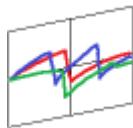


ALEXANDER MCCORMICK



# COMP 4

---

## StratSim - Formula 1 Strategy Calculator

**Alexander McCormick**  
**Poole Grammar School**

**Summer 2014**

**Candidate Number: 2286**  
**Centre Number: 55235**

The design, implementation and testing of a Formula 1™ strategy computer, for the purpose of minimising the total race time of the Red Bull Racing Formula 1 cars in the FIA World Championship.

# COMP 4

---

## Contents

<b>Contents .....</b>	<b>1</b>
<b>Analysis.....</b>	<b>5</b>
Background to and identification of problem .....	5
Problem Identification .....	5
Formula 1 .....	5
The Development of Strategies .....	5
Modern Strategy Computing.....	6
Brief .....	7
Analysis of current system .....	7
System Function .....	7
System Type .....	9
Sketches .....	10
Analysis .....	14
Detailed Brief.....	15
Further Research .....	15
Potential Improvements .....	17
Identification of the prospective user .....	17
Client.....	17
Red Bull Racing .....	17
Identification of user needs and acceptable limitations .....	18
Interview .....	18
Summary of Limitations Discussed .....	22
Initial user requirements.....	23
List of requirements .....	23
Targets .....	24
Agreed Limitations .....	25
Hardware and Software Restrictions.....	26
Schedules .....	26
Data dictionary (from perspective of end user).....	27
Data sources and destinations .....	28
Summary .....	28
PDF Files .....	29
Text File Format .....	29
User Inputs.....	29
Strategy graph output.....	30
Internal IO.....	30
Data flow diagrams.....	31
Summary .....	31
Key.....	31
Existing System .....	32
Proposed System.....	33
Volumetrics .....	36
Analysis .....	36
Data types .....	36
Volumetrics.....	36
Summary .....	38
Feasibility of potential solutions.....	38
Options.....	38
Justification of chosen solution .....	39
Complexity.....	39
Structure.....	39

Rarity .....	39
Use .....	39
Language.....	39
Solution Specification.....	39
<b>Prototype .....</b>	<b>40</b>
Design of Prototype .....	40
Limitations .....	43
End User Feedback .....	43
Further Research.....	45
Final user requirements.....	46
<b>Design.....</b>	<b>50</b>
Overall system design.....	50
Principles .....	50
Structure Chart.....	50
Class Inheritance.....	54
Public Structures and Enums .....	54
Classes .....	54
Inheritance Diagram.....	59
Description of modular structure of system.....	60
Views (forms) .....	60
Model (data).....	62
ViewModel (data processing) .....	62
Data dictionary .....	64
Properties .....	64
Description of record structure.....	67
Static Files.....	67
Dynamic Files .....	67
Databases .....	68
Validation required .....	69
Pace parameters .....	69
Strategy Optimisation .....	70
Driver parameters.....	70
Settings .....	71
Validation Routines .....	73
File organisation and processing .....	75
Initialisation Files.....	75
Data Input Files.....	76
Data Output Files .....	76
Identification of appropriate storage media .....	78
Program Distribution .....	78
System Storage .....	78
Desktop Use .....	78
Mobile Use.....	78
Identification of processes & suitable algorithms for data transformation.....	80
Reading from PDF .....	80
User interface design (HCI) rationale.....	90
Planning .....	90
Sample of planned data capture and entry designs .....	90
Input/Output Windows.....	93
Sample of planned valid output designs.....	96
Description of measures planned for security and integrity of data .....	97
Data Protection .....	97
User Access Rights .....	97
Data Integrity .....	97
Summary of Measures .....	97
Description of measures planned for system security.....	98
Overall test strategy .....	98
During Development.....	98

System Testing.....	100
Test Datasets.....	100
<b>Implementation.....</b>	<b>102</b>
Index of Complex Code Structures.....	102
Program listing .....	104
StratSim .....	104
StratSim.Model.Files .....	107
StratSim.Model .....	120
StratSim.View.Forms.....	165
StratSim.View.MyFlowLayout.....	167
StratSim.View.Panels .....	227
StratSim.View.UserControls.....	276
StratSim.ViewModel .....	291
Unit Testing.....	308
Contents .....	308
Specification Unit Testing .....	308
Strategy Unit Testing .....	314
Validaiton Unit Testing.....	315
Screen shots of working program .....	318
Introduction.....	318
Setting Up and Running a Race Simulation.....	318
Other Panels.....	329
Flow Layout.....	331
<b>Testing .....</b>	<b>335</b>
Test plan.....	335
1. Testing of Data Input .....	335
2. Specification Testing .....	348
3. Window 'Flow Layout' Testing.....	357
4. Unit Testing .....	360
Notes .....	367
1. User Input .....	367
2. Specification Testing.....	369
Hard copy evidence .....	370
1. User Input Testing.....	370
2. Specification Testing .....	385
3. Flow Layout Testing .....	389
<b>System Maintenance.....</b>	<b>392</b>
System Overview.....	392
System Details .....	392
System Specification.....	392
Detailed System Description.....	392
Additional System Features .....	393
Further Information.....	393
Detailed Algorithms.....	393
Annotated listings .....	393
Procedure/function and variable lists.....	394
Commented Routines.....	394
Class Summaries .....	395
Self-documentation .....	395
Important Routines.....	396
<b>User Manual.....</b>	<b>412</b>
Contents page .....	412
Introduction.....	413
Installation Instructions.....	414
Downloading and Installing the System .....	414
Local Files.....	414
Quick Start Guide .....	415

Start Menu.....	415
Pace Parameters .....	415
Strategy Optimisation .....	416
Race Simulation .....	416
Detailed description of how to use the system.....	417
Getting Started .....	417
Settings .....	418
Loading Timing Data .....	419
Strategies.....	423
Race Simulation .....	425
Archives .....	426
Graph .....	426
Driver Panel .....	429
Axes .....	430
Modifying Driver and Race Data.....	430
Samples of actual screen displays.....	431
During System Operation.....	431
File Formats .....	431
Troubleshooting.....	434
Crashes.....	434
Error Messages .....	434
<b>Appraisal .....</b>	<b>436</b>
Comparison of project performance against objectives.....	436
Specification Acceptance .....	436
Real World Testing .....	436
Possible extensions .....	440
Features.....	440
Accuracy .....	440
Code Refactoring .....	441
User feedback .....	442
Appraisal.....	442
Actions .....	442
Project Evaluation.....	444
<b>Appendices.....</b>	<b>445</b>
Appendix 1 .....	445
Calculus used for strategy optimisation .....	445
<b>Notice .....</b>	<b>448</b>

## Analysis

### Background to and identification of problem

#### Problem Identification

While on work experience at Red Bull Racing in Summer 2013, I was shown the computer system that the team uses in real-time during Formula One race weekends to predict the outcome of the races and try to influence them.

While the system that was used seemed technologically advanced, there were flaws in the system that could lend a competitive advantage to other teams. The head of Software Development at Red Bull Racing, Richard Dutton, asked if I could use some of my computing experience to iron out some of these flaws, and to produce a result in less time that was more accurate to the recorded timings; in other words so that the system was less subject to human intervention.

I believed that the current system could be improved upon using a dedicated piece of simulation software. I discussed this with Richard, who will be my client for the project. He has asked me to develop a program that will maintain the accuracy and detail necessary to maintain a competitive edge over the other ten teams contesting the championship, making the system faster, more efficient, and more versatile.

#### Formula 1

Formula 1 is a racing series in which cars built to strict regulations race over 305km to reach the finish line first. The emphasis is clearly on having the fastest car. However, as the sport and technologies have developed, the time-gaps between cars have narrowed significantly, which has brought new technologies to the fore.

In the past, the focus was always on developing the car, as this was where the most time was to be found. Engineers developed parts for the cars that improved the performance by ‘Half a second per lap’, for example. During the course of a race, held over between 44 and 78 laps, this could amount to up to 40 seconds of time – enough to be 4km ahead of the person in second.

This has been progressively restricted as the cars have converged on the same level of performance, and it is now a significant gain to find one-tenth [per lap]. This could be as little as 5 seconds through the race, which means that gains in other areas can become much more significant. It is also worth noting that the major performance differentiator – aerodynamics – requires massive computing power and that the research into finding this one-tenth can cost up to £1m.

It seems logical, therefore, that teams would search elsewhere to find more efficient performance gains.

#### The Development of Strategies

In the early-to-mid 2000s, the Formula allowed two different tyre manufacturers to develop tyres, and for the teams to choose which tyres to use. It was quickly found that the fastest way to complete the race was to design and use a very rubbery tyre, which could provide a lot of grip, but that wore out quickly. As a result, it would need to be changed during the race. The car would also have to be re-fuelled during the race to avoid the weight penalty of carrying up to 150kg of fuel around in the 550kg car.

Since a ‘pit stop’ would cause the car to be stationary for up to 10 seconds, and lose up to 30 against cars that did not stop, timing these stops and getting them to the minimum possible time was critical. Put simply, arranging one’s race to have one fewer stop than your rivals would be worth the equivalent of £5m of aerodynamic development.

This developed the art of the ‘strategy’. A strategy refers to the timing of pit stops throughout a race, and is made up of ‘stints’ of a number of laps between stops. Strategies soon became the most important part of a race weekend – with so much time to be gained and lost it was vital to plan them correctly.

Engineers set to work with pencils and paper to calculate a trade-off between the wearing tyres (they provide gradually less grip as they are used – known as ‘degradation’) and the number of stops. However, with the development of super-computers and the proliferation of computing and ICT, strategy engineers soon turned to software to do the calculations for them. They programmed the computers to run thousands of different strategies, with differently timed pit stops, tyres and race eventualities, and collected a neat printout 12 hours later with the perfect strategy for the race.

Often, this became useless before the race had even begun, when a car stalled on the grid or crashed at the first corner, but when it worked it brought massive rewards for the team. Ross Brawn (OBE), hailed as one of the best strategists ever to work in the sport, was part of a championship winning team in eight of the eleven seasons between 1994 and 2004.

### Modern Strategy Computing

The development of strategic computing and race simulations was accelerated once again in 2007 when regulations changed to allow only one make of tyre to be used. However, for the first time since the start of the century, teams could elect to use the grippier but less-durable ‘option’ tyre, or the opposing ‘prime’.

This led to many more strategic permutations as there was now a trade-off between the faster tyre and the increased degradation, as well as balancing this with fuel use. Both types of tyre also had to be used, and fitting the correct tyre into the race at the right time was massively important in ensuring that the cars finished as high as possible.

In 2009-2010, further regulations changes banned re-fuelling and restricted overtaking, meaning that often strategy was the only way to improve on the car's qualifying position, and hence became vitally important. It is no surprise, therefore, that Ross Brawn once again won the world championship in 2009.

Finally, in 2011, a new brand of tyres was introduced that wore out much faster than before. This forced teams to pit more often, and also meant that the optimum strategy was much harder to calculate. Teams will frequently stop up to three times during a race; the tyres will degrade very quickly. Stopping one less time is still a large benefit though, so it must be carefully calculated.

Modern teams use supercomputers to analyse thousands of strategies in real-time, and can react to what other teams do very quickly. If a safety car is required on track – slowing down and bunching up the pack – a team can capitalise on this and gain positions in the race.

### Brief

I will create a solution to the problem of planning and predicting the outcomes of the thousands of different permutations of strategies that can be executed during a Formula 1 Grand Prix. The system will analyse all permutations to find the fastest strategy possible within the limits of the regulations using timing data provided by Governing Body the FIA, for all cars, and then run them all simultaneously to simulate a race. Individual strategies can then be altered by the user to investigate the effects of this. The system should provide a detailed output of the time loss in different strategies allowing comparison and selection of the optimum, and minimise the complexity of user data input and outputs to the user.

