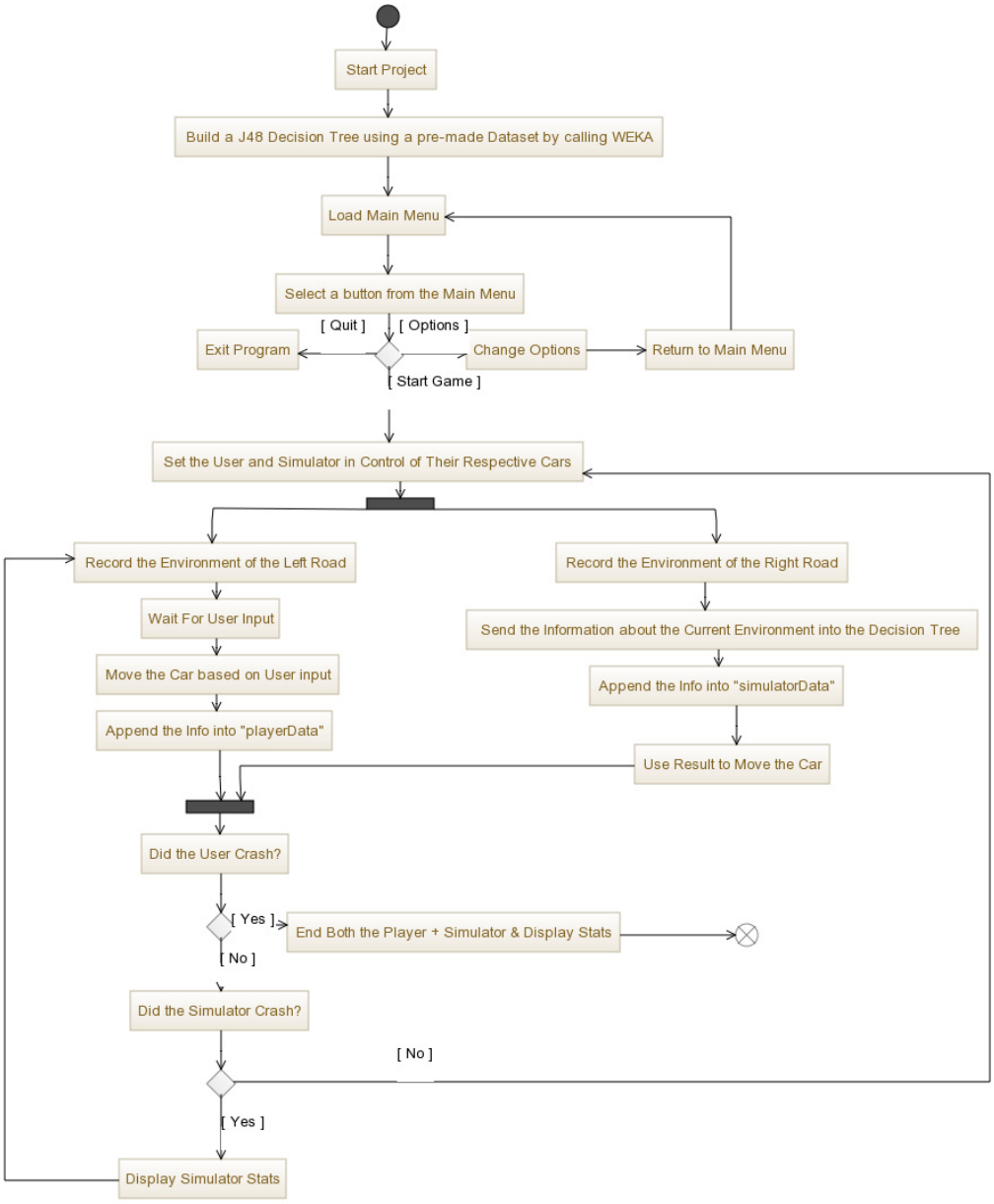
Pre-Conditions: 004
Scope: User Interface
Level: User

Goal: Successfully Finishing The Game
Primary Actor: User
ID: 006
Description:

1. The user launches the application and confirms the start of the game.
2. The games screen appears and the cars start moving.
3. The user plays along and successfully manages to avoid all the enemy cars, until the time is over.
4. A winning screen appears, with the appropriate statistics - points gained and time spent playing without crashing any enemy car - for the user.
5. The user is given the chance to review their progress during the game, replay the game or exit the application.