## [Analysis of current system](#)

### System Function

In summary, the current system is able to receive input data about the cars and the performance that they can exhibit, before displaying this data. The user can enter more data, modify it, or change internal parameters, which all affect the output graph. This is a graph of cumulative time against number of laps, normalised on any chosen driver's average lap. It can also provide a graph of the time gaps around any given driver.

The system can also run the graph based on live data, which is fed in through another system. This allows decisions to be made as the race progresses.

The main data input for the system is in XML format. This defines the driver names, pace parameters, and other metadata. The XML is produced by a dedicated team – the Strategy Team – who analyse their competitors' times to calculate the pace. This data is partly read in dynamically, but there is also data provided by the team in the form of an excel spreadsheet that has to be copied in by hand.

This data input, once loaded, is displayed on a table. The 22 drivers are listed as rows and the parameters in columns, which are populated by the XML file. Edits can be made from this screen to assess the impact of particular changes on the end result.

There is also a small section of the window to add pit stops; the strategy can be altered by adding, removing and changing the stops. Finally, there is an area where different parameters are stored about the four types of tyre. This can be updated in the same way as the driver's pace parameters.

The processing works by taking the driver's base lap time, calculated as a pace, and adding or taking away time based on the age of the tyres, the decreasing fuel load, and the tyres that the driver is on. Through the course of a stint, this generally causes a slight decrease in lap times before the next pit stop. The pit stop is represented by a simple time loss.

The system is able to split a lap into six distinct sectors too, which provides more accuracy of timing information throughout the course of a lap. It also makes it much easier to simulate overtakes – particularly on lapped runners – because these can be carried out in the same sector that the drivers meet. This is a more accurate model of overtaking. However, it requires much more data, which I cannot get access to.

When multiple drivers are combined in a race simulation, more factors come into play. In this situation, there is overtaking to worry about – the system recognises if a car behind is faster in a straight line and faster over a lap, and then allows for an overtake to take place. This is an accurate model of overtaking, however, previous systems used a probabilistic approach, where there was a certain likelihood of an overtake occurring, which more closely models real life (where there are other variables at play).

If drivers cannot overtake, they will get stuck behind other, slower cars. They therefore have to match the pace of the car in front, and their own pace parameters become irrelevant, even though the tyres still wear and the fuel is still used. It is possible to allow the tyres to wear less, and for the driver to use less fuel, but this system does not support it.

To output the graph, all of the timing data is calculated and the simulation finished. The complete set of values is then presented to the user – the axes' scales can be chosen by the user to see the full race or just a small number of laps, and the lines adjust accordingly. The lines for the graph are plotted in a defined colour for each driver between the recorded timing points. Time is on the y-axis with number of laps on the x-axis; the gradient represents the lap time and the curve represents the total cumulative time for the race. The aim, therefore, is to have the curve as high up the graph as possible, but also on top of everyone else's!

The strategy lines themselves have handlers for right-click events, which add pit stops at the given location. This will automatically update the table at the bottom of the

screen which displays the pit stops as a table, so the user can see all of the information updating as soon as it is added.

The axes can be adjusted manually by the user with on-screen controls, meanwhile the pit-table section is visible at all times. This means it can be quickly adjusted if necessary. The data input table, graph of time against laps and the third graph showing the delta between the reference car and other cars are all selectable tabs. They can be dragged into a new window if necessary for large scale viewing by the strategy team at the track, who have a 30" monitor to display it on.

As such, all of the controls on the windows are auto-scaled to fit the window, regardless of the size it is (although there is a given minimum). The system is almost entirely contained within one single window, but the movable tabs do allow for some flexibility with this.

Finally, it is worth commenting that the system will update live throughout a race to allow the team to make decisions based on real-time results. This requires the system to tap into a data feed and also validate the information that is provided to make sure there is an accurate plot of the race. Every time that the system is pushed new data, it re-calculates the strategies and re-displays them. Pit stops are also recognised live, and can be modified live as well.

### System Type

The System is programmed in a high level programming language, C#. It utilises object oriented design and also implements an MVVM structure. This means that there is a model, which includes all of the data and references for the project, a view, which provides the output data, and a 'ViewModel'. This is the interface between data and output. By structuring a program like this it is very simple to populate the ViewModel with data from the model; the view can also interface with the ViewModel to handle any data that is input on-screen. The view is always created from the ViewModel, which separates it from the data and makes data changes easier.

This works particularly well with the database concepts that are used at Red Bull Racing to store data efficiently. The database allows massive amounts of data to be stored on the servers, which can then be pulled to each computer that requests it.

The system also makes use of interfaces – made possible by the OO design – which means that in the event of regulations changes or different driver requirements, it is possible so simply program an extra class in. Even though the processing required by this class could be completely different, as long as the public subroutines are named the same thing the interface handles the calls and the system can use either class.

An example of this would be for the 'driver' class. If the regulations meant that a driver now required a 'jet thrust effectiveness' parameter, this could be updated in a new driver class without touching the old one, and the system would still function effectively.

### Sketches

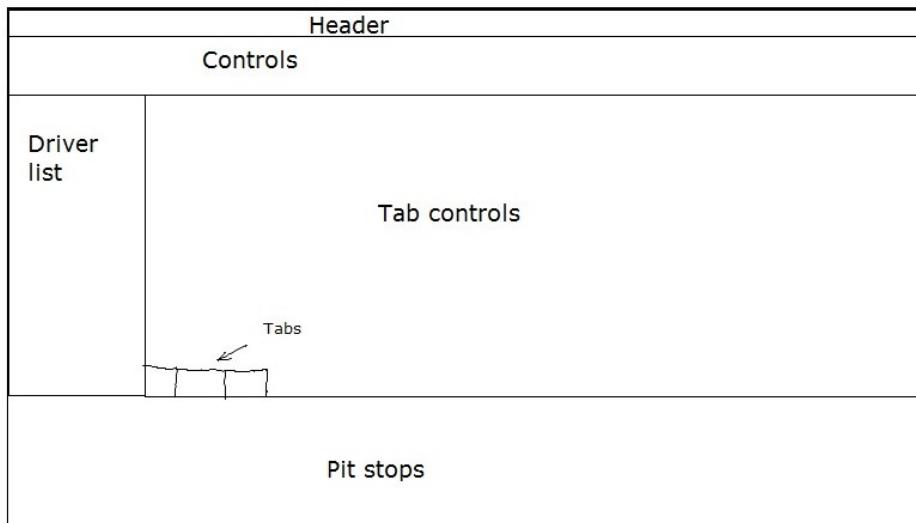
Due to the confidentiality and secrecy that surrounds all of the data held by the F1 teams, I am unable to provide screenshots of the current system. However, I have seen it and taken notes in detail, and I am now able to provide sketches to demonstrate the layout and functionality of the system.

These sketches are not proportionally accurate, but do provide a good representation of the actual program in terms of design and layout.

### **Window Overview**

The main window overview looks like a standard windows form. There is a header and controls at the top, and the data is displayed in panels below. The content of these panels is described in more detail later on.

All of the data is displayed in just one window, which is useful for the engineers as they do not have to arrange the windows and switch between views. This is something that I should try to emulate. Another useful feature is that the tabbed layers can be dragged away and will open a new window with the selected view displayed full screen. This is useful on the high-resolution displays when trying to see large amounts

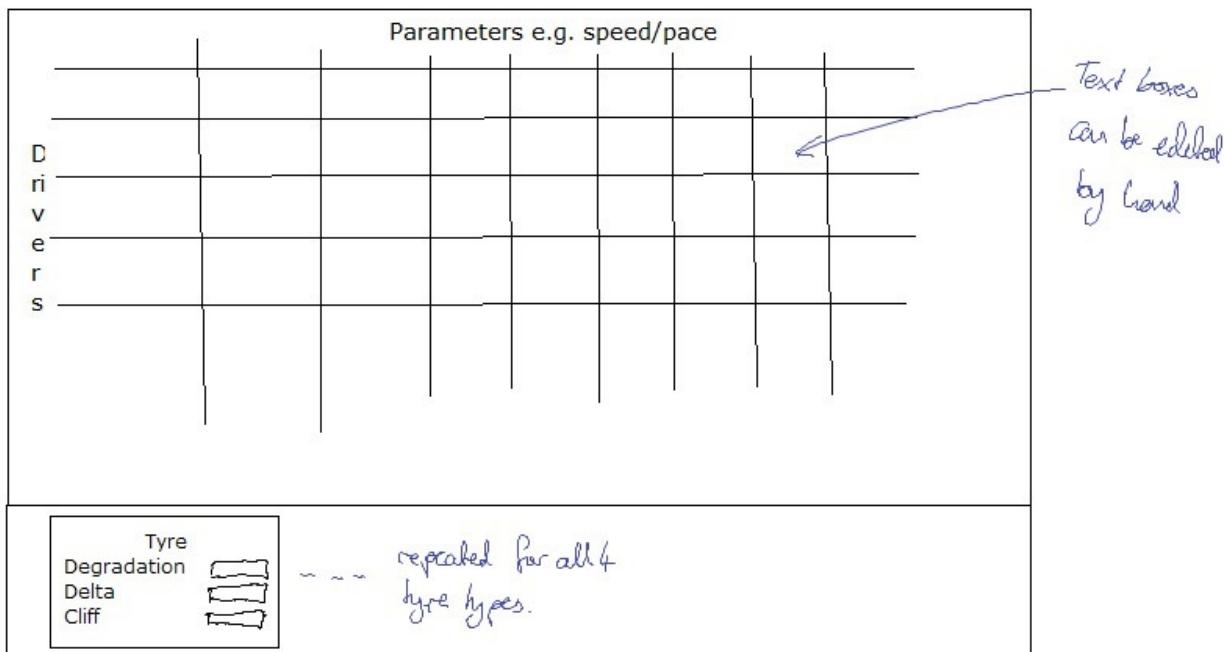


of data at once.

### Main View (Input, Output)

The main view in the window can be flipped between three different views. These are the data input window, and two display graphs. The input window is simply a grid of text boxes, which are populated using supplied values and then edited by hand.

There is also a section of the window where the tyre parameters can be input. These are general for each driver and therefore do not take car performance into account, which would be a good thing to try to improve.

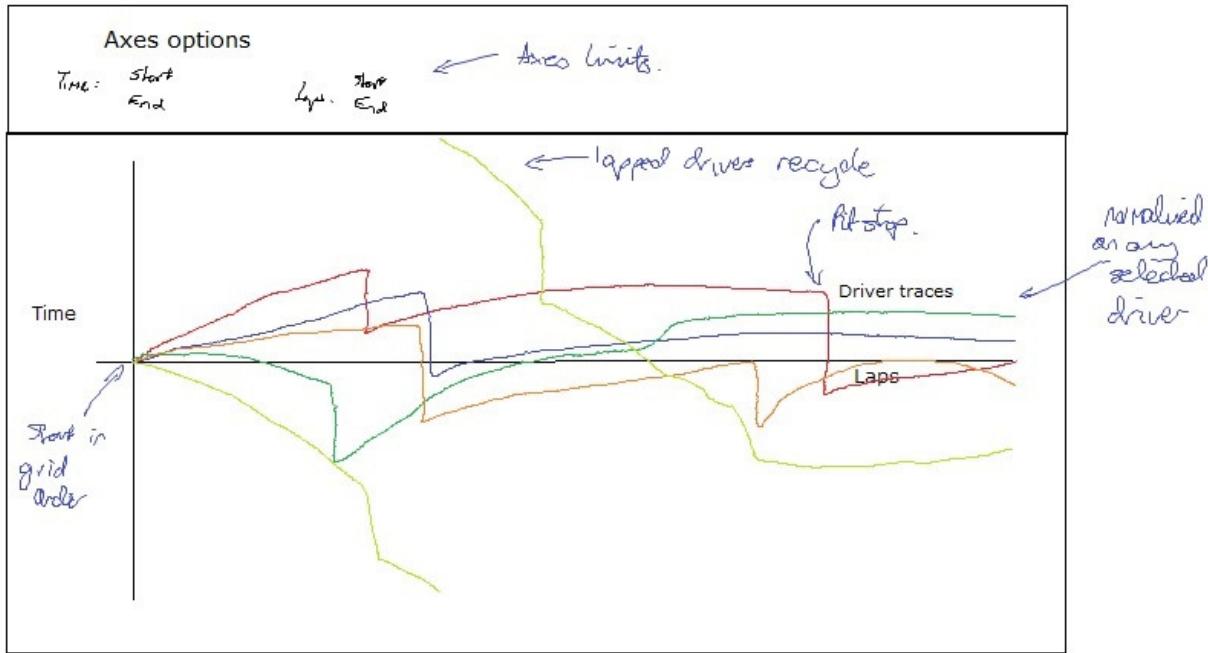


The main graph that is used is a graph of cumulative time. However, to allow more detail to be shown, the system normalises the graph on a selected driver's fastest lap time. Breakpoints can also be selected, so that the graph can be normalised on the average up to a point before the end. This is a useful tool in the event of safety cars, which cause a large shift in the total race pace.

The graph will populate live as the race progresses, including simulating pit stops and overtakes as they happen. If incorrect data comes into the system it is able to correct itself when more data comes in, by working backwards to fill in gaps in the timing. For example, if a sector time is missed, the system is able to work backwards from the lap time to generate the correct sector time and therefore update the graph.

Another useful feature of the graph is that the lapped cars will continue to be displayed. Once they have dropped off the bottom of the screen, they appear as being ahead of the lead driver. A time loss is then demonstrated as the leader (and subsequent cars) have to get past the lapped car. This is helpful in avoiding traffic and

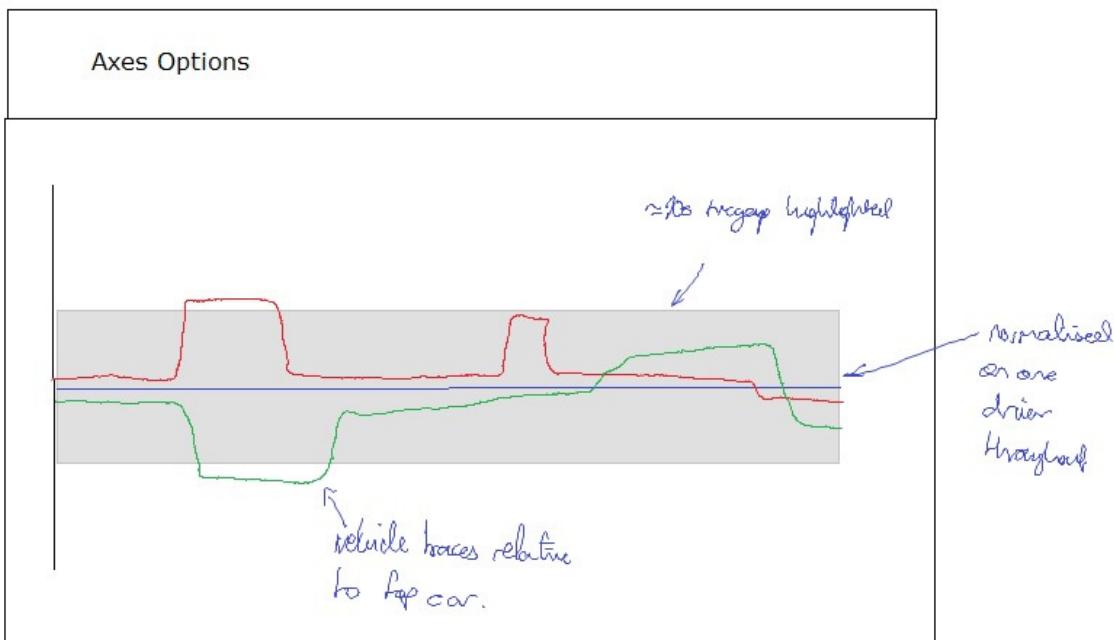
making sure the driver is aware of what may be happening ahead or behind on the



track.

A graph that is also used by the engineers is the ‘pit now’ graph. This displays the selected reference driver’s times as a completely straight line throughout. The other drivers then have their cumulative time relative to this displayed. The pit now graph has a shaded section which shows the pit time loss ahead and behind the driver. This makes it much clearer and easier to tell when he needs to pit – it demonstrates where the driver will be able to pit and return to ‘clear air’ with no cars ahead, and when it would land him in ‘traffic’.

The graph does not show an accurate representation of the race but the mathematics involved in displaying it are much less complex, according to Richard, and it is



therefore worth trying to include.

Both of the graph sections also have a series of axes options, which allow the maximum and minimum displayed values on the axes to be displayed. This means that the window can be edited to display more or less accuracy of the data. It is displayed in the same window so that the effects on the graph can be seen instantly, as is the case for the pit options and the driver list.

### Sidebars

The windows include a sidebar for the driver list. This includes a simulated timing screen, showing the tyre that a car is on, the number of pit stops it has done, and also crucially, the gap and interval values for each driver. This means that the team can assess in real time how the cars are related numerically, as well as graphically.

Each driver also has a checkbox next to them, which defines the reference driver. When the box is checked, the graph adjusts to the average lap time of the selected driver, so that any particular driver can be displayed. This is useful because it is not possible to predict incidents or delays that could cause the car do be dropped down the field.

A screenshot of a software interface showing a driver list and a graph. The driver list table has columns for Position, Driver name, Reference driver checkbox, Gap, Interval, and Current tyre type. Alonso's row is selected, indicated by a blue border. The graph below shows five curves representing different drivers, with Alonso's curve being the topmost. A legend on the right identifies the columns: Position, Driver name, Reference driver checkbox, Gap, Interval, and Current tyre type. A note at the bottom of the sidebar states: "Drivers are listed in position order".

	ALONSO	<input checked="" type="checkbox"/>	3	1.801	0.213	P
Current tyre type						

Drivers are listed in position order

Legend:

- Position
- Driver name
- Reference driver checkbox
- Gap
- Interval
- Current tyre type

At the bottom of the window is where pit stops can be edited. The window includes options for as many pit stops as required to be entered for each driver. Each pit stop can have a lap number and tyre type attributed to it, so that when the race reaches a certain lap any pit stops on this lap happen.

The window also allows for various options at the pit stop, such as a change of front wing or a check of a particular part of the car, which can add time to the stop. This means that the team can accurately determine if it is worthwhile bringing the car in to stop when there is a problem. The car is sometimes not badly damaged enough to

warrant a pit stop if it will bring the car out into traffic, but if the track is clear, or a pit stop is due anyway, it can be worth changing.

Each driver's trace on the graph contains an event handler for the right-click event. Right-clicking a particular location inserts a pit stop at this location in the race, while bringing up a menu to allow the parameters of the pit stop to be edited. This will also update the table of pit stops at the bottom of the screen.

Driver	Lap	Stop 1		Confirm	Stop 2
		Tyre	Special		
				<input checked="" type="checkbox"/> e.g. front wing change	Space for more stops

### Analysis

The current solution is clearly an advanced piece of programming and I am not going to be able to improve on the level of technology used in it, although I can emulate a similar level. To improve the system, I will look for where performance improvements can be found, and aim to exploit this to produce an easier-to-use end result.

I will attempt to achieve the same level of accuracy as the current system, and there are some areas where I will try to exceed this.

The solution is programmed in a very efficient and readable manner, using a strong, object oriented structure. The use of interfaces, makes it robust and future-proof, but there are occasions where it does produce incorrect outputs. This is something that could be improved upon.

The system has been designed to make it as easy for the user as possible to get to the data that they want to see; it is intuitive in this way. This means that the processing can be done very quickly. In essence, the windows are also simple in their layout, with the main tabulated controls and then sidebars showing supplementary information.

I believe that there are specific areas that can be programmed in a more effective way for producing an effective, efficient end result:

The system is predominantly used during the race to see how it is unfolding graphically. However, I believe it should be possible to use the system effectively

before the race and get an accurate picture of what to predict. This will be achieved by making sure that the inputs such as tyre degradation and fuel consumption are specific to each driver. Being able to use the system before the race allows for effective optimisation of the chosen strategy, and this ought to be very useful for the team to see.

It also seems that the user will have to spend a long time inputting data. Where the time could be used for analysing different strategies, it is instead used to copy across large amounts of data from one document to another. This can certainly be improved, by automating the data entry process.

The level of simulation is also limited, and there appears to be little function available for optimisation, save for the user's inputs. There is time to be gained in the race by optimising the pit stop laps, and also by optimising one driver's strategy against his competitors, as well as against the clock. This is the purpose of Monte-Carlo simulations, which are used in a limited way in another tool used by the team. While the potential for me to use a similar algorithm is limited, I believe that there are areas in which the strategy would benefit from a similar type of analysis.

Finally, the graphics used by the system take a long time to process and display. This can lead to a jerky display and means that the actual picture is delayed a little from the live event. Given all of the processing that must also be completed, the delay can be very large, which can cause problems for the team when they try to make decisions in split-seconds. I think that the way that the graphics is used and displayed could be simplified without loss of accuracy or resolution to improve the speed of display, especially on a large monitor.

## Detailed Brief

### Further Research

The **Monte-Carlo simulation** has come up a lot in my research into F1 strategy systems. I want to investigate how this works, and if there is the potential of emulating the algorithm in a simpler form.

According to *Palisade*<sup>1</sup> – a large risk assessment enterprise, the Monte Carlo simulation takes a scenario in which there are potentially hundreds of variables, and runs a simulation using a random value for all of them. Each of the variables must be modelled by a probability distribution. By choosing values at random within this distribution, for every one of the variables, a simulation can be attempted with definite values.

Running this simulation thousands of times, and recording both the output and the probability of the inputs chosen occurring, the system can build up an accurate picture of the most likely scenarios. Having processed them, it can also drop back into its

---

<sup>1</sup> [http://www.palisade.com/risk/monte\\_carlo\\_simulation.asp](http://www.palisade.com/risk/monte_carlo_simulation.asp)

archives and find the particular result that occurred when the variables that have appeared in real time were tested. It is a very efficient way of processing through large volumes of probabilistic inputs.

Its use in strategic computing is to analyse the probability of things like rain, overtaking, and tyre performance. However, it is used more generally when an algorithmic solution is impractical and an iterative solution (e.g. trying every possible combination) would take too long. Clearly, for 100 variables (in mathematical terms, 100 dimensions) solving an equation by iteration would be ridiculous.

Instead of a Monte-Carlo simulation **differential calculus** can be used for optimisation. Where an expression can be generated involving just two unknowns, it is possible to find the minimum or maximum value of the expression.

In our case, we can develop an expression for the time taken when a particular strategy is run, in terms of the length of the race stints that are chosen. By then differentiating this expression, the rate of change of race time with respect to strategy length can be found.

When the rate of change of time is zero, we know that a minimum time has been achieved. We cannot increase or decrease the stint length to get a lower time, hence the stint length at this point is an optimum. If we can calculate this optimum, finding the optimum stint length becomes very easy.

See Appendix 1 for details of the expressions calculated.

The **Current F1 Regulations** will be vital in understanding the nature of the problem that I have to solve. These can be found online at the website of governing body The FIA. They explain in detail the technical and sporting requirements of the cars and teams. We are most interested in the sporting aspects – this is where the actual racing is dealt with.