Pre-Conditions: 001
Scope: User Interface
Level: Sub-function

9.3 Appendix C - Updated UML Activity Diagram



9.4 Appendix D - Minutes

Minutes for the meeting held on the 27th October 2014

```
General Information:
Meeting Minutes.
Meeting 6.
Formal Group Meeting 3.

Date and Time:
27 October 2014.
12:00 - 12:40

Attendance:
Lee Haines
Samuel Armstrong
Syed Khaidzir
Madalina Stoica
Qi Xing
Ophelia Tsang
[ Not in attendance: Jianan Dai (ill) ]

Agenda for Meeting:
- Discuss WEKA tutorials.
- Discuss Unity engine.
- Discuss future plans.

Progress of Meeting:
- No more tutorials planned for WEKA by the PHD student.
- Need to email the requirements to Ender and Asta by Friday.
    - Need to figure out parameters like other cars speed and number of cars present.
- Need to contact the project coordinator about the use of a different language than the spec (for the Unity engine).
- Need to set a date to begin writing the initial sudo code.
- Begin the project website (still need decisions to be made so there is information to write).
- Informal after meeting:
    - Sam; do project brief for website by Friday,
    - Afiq; work out if WEKA can be used with C#,
    - Mada; look over the requirements,
    - Lee; gantt chart,
    - Set a date for the sudo-code session: 3pm-6pm Thursday,
```

Minutes for the meeting held on the 26th January 2015

```
General Information:
Meeting Minutes.
Meeting 17.
Formal Group Meeting 8.

Date and Time:
26th January 2015.
12:00 - 12:30

Attendance:
Lee Haines
Samuel Armstrong
Syed Khaidzir
Madalina Stoica
Qi Xing
Ophelia Tsang
Jianan Dai

Agenda for Meeting:
- Discussion about what state the project is currently in.
- Long term future plans of what is to be achieved.
- What specifically are the next objectives.

Progress of Meeting:
- All the basic game implementation has been added.
  - Removal of sample files to make the project easier to view.
  - Changed the point system to be handled by the player and in c#.
  - Added spawning mechanics for enemy objects.
  - Collisions work between the player and the enemy objects.
- Long term plan is to split into two teams; advanced game mechanics and machine learning.
- The next objectives include UI, difficulty and bug fixes.
```

A full list of minutes are available in the SVN repository at: https://code.cs.nott.ac.uk/p/gp14-sp-exo/doc/

9.5 Appendix E - Testing Plan

Testing Issues	**Result**	**Method**
1. The view of the road should be top down.	Pass	Observing the game surface.
2. The users car should stay near the bottom of the screen allowing enough time for the user to see what is coming.	Pass.	The car will go outside of the surface when crashing the enemy cars or automatically go outside. When the Collides happened, the game crushed. The collides happen, the game over interface is showed.
3. The users car should firstly appear at the bottom of the screen, on the middle line of the road.	Pass.	Through the code, the user car position is (0,0), which is the bottom of the screen, on the middle line.
4. The left lane should always be priority for the users car.	Pass.	The score will be added by 2 when user car is on the left road.
5. The GUI should display three lanes and sufficient space on either side for the car to go off-road.	Pass.	Observing the game surface.
6. The car should be able to go off-road, turning the three lanes into 5 different options.	Pass.	Starting the game and controlling the user car, the car can be drove on three roads and two sides off-road.
7. The two extra lanes representing the off-road route should have the colour green, embodying grass.	Pass.	The graphics has been added.
8. The user gains 2 points with every car they overcame while driving on the left lane of the road.	Pass.	Watch through the code and observing the score change when the user car on the left road.
9. The user gains 1 point with every car they overcame while driving either on the middle or the right lane of the road.	Pass.	Watch through the code and observing the score change when the user car on the middle and right roads.
10. The user loses 1 point with every car they overcame while driving on one of the off-road lanes.	Pass.	Watch through the code and observing the score change when the user car on both off-roads.
11. All enemy cars should have the same colour, this colour should be easy to distinguish from the user's car.	Pass.	Observing the game surface.
12. The same exact dimensions and shape as the user's car.	Pass.	Observing the game surface.

13. enemy cars should randomly appear only on top of the screen.	Pass.	Observing the game process.
14. When they appear they should fall down in the direction of the users car, simulating the movement of the cars on the screen, without changing the lane they initially appear on.	Pass.	Observing the game process.
15. Other cars should never crash into each other.	Pass.	Observing the game process.
16. When two cars get within a proximity the system should register a crash.	Pass.	The system now reports a crash. The "Game Over" function has been added.
18. The screen should show the length of the run.	Pass	The length of the run is included in the statistics on the game over screen
19. The screen should show the amount of points the user has gained while playing the game.	Pass.	Observing the game process.
20. The system should give the option of reviewing the run with the automated driver in charge.	Fail.	Uncovered function.
21. The user should be given the option of playing the game again.	Pass.	The user can use this function only when they pause the game but not finished the game. The function is added.
22. The program should be efficent enough to run smoothly and success-fully on the lab computers.	Pass.	Running the game on the lab computers.
23. The program should have a simple and clear GUI that should be able to be interpreted by all adults.	Pass.	Observing the game surface.
24. The program should give inexperienced users a clear specification about how to start and control the simulator.	Pass.	The instruction has been added.
25. The program should be stable as possible and if the program crashes it should do so gracefully while saving as much data as possible.	Pass.	The program is written in C# in Unity, the code is stored in the Unity.
26. The difficulty of driving should be adjusted to a suitable situation in order to obtain useful data.	Pass	The difficulty of the simulation can be adjusted from the main menu.
27. The program should respond to input from the keyboard/mouse.	Pass.	The starting and finishing is input from mouse, the user car controlling is input from keyboard.
29. The user should be able to quit the game early or pause the game.	Pass.	Observing the game process.
30. The options can be manipulated using the keyboard or mouse.	Pass.	