Specifically, we are interested in articles 25 and 29. These deal with tyres and fuel respectively. Article 25 specifies that only three sets of prime and option tyres may be used in the course of a race. Similarly, there are four sets of intermediate tyres, and threat the extreme wets. At least one of each of the ‘prime’ and softer ‘option’ tyres must be used during a race. To ‘use’ a tyre, it must be fitted for at least one lap; it must have left the pit lane. This includes before the start of the race as the car goes to the grid. The tyres that the top 10 cars use in qualifying must be used during the first stint of the race, hence they will not be new. This means that there may be a loss of performance in the first stint.

Article 29 on fuel prohibits any refuelling during the race, which means that pit stops will be faster than if this was included. However, the car will have to be fully loaded on fuel before the start of the race, which means that the fuel effect will have to be considered. 20% of the starting weight of the car is fuel.

### Potential Improvements

- Simulation Accuracy
- The overtaking model can be improved, taking into account the likelihood of an overtake occurring and working out if it has occurred using a probabilistic approach.
- Data can be tailored to each particular driver, so that his driving style is taken into account when working out how he and his car will perform during the race.
- Data Input
- Data can be collected from documents provided by the FIA that I can use and analyse inside the program. This would speed up the data input.
- Optimisation
- The driver's strategies can be optimised by the system so that they will produce the best possible outcome automatically, without the need for user input.
- It ought to be possible to predict more effectively the strategies undertaken by other drivers, so that the team is ready to react more closely to these.
- Graphics
- The graphics can be simplified without significant loss of accuracy so that the screen can be refreshed on a more regular basis. This will give the race engineers a better chance to get the decisions correct.
- Graphics can also be re-arranged on screen to make more use of secondary windows. These can be used for data entry so that the main screen can be kept clear for the graph, for example.
- Validation
- In some cases, the system can produce erroneous results. I think that running the simulations in a different way will make the results more reliable.

### Identification of the prospective user

#### Client

My client will be the current head of Software Development at Red Bull Racing, Richard Dutton. He is a programmer and designed the current system that is used by the team to analyse strategies in real time.

He is very computer literate and is therefore able to converse in technical terms, which means that any user manuals that are provided can be written in technical language.

He will also be my end user, as he is in charge of any interface with the system. However, the system will also be used by the Strategy team – a small unit who will use it to work out exactly when is optimal to pit. The race engineers Guillaume Rocquelin and Simon Rennie may request information from the system, so it will need to be provided in a way that they are able to act efficiently on.

#### Red Bull Racing

Red Bull Racing is a Formula 1 team based in Milton Keynes. It has won the past three constructors' and drivers' World Championships, and has a significant lead in

the 2013 championship. The team has a supercomputer used for aerodynamic modelling and strategic simulation.

The system will have to run on their computer systems and will therefore need to retain any data formats that are used at the HQ. It will also need to suit the graphics capabilities of their laptops and ensure that it can run fast enough to provide an interactive processing interface.

## Identification of user needs and acceptable limitations

To conduct the interview I travelled to Milton Keynes to meet Richard, so that we could discuss exactly what he required from the system. The meeting was very valuable, and has allowed me to produce the list of initial user requirements. The interview is documented below.

Due to security and secrecy constraints, the interview was not recorded and this is therefore not a transcript. Every effort has been made to ensure the accuracy of the record but cannot be interpreted as direct quotes.

### Interview

During the interview, I set out to identify the extent of the project, as well as some of the standards and requirements to be implemented. The first part of the interview deals with the system's function, and the second part is about the specifics of the calculations that will be implemented.

#### **Can you give me an outline of the process required to simulate a strategy here at Red Bull Racing?**

Yes. The first thing that is done is that the Strategy Team uses a combination of tools to come up with data for the various different parameters that we simulate. This is saved as an XML file, which is then put in the database layer that we have here.

We in the computing department then load this data into the system that we have. This is automated to save time. There are also some track-specific parameters, including tyre degradation and fuel effect that are inputted from an excel document. This is hand-typed. Finally, we have some alterable parameters which will affect how the race will play out, and we have to input these by hand based on experience.

The pit stop laps are selected based on data from practice. These can be changed very easily by hand to edit the strategy and affect the result.

The simulation runs sector-by-sector by simulating these times for each car. This is based on their pace parameters that have been entered previously. At each sector, the race is assessed for overtakes – this will initialise passes if the car behind is much faster than the car in front. When a pit stop occurs, the sector time that is between Pit entry and the Pit Merge Point will be increased to the known value for a pit stop. In this time, cars behind the one that has pitted will pass it. If overtakes cannot be

completed, the car behind will remain at a set time gap until the opportunity to pass arises.

**It sounds like there is a lot of data that has to be entered in different formats. Is this a problem?**

In general, no. We only need to do it once in a weekend, but it does take a fair amount of time. It's inconvenient more than being a problem. Because each individual section of the team has their own tools and works using their own methods, it is easiest to let them produce their data and then we can deal with it.

**Would automating the procedure be beneficial?**

I can see it being a challenge! At the moment the data is based on what we know for our team, and some of it is assumed to be the same for the whole field. So in that respect getting some data that varied between the teams and individual drivers would be useful.

**How and when is your system used?**

We set up the simulation on Saturday night, to look at the way the race might pan out with pit stops. However, its main function is during the race. Data comes in from the FIA which we strip down and pull the timing data out of.

We can, in real time, see how the race is developing graphically, and we can use this to plan our pit stops in the remainder of the race. Our tools will let us plan to avoid traffic later in the race. It also means we can predict what the drivers on different strategies will do, so that we know when we have to make passes and when we can sit behind other cars, knowing that they will get out of the way.

We can also play back races in the past, so that we can analyse the races in retrospect. This can help the team and drivers know where they can perform better, and can also allow the strategy team to look at how they can alter strategies in the future.

We have other systems which run elements of Monte Carlo simulations to look at what might happen in the race, although they're not fully developed yet. The system I have here isn't used to optimise strategies.

Equally, it cannot tell us what we should do in a major event, such as a change in conditions, a safety car, or a mechanical issue. For this we refer to other systems and algorithms.

**So data is read live from the FIA, what format is this in and what other formats do you use?**

The data is supplied to us in an odd format. It tells us how to draw the timing screen which all of the teams use as a go-to for the majority of timing information. We read this as a stream and pick off the bits that are useful, like the times and speeds of the drivers.

Predominantly, we save our data in floating point format. The data for drivers, tracks and teams is saved in the database layer that we have here, and is called upon whenever the program needs it.

In terms of units, we use the FIA standard to keep it simple. This is times in seconds and speeds in  $\text{kmh}^{-1}$ . We measure fuel in kilograms (this is independent of the temperature so is a more accurate, and useful, measure), and the time effect of fuel in  $\text{skg}^{-1}$ .

It's also worth noting that time is considered to be increasing in the positive direction, therefore the effect of tyre degradation and fuel is a positive number. This means that more degradation will produce a larger lap time.

**Do you use text files for temporary or permanent data storage?**

Not especially. They are a useful and very easy format, but file I/O is slow so we try and keep things inside main memory as far as possible. Outputting data to a text file is a useful way of keeping it saved for later analysis and of transferring data from one system to another.

**How are the settings stored – the units and the driver names, for example?**

The units, driver names, track names, and anything else that we won't expect to change over the course of a season will be hard-coded. This isn't perfect practice, but because all of the systems here are linked it is very easy to push out a new release of the system. If we need to change the driver names, as we do before the start of a season, this can be done by editing the code.

**Is there a requirement for any data security or encryption?**

No. We have a database which is fully secured here at Milton Keynes, and everything is hard-wired anyway. Once the race is over and the result is known the data is pretty much all in the public domain anyway.

**What are the hardware limitations?**

The pit wall uses Lenovo laptop chassis; they are quite highly spec'd but have severely limited graphics capabilities. They have 8GB of RAM, which is sufficient for quite substantial calculations, but when it comes to updating graphs in real-time the system can get quite slow and jerky.

This is a particular issue on the 30" monitors that are used in the ops rooms. Increasing the size of the graphics to fit this area requires a lot of processing. The window will definitely need to be scalable.

**Do you require significant data validation?**

When it comes to the parameters we use to work out the drivers' pace, clearly these need to be within sensible bounds. Negative degradation, for an example, should be flagged up.

The accuracy of the timing system is also important. It is worth checking, for example, the sum of the sector times is equal to the lap time that you are using, to confirm that the data is correct. This would come under a verification check.

It is far more valuable to the team that the system works brilliantly for 90% of the time than is only average for 99% of the time; as a result we can tolerate a few exceptions being thrown as long as they don't cause the program to crash. Because the function is so important, there is less emphasis on fancy aesthetics too.

Clearly the interface has to be logical and smooth – this ought to contribute to the ability of the program to function efficiently and effectively. But it need not be designed with optimum aesthetics in mind from the start.

### **What are the variables that can be altered within a strategy?**

The ‘strategy’, in the context of the program, is based on pit stop laps and the tyres that are used for these stints. This is really all that can be altered to alter a strategy; the rest is pace. We use pit stop laps so that there is less calculation to be done when calling the pit stops. Previously, we used the start of the stint and the length of the stint, but basically one of these variables was redundant, which meant passing it around through the program was not efficient – using just the lap that the stop is planned on now more useful to us.

However, there are some permutations when it comes to the tyres. We can use one of four types of tyre, but they can also be brand new or slightly ‘scrubbed’, where they have been fitted to the car in a previous session. This alters their performance characteristics. This is not currently simulated through.

What is simulated is the tyre ‘cliff’, which occurs when the tyre has no more rubber left on it to provide grip and therefore begins to lose a lot of time each lap. We have to simulate this.

### **How do you simulate overtaking?**

Again, this was something that has changed from the previous system. In the past, we used a probabilistic approach which modelled real-life quite accurately. However, now, we use a system that is flat; if certain conditions are fulfilled it allows an overtake to take place.

This requires both a pace delta and a speed delta – we require the car to be lapping faster by a certain amount and be faster in a straight line. This is sufficiently accurate to model the races by.

### **And what about the DRS?**

The effect of the DRS is taken into account when we set the required overtaking parameters. It is a bit of a ‘fiddle’ factor, but as long as it produces the correct results we are not too concerned.

**What functionality will you require from the output graph?**

It would be good if the graph had axes that were user-defined. This means we can zoom in on a particular area to see exactly what is happening there.

We also currently have two different views – an absolute and a relative time normalised graph. These have slightly different purposes and they look very different. Both are useful. We also have on-click events on the lines on the graph. It would be good if these could be utilised for adding pit stops or editing the strategy, because this is a very intuitive and quick way to alter it.

**How should the end result look?**

It would be good if the program was all contained in one window. This would make it much easier for the team to work on and analyse data in the ops room, because they have a very large monitor onto which the system will be projected. If this is all in one window it is much easier to scale up and keep the proportions.

The layout needs to be logical and sensible but we don't need fancy graphics. Not only will these increase processing time but they may also detract from the usability of the system, and this is absolutely key. Things need to be clear in the outputs too, so using colours on the graph to represent the different drivers is a must, for example.

**Summary of Limitations Discussed**

Richard commented that if I can complete the project it would effectively replace the roles of the entire strategy team, which is a fairly large group of dedicated race engineers, and two software developers, for a race weekend. Clearly, this is not going to be possible if I aim to achieve the same level of functionality and detail that their systems use. As such, it was important to identify the limitations of the project.

- Data accuracy
- Most of the data that I have access to shows data to only one tenth of a second. As a result, the processing will not be quite as accurate as in the current system.
- I will also not be able to replicate the exact parameters of cars (such as tyre degradation and fuel effect) because I do not have access to telemetry data.
- The system cannot be run live because I do not have access to the feeds. It will have to be used prior to the race to predict and plan strategies.
- Presentation
- Given that I have limited time, the presentation of the solution is not expected to be perfect.
- Validation and verification
- There ought to be checks to ensure that data is reasonable. However, long and time consuming checks against previous data are not worth implementing.
- The system will be designed for optimum functionality. Exceptions need to be handled to prevent it from crashing but the presence of bugs is to be expected.
- The current system can try and work backwards from the given timing data to validate the results – it uses sector and lap times to build a complete picture of how the car is progressing around the circuit. This would be very difficult to program so it will not be implemented.

- Strategy permutations
- The option to iterate through different strategies automatically to find the best one is something that is not required. It could be implemented if there is time at the end, but it is not a requirement from the outset, because working by hand can achieve an acceptable result. Other tools can also do this more efficiently on supercomputers.
- Weather and safety cars need not be simulated, because they will take time and are fairly rare occurrences. It is also very difficult to get enough data to make accurate predictions about what will occur if it rains.

## Initial user requirements

### List of requirements

1. Data Input
  - a. The system will automate the process of inputting times from FIA PDF documents. Data should be input from the first, second and third practice session, and qualifying.
  - b. The system will perform calculations on the input data to define parameters about a driver's pace. These include:
    - i. Top Speed
    - ii. Base lap pace
    - iii. Option tyre delta
    - iv. Tyre degradation (prime and option)
    - v. Fuel effect
  - c. The system will allow the user to input data specific to each individual track. This includes:
    - i. Overtake speed delta
    - ii. Overtake time delta
    - iii. Pit stop loss
    - iv. Fuel consumption
  - d. The system will save the input data in race and session specific files, which can be accessed from previous years to save time.
2. Lap time archiving
  - a. The system will save data from the input sessions to text files, which can be accessed at any time to view past lap time data.
  - b. The system will calculate fastest laps and average laps from previous sessions so that these can be displayed to the user.
  - c. The output of data will be tabulated for ease of viewing.
  - d. Data should not be able to be edited or corrupted during viewing.
3. Strategy optimisation
  - a. A strategy is composed of:
    - i. Stints, of a calculated length. These are interrupted by pit stops
    - ii. Tyre strategy, which is the order in which sets of prime and option tyres are used

- b. The system will find the fastest strategy for each driver based solely on their pace parameters, and not considering other drivers
  - c. The system will set this fastest strategy as the default strategy for the driver.
  - d. The fastest strategy will be displayed to the user in the form of stints. Each stint has a length and a tyre type.
  - e. The system will allow the user to edit the strategy by changing the tyre type and the stint lengths.
4. Race Simulation
- a. The system will put all of the drivers' strategies together in a race.
  - b. The system will run a lap-by lap simulation of the race based on the defined pace parameters for each car.
  - c. The simulation will include:
    - i. Pit stops
    - ii. Overtaking (and the effects of 'traffic')
    - iii. Tyre degradation and delta effects
    - iv. Passing lapped cars.
  - d. The results of the race simulation will be displayed graphically, with the cumulative time (normalised on any selected driver's average lap time) plotted against number of laps for each driver.
  - e. The graph will have adjustable axes to allow the user to change the detail of the view.
  - f. The strategy can be changed from the same window as the graph to see the effects visually.
5. Material Requirements
- a. The user will be able to work interactively with the system.
  - b. The system will not exceed the RAM limitations of the computers it is running on.
  - c. The system will use simple graphics to prevent the display from becoming jerky.
  - d. There is no need for encryption or data security.
6. Settings
- a. Settings need to be alterable but do not need to be altered within the running program.
  - b. The units should be in line with FIA standards
    - i. Speed:  $\text{kmh}^{-1}$
    - ii. Time: s
    - iii. Fuel weight: kg
    - iv. Fuel consumption:  $\text{kg.lap}^{-1}$
  - c. Time is increasing in the positive direction

*I will now analyse the initial requirements to help direct the remainder of my research.*

### Targets

Having discussed the potential for the system, I can now propose some reasonable targets for how the system should perform, and what should be possible when it is finished. This is hopefully a list of what the program should be able to achieve. These will also form the basis of the data flow diagrams that will be completed later on.

I want to be able to collect data from the PDF files and process this to collect the car's parameters. I then want to be able to process these parameters to find the theoretical fastest strategy with no other cars to worry about.

All of the strategies will be loaded into a race. The race will be 'run' with overtaking and pace simulated throughout. The positions and time gaps at each stage through the race will be calculated. These will be displayed onto a graph of time vs. laps through the race.

Once the data has been displayed on a graph, the user will be able to edit their input data to alter the strategies. The pace parameters can be tweaked, and the pit stops and tyre strategies can be changed, so that the user is able to find the best possible strategy. The graph will refresh once the data has been input.

The system will also allow the user to save timing data. Once it has been input from the PDFs the specific timing data will be saved and output to a text file. These can be recalled by the user if necessary, so that past timing data can be easily recalled.

### Agreed Limitations

There will be no requirement to take weather conditions into account, and there is no real requirement to deal with safety cars during the race. This means that the race will effectively run from start to finish according to the same rules, making the processing easier.

There will be limited validation – enough to prevent crashes but because the system is very specialist, it must function efficiently. Excessive validation will take time which could be better spent improving functionality.

The aesthetics of the input and output are largely unrestricted to allow the best level of functionality, even at very high resolutions and display sizes. However, there is the requirement for the interface to be logically laid out, so that editing the data is efficient and quick.

The system does not need to optimise the strategy when other cars are considered as this can be done by hand. It can be optimised when no other cars are considered to give a guide as to how best to use the tyres.

Finally, I will be simulating the race lap-by-lap instead of sector-by-sector. This reduces the processing time required, and reduces the stress on the graphics. While it is not as accurate as a model, the calculations are very similar when applied to a shorter time period. I am also restricted in my access to sector times, which means

that it would be difficult to get the data in order to do update more often than once per lap.

### Hardware and Software Restrictions

I am not subject to particularly stringent hardware restrictions. The system should not be trying to complete Monte-Carlo simulations or undertaking any type of batch processing; it should all be interactive so steps that are executed at each stage mustn't be too complex.

I am restricted in software terms because my end user would like the program in program code; it must be a programmed system as opposed to a spread-sheet or database. However, it is likely that this is the type of solution I would choose anyway. For this reason this is not a significant restriction.

The graphics used in the system need to be fairly simple; anything too complex in the graphics will take too long to display. However, because I will only be processing lap-by-lap there will be fewer updates to the graphics, and the drawing can afford to take a little more time.

The system will have to run on windows OS, and on the AMD cores that the race team currently run. However, the team at the base will be using desktops that have Intel cores, which means that the code may have to be compiled differently for each machine. This is not a concern during the programming, but when the solution is released it is something to consider.

### Schedules

These schedules have been discussed and agreed with my end user.

The prototype for the system will be produced in time for a meeting with my end user on 27<sup>th</sup> September. This can then be analysed and developed into a full solution, based on the user's feedback at this stage.

The design and production of the code will take about a month from the completion of the analysis; therefore I will allow until the end of October for the first iteration of the system to be complete.

This allows time for the system to be tested initially, before a second version can be produced to take into account the results from testing. This will also mean that any issues in the programming that developed early on and were too difficult to correct during the production of the first solution can be fixed, resulting in a more efficient and more readable solution.

The completion of version 2 is expected to take until the start of December, at which point a thorough test program will be executed to ensure that the system is fully functional. Any changes that need to be implemented after this should not take too long.

After testing, an evaluation of the system can be undertaken to get some user feedback and to assess how the program could be expanded, if necessary. There will also be maintenance documents and user manuals produced so that anyone taking over the system will be able to understand how to use and maintain it.

This will hopefully be finished during February, so that if there are any more requested changes, or the system could be expanded, given time, this can be done.

### Data dictionary (from perspective of end user)

This section defines the parameters that are used by the current end user in his system. Some of them will be carried across to my system and others will be worked differently.

This is a general list of what particular values are generally named, which should be referred back to to avoid confusion at a later date.

Data	Description	Type	Example
Pace	The base performance of a driver, represented as a lap time.	Float	91.302
Tyre 'deg'	Time loss per lap on a particular tyre.	Float	0.3
Tyre delta	The amount by which the option tyre is faster than the prime	Float	-0.6
Fuel Effect	The time loss per kilogram of fuel in the car per lap.	Float	0.025
Fuel Consumption	The number of kilograms of fuel used per lap.	Float	4.2
Top Speed	The fastest attainable speed for a car, in kilometres per hour to one decimal place.	Float	309.7
Position	The driver's position in the current session.	Int	2
Lap Time	The number of seconds taken to complete a lap.	Float	91.865
Gap	The number of seconds separating any particular driver and the leader	Float	11.030
Interval	The number of seconds separating any driver and the driver ahead	Float	0.898
Stint length	The number of laps completed between pit stops.	Int	12
Tyre	The tyre type used for a particular stint	Int	1
Tyre cliff	The number of laps before the tyre loses grip significantly	Int	18
Driver number	The driver's race number. This is between 1 and 23, with no number 13.	Int	1
Driver name	The driver's surname	String	'Button'
Line colour	The colour in which the driver's graph traces will be drawn	Colour	'DarkBlue'
Overtake speed	The amount by which a chasing car has	Float	10

delta	to be faster than the car in front for an overtake to take place		
Overtake time delta	The amount by which a chasing car has to lap quicker than the car in front for an overtake to take place	Float	0.1
Pit stop lap	The lap on which a pit stop will take place	Int	13
Axis upper bound	The upper bound of the values displayed on a graph axis	Int/Float	18
Axis lower bound	The lower bound of values displayed on a graph axis.	Int/Float	32

## Data sources and destinations

### Summary

My system will read its input data from text files. However, the text files will be created from data that is simply copied and pasted straight out of a PDF document. There will also be some data that is input by hand by the end user.

The outputs will be far more wide-ranging. I will need to output a graph, which can be viewed in detail and can be altered by the end user. I will also need to save the calculated data to make sure it can be accessed at any point by the end user for reference purposes.

There will be data input from pre-set files which is processed and displayed in drop-down boxes on the windows, allowing selection of the right type of data and race. Some calculated data will also be displayed, requiring dynamic controls on the system.

### Sources

Entity	Data
PDF Timing Document	Timing data
Settings text file	Settings
User	Click events for system control

### Destinations

Entity	Data
Parameters text file	Pace and speed parameters for each driver
Race data text file	Timing data for race
Drivers sidebar (GUI Item)	Gaps and Intervals on each lap
Pit stops sidebar (GUI Item)	Pit stop data for each driver
Strategy graphs (GUI Item)	Timing data for race
Parameters data table (GUI Item)	Pace and speed parameters for each driver
Timing archives text file	Timing data from PDF
Settings text file	Modified settings

### PDF Files

The FIA Specification PDF files are provided at the end of each session over a race weekend. They contain timing data from the circuit's sensors – they can provide lap times, top speeds, and sector times. These files have no embedded text, and PDF is not a nice format to use. However, if the text from the PDFs is copied to a clipboard, it can be much easier to analyse as it copies into a very consistent text format.

The PDF files are typically about 100kb in size, despite containing only 10-15kb of usable data, which means that if they could be analysed and then removed, this would save space on the system.

The PDFs can be downloaded from online and saved locally.

### Text File Format

For any data that is read in or read out using the system, text files are by far the easiest way to store information. This includes files holding information about the current settings, for example, and data needed to populate the system's variables. Driver names, and track names and lengths, come under this heading.

Finally, any data that is calculated and output by the system, such as the results of the simulation, will be written to a text file.

File I/O is fairly well catered for in program code. It is possible to use direct IO operations, but streams are more efficient and allow data to be read and processed very quickly. This means that the text files are remarkably easy to use.

Text files containing different data types will be both a source and destination for the results of the simulation.

### User Inputs

The user will have to enter some text into the system by hand, and this will be done almost entirely through typing into text boxes. At this point, the data that the user has input needs to be converted into the intended variable type (it is always read as a string) and then it can be allocated to the correct variable slot. The user's inputs could include a degree of error or randomness, which means that validation is a sensible inclusion into the system.