		The options has been added but only allow the users to choose by using mouse.
32. The number of enemy cars on the road should increase.	Pass.	The function is added.
33. The car should only drive within the lane and off road.	Pass.	The car will go to the position, which the program didn't record. The problem is fixed by reporting game over when two cars crash.
35.The Scoring System should look more like the prototype on the website. i.e. Users should be able to see +2 for being in the left lane. Ignore the score values used in the image.	Pass.	Scoreboard covers the basic functionality, but lacks the information to the user (+2 left lane, -1 offroad) The scoring system is added and looks like the prototype on the website.
34.The program should save the environment at regular intervals.	Pass.	The positions of enemy cars and main character car are exported.
36.This data should be saved in a format that can be easily run through WEKA.	Pass.	The document "wekaexport" is in the format of docx. The document "TrainingSet" has been added with the format of txt.
37.The simulator's behaviour should follow the rules of the main character car's rules.	Pass.	The simulator won't crash with enemy cars. The simulator runs in the positions which the program is recorded. The simulator share the same score calculation methods.
38.The simulator will perform, at least, almost as well as the player the model is based off	Pass.	From the t-tests it was determined the simulator performs correctly.
39. The options menu should allow the user to choose from a directory of models to take the place of AUDRI	Fail.	The options only contains the difficulty option so the dataset used has to changed directly from the code

9.6 Appendix F - Testing Documentation For AI

Testing Documentation for WEKA
Documentation showing the progression of the AI in the attempt to get a perfect driver.

Date	Attribute Set	DataSet Size	Recording Data Interval (s)	AI Refresh Rate (s)	Details	Results
18/02/15	CurrentLane, Y-Coordinates of lowest cars from each lane, **class attribute = decision of movement (left,right,stay)**	~1500 (new dataset)	0.1	0.1		Alot of flickering between lanes very quickly, moving very quickly back and forth, fails to mimic the users movement.
18/02/15	^^^	~3000	0.1	0.5	Slowed the refresh rate of algorithm to reduce number of movements made	New behaviour: car stays in lane 1 until an enemy car appears in that lane. Then correctly makes decision to move. However it always chooses to move into the outside lane (left)and then repeatedly chooses left even though it can't move.
18/02/15	^^^	~1500 (new dataset)	0.1	0.5	Changed the Y-Coordinates of the attribute set. Now LaneB always represents the lane the user is in. LaneA is the lane to the left of the driver and LaneC the right. Reasoning for this is that it allows the AI to make a more relative decision based on the cars in front of it.	Same behaviour as above. Car moves into left outside lane and never moves out of it.

19/02/15	Y-Coordinates of lowest cars from each lane, **class attribute = current lane the car should be in {1,2,3} excluding the two outside lanes**	~1500 (new dataset)	0.1	0.5	We decided to simplify the decision the AI has to make. We did this by changing the class attribute to represent the lane the car should be in, instead of a relative movement decision. Also reverted the previous change with the Y-Coordinates of the lanes	Attempt at copying the users behaviour. However if the user recording the data follows a priority for the left lane. WEKA produces a tree with only one decision based on the Y-Coordinates of LaneA (the left lane) *Update 28/2/15: After later tests we realised this behaviour to due to WEKA pruning the tree by default due to the unbalanced dataset*
21/02/15	^^^	~4000	0.1	0.3	Increase in size of dataset with more random user behaviour to gather data for more lanes	Copies the users behaviour when switching between lane 1 and 2. When difficulty increases and the AI needs to make use of the other lanes, it crashes.
23/02/15	Initial lane of the users car. Y-Coordinates of lowest cars from each lane, **class attribute = final lane the car should be in {1,2,3} excluding the two outside lanes**	~2500 (new dataset)	0.1	0.3	Added a new attribute: initial lane. Now the AI can use the users current lane when determining a new lane.	Slightly better behaviour. Still trouble with lack of data regarding movements into the third lane.
05/03/15	^^^	~4000	0.1	--	We decided on a change of tactic. Instead of using a player to extract a dataset, we created an example behaviour using if statements, like a makeshift AI.	Successfully follows the intended behaviour.

					Then we used this to record a dataset	
12/03/15	^^^	~4000	0.5 if no movement	--	Data is improved using selection based on points total of the expert. This eliminates some the error introduced by bad car spawns or human error	Successfully mimics the ‘expert’ according to the t-tests. Further test prove using an interval of 0.5s for no lane change reduces the excessive amounts of logging. Movement still logs