There will also be some input through combo boxes, which will be used where a finite list must be selected. They are much simpler to use for data input, because the order of the indices is known when they are loaded. It is very simple to find the selected index and therefore return what the box is representing at any stage throughout the code.

Clearly, the user will use a mouse to ‘enter data’ although most of the mouse events are handled with pre-designed event handlers. I will need some of my own too.

Largely, the event handlers will call particular functions; the mouse is what controls the processing of the system, rather than being real, useful strategy data input.

### Strategy graph output

The strategy graph is a destination class for most of the data that is processed. This is where the timing data, including driver positions and cumulative time arrays, will be output to.

The graph will then plot these data against the number of laps completed, for each driver. It will, in essence, produce 22 traces of lap time, which will allow the effectiveness of any particular strategy to be assessed.

The graph will hopefully be separate from any actual processing of data. Instead, so that it can have multiple uses, it will be a generic graph class that plots a list of defined points on its axes. This means that the programming is more robust, and that again, a second graph can be programmed in parallel if necessary.

### Internal IO

Some of the data that is processed will be saved internally and then recalled later in the program. Provided that it is not too big, the data that is generated within the system will stay within the main memory of the computer. This reduces access time, which massively increases the speed of the system overall.

The data is stored in a defined type, which means that the operators all have defined functions. This makes the programming itself easier.

Whenever data needs to be referenced, it can be done from inside the program. Typically, the large amounts of data that are held will be inside the Program class, which means they are accessed using the identifier `Program.[variable]`

## Data flow diagrams

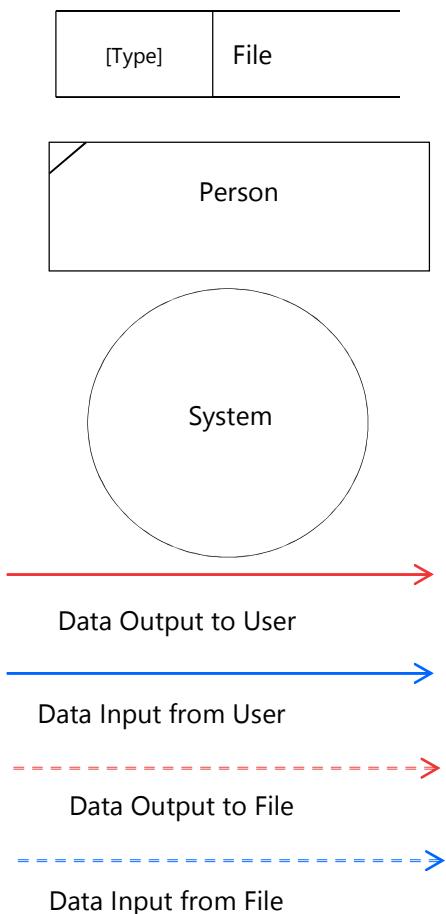
### Summary

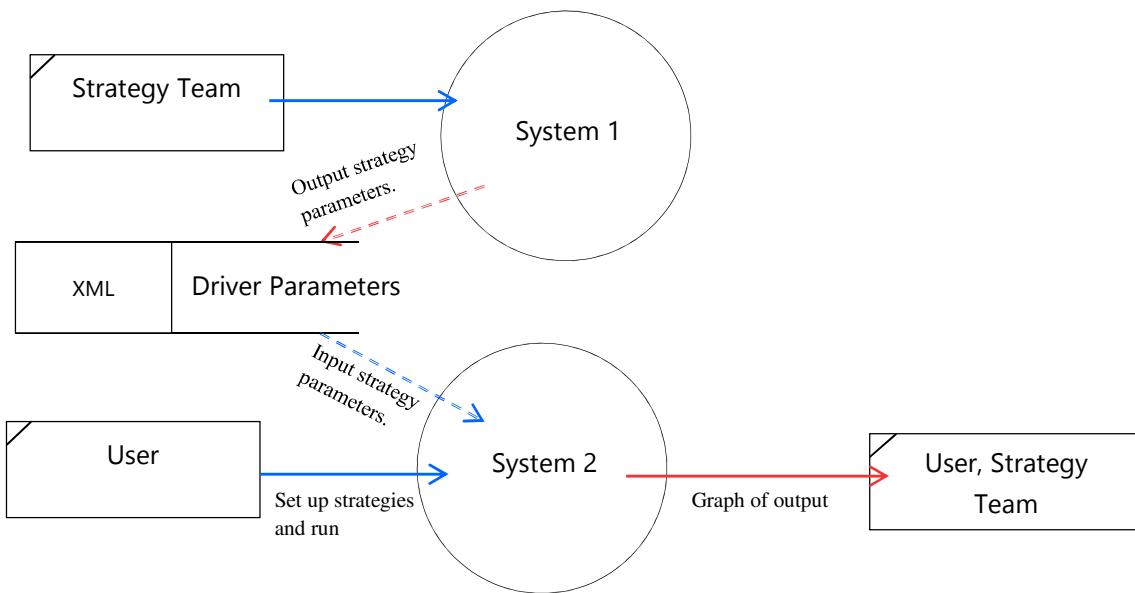
Data flow diagrams demonstrate the way that variables and data are passed between routines. Data is loaded from or written to files, and acted upon by routines. Data is passed between these routines as required for the processing.

On the next page is a demonstration of the data flow through the current system, which involves a lot of user interaction and file passing between different components.

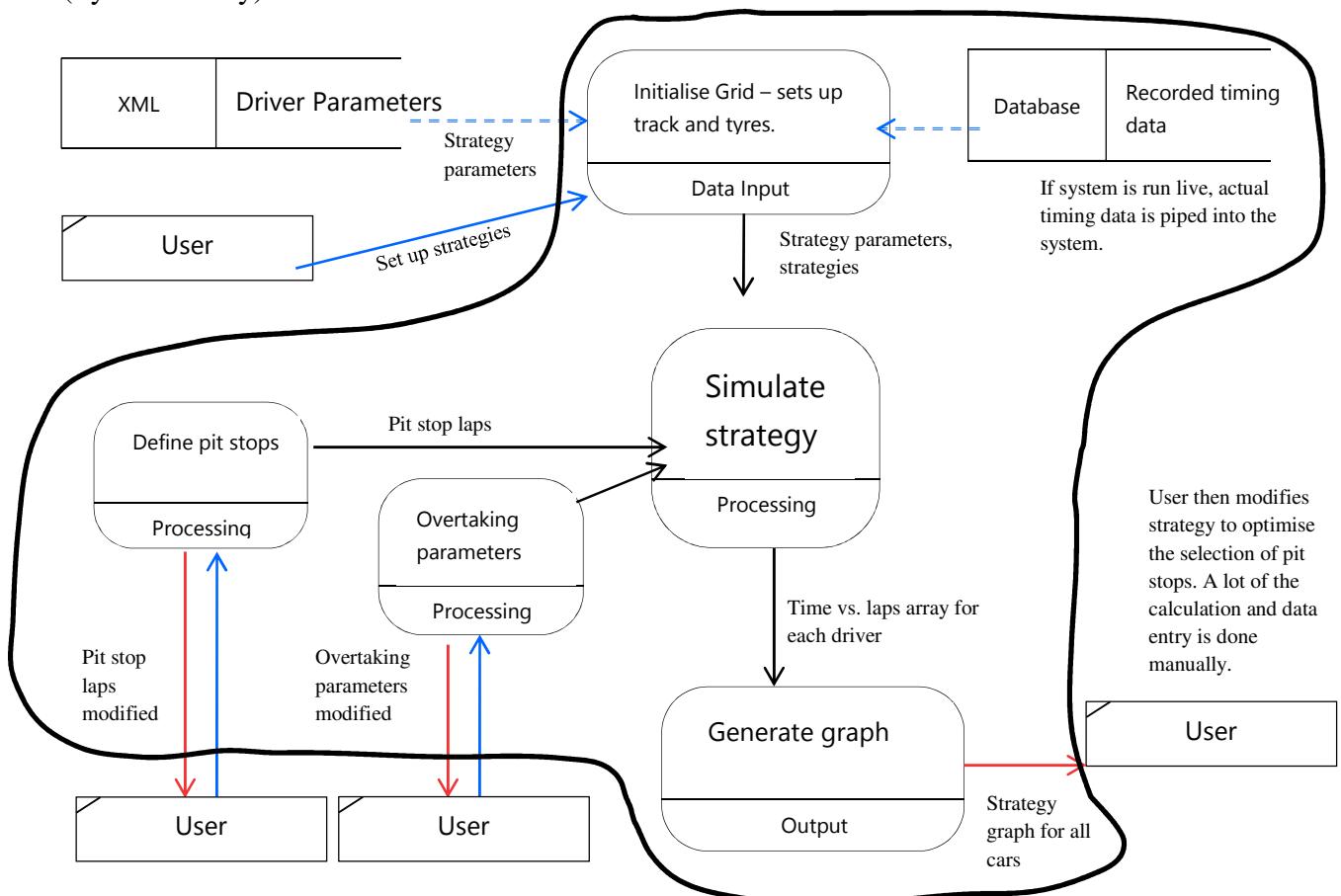
After this, a demonstration of what my system will need to do shows how it incorporates more functions, but requires less user interactions.

### Key



Existing System**Top Level****Processing Level**

(System 2 only)

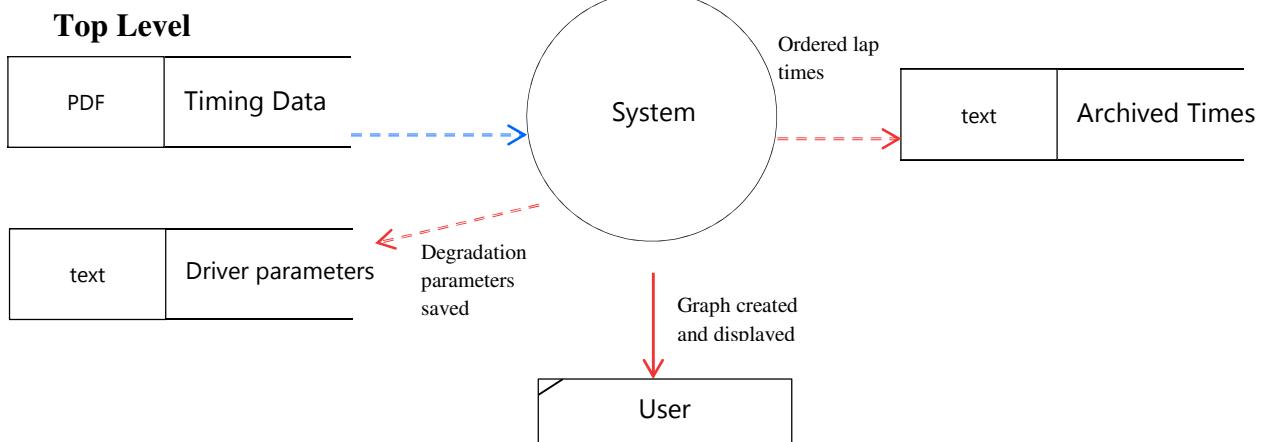


The current system is clearly very complex. It involves a lot of user input and there are several different tools that are required to produce the end result. No doubt these contribute to excellent accuracy, and help separate the work of the different teams, but it does mean that data has to be continually passed through files, which are often in different formats, between the systems.

The amount of user interaction with the system inevitably means that it can be quite slow to optimise, or simply to set up. I think that I can automate these processes so that the user can simply set the system running and then see an optimised output. It should, however, be possible for the user to affect the system and see an interactive response.

### Proposed System

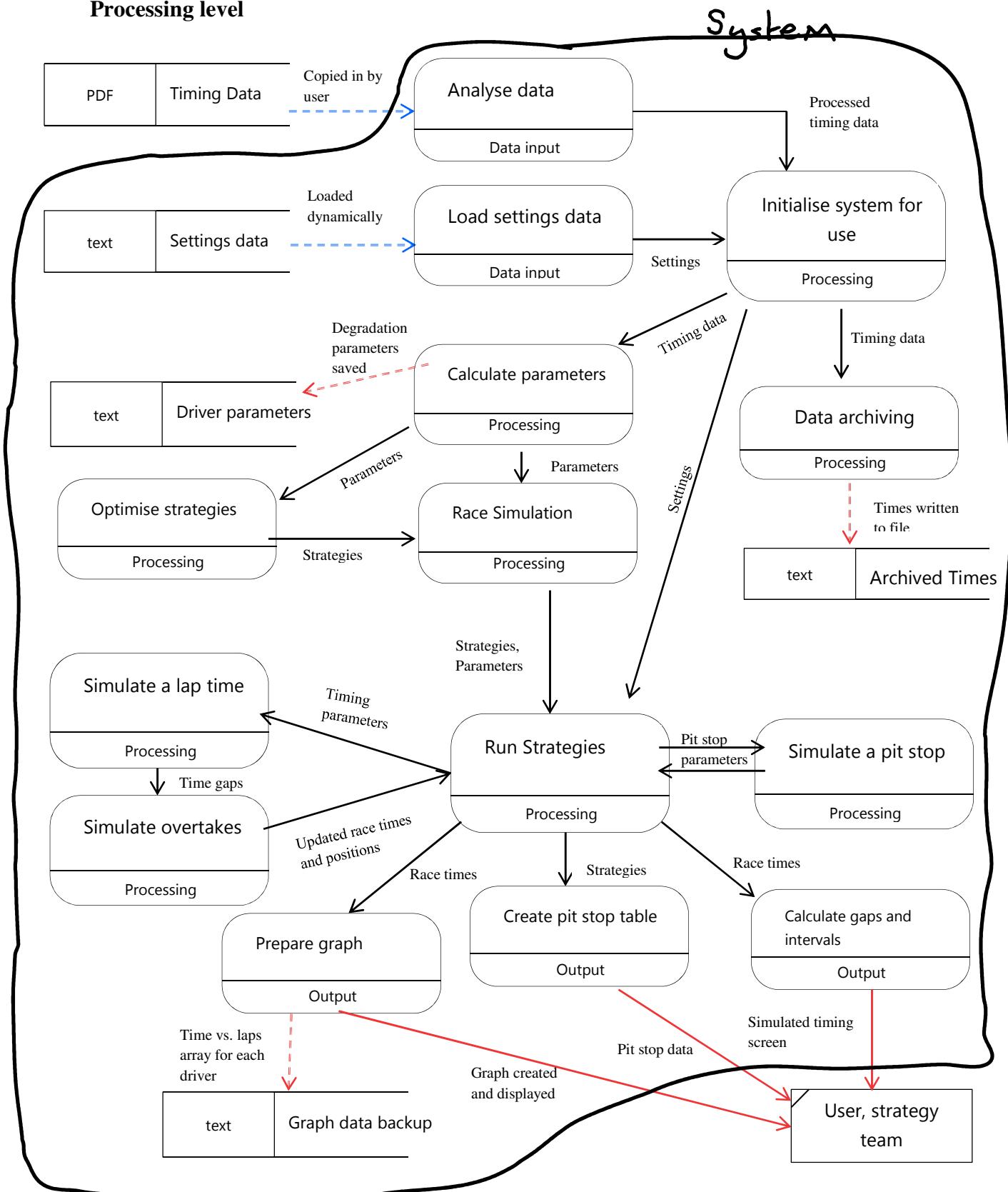
#### Top Level



My proposed system limits the amount of interaction with the user, and also clearly reduces the number of different tools that are required for the system to function effectively.

Hopefully, by combining the systems, the work done by each one can be assisted by routines and classes from the other system – this will help to keep data formats consistent, and reduce the time required to communicate changes to any parameters, and update these changes at the source.

I have also added a third section to the system. This will archive the timing data so that it can be retrieved whenever necessary in a much easier to use format than PDF.

**Processing level**

At the more detailed level, the system that I am proposing becomes very complex. However, it is largely sequential, with the exception of certain processes which serve as a data centre, distributing the same data to different locations as required.

The user will input data from the PDFs, which is immediately processed by the system. Settings data is also loaded at this stage – this will in fact come from three files – the Track file, which contains data about all of the tracks that will be raced in a season, the Drivers file, which contains the name of all of the drivers that race in a season, and the Settings file, which contains default values for any parameters that require a user input. These include the overtaking pace and speed delta, and pit lane loss. These could be hard-coded, but the user may want to make changes to the defaults for infrequently changed parameters so that they do not need to be entered every time, or altered in the code and require a new release.

There will then be a routine to load and process the timing data and settings; this will serve as the base for any data that needs to be passed out later on. The archived data can be stored from here too.

Once the timing data is in a usable form, I will analyse it to work out the required parameters, according to calculations that will be derived later on. These parameters will be saved as a backup in case the system fails at any stage. It will also allow the team to compare them in a consistent format with other races to check consistency and also notice trends in different cars' performances.

From here, the strategies can be optimised. This routine picks the best lap to stop on, and the best tyres to use at any given time. This is returned to the race simulation routine, which applies the known parameters, the strategies, and the settings (including the number of laps and overtaking parameters) to each car. This routine will delegate to three subroutines – the lap time simulation, overtakes simulation, and the pit stop simulation. Each applies their relevant parameters to the race times to advance the race by one lap.

Once the race data has been collected, it can be converted into data suitable for output. The strategy graph, pit stop list, and driver list will be calculated, and then passed to the relevant 'draw' routines to be presented on-screen to the user. Any changes that are then made will cause the system to repeat, re-running the simulation to collect the results and outputting to the user.

The data used to produce the graph will also be exported as a text file so that it can be analysed easily in other programs, such as excel. This is a useful tool for the user to have.

## Volumetrics

### Anaylsis

There is clearly a lot of data to be stored within the program. If possible, the data that is used for processing will be stored in main memory, while the records used for input and output will be saved as files on the hard disk.

My end user has stated that the laptops used on the pit wall have 8GB of RAM, and I will have to make sure that my system does not take up a significant amount of this. I have to allow for the graphics and internal properties that are created along with this type of system too – this includes the design of the windows and the additional memory required as a temporary storage location when calculations are taking place.

The system can be split conveniently into several simple sections. This will form the basis of the classes that are implemented in the object oriented design later on, but for now it is a convenient way of assessing the amount of data that will need to be stored.

For the purposes of the ease of calculation, many local variables have been omitted from this analysis. This is because they will have to be added as required in the programming. Based on the fact that an `int` takes up only four bytes, and the same for the `float` data type, the volume occupied by these values will not be significant.

### Data types<sup>2</sup>

Type	Volume (bytes)
Char	2
String	Length*2
Int	4
Float	4
Double	8
Long	8
Bool	1

### Volumetrics

#### **Stint**

A stint is a collection of laps. Each driver will have a list of stints from each session.

Variable	Type	Storage
Session	Int	4
Tyre	Int	4
Length	Int	4
Start lap	Int	4
Average lap	Float	4
Fastest lap	Float	4

<sup>2</sup> Microsoft developer network

Average degradation	Float	4
Lap Times	List<Float>	20
<b>Total</b>		<b>48</b>

## Race

The race will consist of a list of stints for each driver, as well as parameters regarding the track used.

Variable	Type	Storage
Track Index	Int	4
Fuel Consumption	Float	4
Laps	Int	4
Stints	List<Stint>[22]	3168
<b>Total</b>		<b>3180</b>

## Drivers

There will be 22 drivers

Variable	Type	Storage
Pace	Float	4
Tyre 'deg'	Float	4
Tyre delta	Float	4
Fuel Effect	Float	4
Fuel Consumption	Float	4
Top Speed	Float	4
Position	Int	4
Lap Time	Float	4
Gap	Float	4
Interval	Float	4
Tyre cliff	Int	4
Driver number	Int	4
Driver name	Char[20]	40
Line colour	Int	4
Stints	List<Stint>[5]	1200
<b>Total</b>		<b>1292</b>

## Tracks

There could be up to 23 tracks

Variable	Type	Storage
Name	Char[20]	40
Fuel consumption	Float	4
Laps	Float	4
<b>Total</b>		<b>48</b>

## Overtakes

There could be up to 600 overtakes in a race

Variable	Type	Storage
Speed Delta	Float	4
Time Delta	Float	4
Time Loss	Float	4
Back Marker	Bool	1
Position Gained	Int	4
Driver numbers	Int[2]	8
<b>Total</b>		<b>25</b>

### Summary

Collection	Quantity	Size	Storage
Race	1	3180	3180
Driver	22	1292	28424
Track	23	48	1104
Overtakes	600	25	15000
<b>Total</b>			<b>47708</b>

In total the system is likely to take up some 47kb of RAM. However, this does not include the amount of data that must be included when creating the graphics. Further research will need to be done in this area to ensure that the computer can cope with the speed of processing required to implement this effectively.

The amount of space required is not significant in terms of the 8Gb of RAM that the computers have, so there should be no real performance issues.

## Feasibility of potential solutions

### Options

Firstly, a hand calculation could solve the problem, and it would be very efficient for one single strategy. However, to simulate thousands of strategies, this needs to be automated.

I could use a spreadsheet package, such as Microsoft Excel, and this has advantages in that the mathematics can be handled easily. It is also customisable through the use of macros, and can be automated to take control of this.

A database tool is also customisable – for example Microsoft Access can be supplemented by SQL. It would be very easy to make the data output very attractive, and it is also an excellent way of storing data for later reference. I could make links between the different simulated strategies and help to build up a database of statistics, which could later be referenced to speed up processing.

Finally, I could design a bespoke solution in a programming language. This would allow all of the data to be processed efficiently and in a relatively small package, making it economical on space. However, it will be complex to produce some of the graphics, and data input can be very time consuming unless this is automated too. It

becomes difficult to make the program look attractive, but I am told that this is not a significant concern.

## Justification of chosen solution

### Complexity

Due to the complexity of the solution that is required, it will require a programmed solution. There are no existing general purpose software packages that can be modified to run the required simulations because of the requirement to automate the analysis of data in the input.

### Structure

The project lends itself well to a 3<sup>rd</sup> generation programming language because of the iterative nature of the calculations that need to be undertaken. The instructions will also be imperative in nature, which means that a fourth generation language is not ideally suited to the program.

### Rarity

There are only 11 Formula 1 teams in the world, and each uses its own system to try and gain an advantage over the other teams. This means that each system will be completely unique, and therefore requires that the solution be bespoke. This also allows the implementation of some hard-coded features, and will reduce processing and compilation time as a result.

### Use

Use of the system will be frequent but the input parameters will not change significantly through the course of a year. There will be minimal changes required from year to year but these can be incorporated in text files external to the main project code. As a result, the program should be compiled as an executable file which can then be installed on all of the operating systems required. The operating systems used will be windows 7 only, and the program will only be run on the one, standardised type of PC used at Red Bull Racing. This makes the compilation quite simple.

### Language

My end user has specified that he would like to incorporate the project with some of his other projects that are used for strategy simulations, such as the timing screen, gps data, and weather information. In order for this to be a possibility, it must be written in C#, therefore this is the language that I will be using for the project.

### Solution Specification

- The solution will be automated in program code
- I will use C# due to the requirement for the user to be able to integrate it at a later date

- The solution will be compiled for use, although the user will be sent the source code.
- The solution will be specific to Formula 1
- The solution will be operated through forms to allow readable data output.

## Prototype

### Design of Prototype

To give the end user an idea of how the system will look and work, I produced a prototype system that demonstrates some of the functionality. I demonstrated this to him to get feedback on how it operated and how it was designed to look.

The prototype was designed to prove the point that the system would be able to take input data from the PDFs, process it, and produce accurate values for the pace parameters. This comprises the most complex third of the program, from the data input stage to the point where strategies can be optimised and simulated.

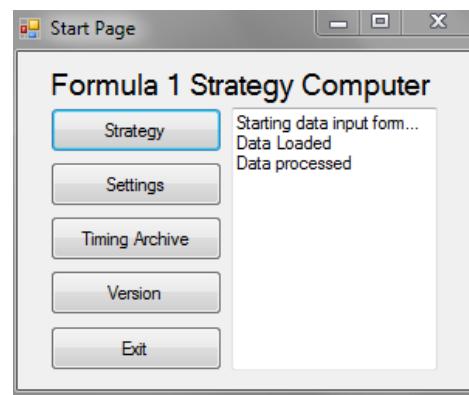
The prototype was initially developed in a CLI to open a file and read the data in, then to gradually break down the data until it had achieved raw lap times. Once the code was working in the CLI, I moved it into a form which allows the user to open multiple different files. Once there is data loaded for more than one session, the system can begin to create the parameters.

The prototype was designed using an OO structure, with the intention of testing out the potential designs of classes. This has proved successful as many of the classes that were created can be copied into the final solution.

It was programmed in a short time-frame and therefore often experiences errors. However, it appears to be a fairly robust system that functions very effectively and very quickly, which is promising when it comes to the development of the final solution.

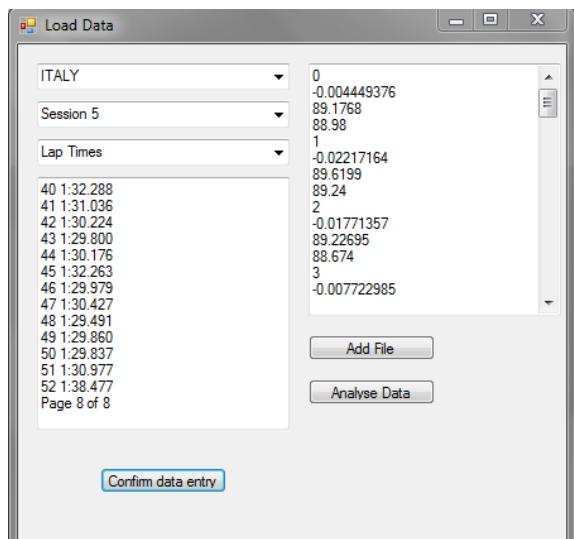
The start window, for the prototype at least, demonstrates the functions that the system will have. It will have the strategy calculations function, an ability to edit the settings used, the timings archive functionality, and a standard version window and exit function.

On the right, and purely for debugging purposes, a text box is used which can be written to when events occur. In this case, the sequence of events shows that the form 'Load Data' has been started and initialised. The data is then loaded into main memory from the PDF that is selected, and the data has been successfully processed into reasonable data for outputting.



On the next page is the ‘Load Data’ window itself. The PDF that is desired can be selected using the combo boxes, which are temporary solutions. They will be formatted more effectively in the final solution. Once the PDF is opened, the data is copied directly into the text box below, and the ‘confirm data entry’ button can be pressed.

This sets off a series of calculations. The timing data is interpreted and sorted into particular drivers and particular stints, which makes it easy to analyse the data.



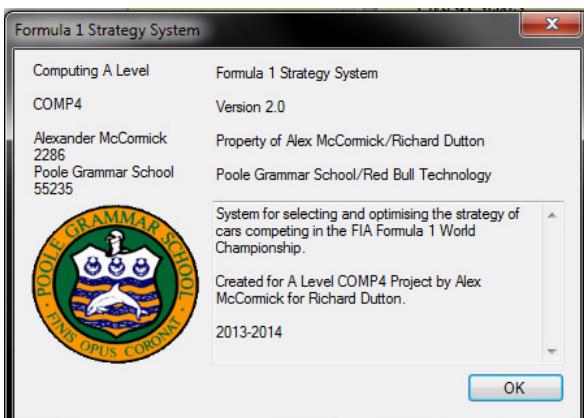
To prove that it has loaded correctly and has been successfully processed, the right-side text box shows a stint for every driver during the race. It shows the average time change per lap, which can be used for assessing tyre degradation and fuel effects, the average lap in the stint, which will be used for standard time analysis, and the fastest lap in the stint, used to get a measure of the car’s outright pace.

These three values are created by the functions shown on the next page, and form the basis of all of the calculations that must take place to generate the vehicle’s pace parameters.

From this page, the user will have the option to add extra files to the stints that have already been added. This means that they can collect all of the data required for the system to be able to model a race.

The analyse data button will then cause the system to generate the parameters. To prove that the system works, I set it to calculate the tyre delta, which is the difference between the driver’s fastest lap in the first practice session and the second practice session. The results, of which a selection are shown, are all between 0 and -1.3s, which is roughly in the range that is to be expected. While it may not exactly reflect the real delta, it gives a very good estimate of what should be expected.

VETTEL	-1.299995
WEBBER	-1.027
ALONSO	-0.2699966
MASSA	-0.9300003
BUTTON	-0.5029984
PEREZ	-0.3800049
RAIKKONEN	-0.8250046
GROSJEAN	-1.178993
ROSBERG	-0.3369904
HAMILTON	-0.2249908
HULKEMBERG	-0.7639923
GUTTIEREZ	-0.1210022



I am satisfied given these results that the system works as intended. I then showed it to my end user to receive his comments.

A version window has also been created to prove the ability of the system to function in different windows, and to also keep track of different system versions when it is distributed and tested.

### Functions code:

```

public List<float> lapTimes;

#region Statistics
public float fastestLap()
{
    float fastestLap = 200;
    foreach (float l in lapTimes)
    {
        if (l < fastestLap)
            fastestLap = l;
    }
    return fastestLap;
}

public float averageLap()
{
    float sum = 0;
    int totalled = 0;
    int count = 0;

    foreach (float l in lapTimes)
    {
        if (count++ != 0)
        {
            sum += l;
            ++totalled;
        }
    }

    return (sum / totalled);
}

public float averageDegradation()
{
    if (lapTimes.Count >= 3)
    {
        return (lapTimes[lapTimes.Count - 1] - lapTimes[1]) /
lapTimes[lapTimes.Count - 2];
    }
    else
    {
        return 0;
    }
}
#endregion

```

### Limitations

The prototype is clearly limited in its overall functionality because of the time that was available to program it. It cannot complete all of the functions of the finished product because of the restrictions on time.

The code is not written in the most efficient way possible because it is designed to be the basis for a much larger system. As a result there are some redundant parts that will be included once the system is expanded. The object oriented design is also not optimised as this design gave me a chance to experiment a little with how the code should look. There will certainly be some changes when the final version is developed.

The graphs that I would like to program have not yet been achieved. This is because of time constraints. There has not been time to demonstrate this section but I believe that a later prototype could demonstrate it before the final version if necessary.

Finally, there is no validation in the system, and certain parts have been hard-coded. These will be alterable in the final version, so that the user is more easily able to change settings. The validation will also be implemented but not to a full extent; after the interview it was established that extensive validation is not worthwhile pursuing.

### End User Feedback

On the whole, Richard was pleased with the way that the system loaded and processed the data so far. However, he had some useful suggestions for improving it, as well as some general pointers on the design and the code. These are detailed below.

### **Functionality**

The system currently requires data to be copied into the text box, where it is then processed. Richard suggested that operations involving the clipboard are quite easy to achieve, which would save a couple of operations. I was unaware of this functionality and will therefore try to implement it in future versions.

When the system loads the data, Richard suggested that it is processed straight from main memory, rather than being written to a text file and then read back in. This is a legacy of the modular design, which can be easily changed. File IO is apparently very slow in comparison to accessing main memory, so it would result in performance improvements if this approach was taken.

The system also requires the user to know the path to take through the system. If it was designed to be more sequential – i.e. the sessions from practice one through to qualifying had to be entered in a defined order before the user was allowed to progress to the race simulations menu, it would be much easier to use. I will take this on board and try to design the system to take this into account, while also allowing data to be added at a later date.

### **Code**

Richard's first comment on the code was that my type names, e.g. 'aDriver', did not require the 'a' in front of the name. The compiler can differentiate between type names and variable names using syntax and it is generally accepted that the programmer will also be able to understand the difference. I will therefore drop the additional identifier.

Similarly, the use of 'frm' ahead of the window names and text boxes is unnecessary, as programming languages now are trying to move away from the requirement to explicitly define variables of a particular type. The 'var' statement in C allows the program to dynamically define the variable, which is more efficient and more useful than explicitly defining it. Therefore it is sensible to try and move the type out of the variable name.

There is inconsistency in the definition of variables too – the drivers are stored as an array and the tracks are stored as a list. Richard stated that arrays were easier to use as it was possible to hone straight in to the required index. I will program the track list in this way too.

I asked about the way that the array of drivers is indexed too. The driver numbers start at 1, and due to motorsport tradition there is no number 13. To zero-index the array makes quite a complicated conversion, but Richard said that this was necessary; it is how the current system works. I will make sure that the changes are made to index this array at a zero base and increasing sequentially up to index 21 for the 22<sup>nd</sup> driver. The prototype indexed the drivers at their race number, with the 0 and 13 indices not used.

There are regulations changes coming next year in the sport, and potentially in future years too, which may affect the way that strategies are processed. Richard therefore suggested a complex language structure known as an interface. This allows several different items which have similar effects to be linked.

For example, I can use an interface to link the model with a different strategy class, should this be required when changes are made to the strategy. If the routine called 'getBestStrategy' has been programmed in both the current strategy class and the new strategy class, and it is defined in the interface, I will be able to call it from the program without concern over which class has been implemented. In theory, the implemented class can simply be swapped over in future years, containing the changes that would have to be made into just one class.

## Visuals

The prototype will utilise a lot of windows in its presentation. This is not effective when it comes to displaying it on large screens, as the small windows will all need to be scaled, which takes time to re-arrange. Richard suggested that I designed the system to match his system so that it would scale well on the larger displays, using tabs and panels that can be switched between quickly within the same window.

There were some minor comments about the way that the combo-boxes were defined. The race names in the current system refer (in the most part) to the town or city near the track, or the track name itself. Hence, the British grand prix is ‘Silverstone’ and the Italian grand prix is ‘Monza’. I have defined the race names based on the country, which I believe should be changed for consistency.

The session names, currently defined as Session 1, Session 2, etc. should also be defined properly as First Practice Session through to Qualifying. This is a small issue as it is hard coded at the moment, so I will make the changes as required.

### Further Research

Based on Richard’s comments, there were several areas that I needed to research to make sure that I was confident heading into the final, agreed requirements that I could program everything that he had requested.

I also took this opportunity to look into things that had the potential to affect the system in the future, in order to make it future-proof, and to familiarise myself with some of the concepts he suggested using.

As such, I researched

- Interfaces – used for linking very different objects with similar properties, and allowing for flexibility in the construction of the system.
- Clipboard operations – so that the data that is copied from the PDF can be automatically inserted into the system.
- Tab controls – the code and programming practice required to make these work will be vital in ensuring that the system functions smoothly.
- Future regulations changes – there will be changes to the type of tyres available in each session next year, which will mean that the data analysis routines will become more complex.

## Final user requirements

The final user requirements complete an agreed list of the requirements and limitations of the solution. This is based on the initial requirements, and adapted to take into account the research and work that has been done since then.

Items highlighted with red text have been added or significantly modified as a result of the research that has been undertaken. This does not include the clarification of methods for calculating the system, because these are part of the specification and not the requirements.

This list is complete; anything that is not mentioned here as a requirement has been agreed as an acceptable limitation of the solution.

### 1. Format

- a. The solution will be automated in program code
  - i. It will be programmed in C#
  - ii. It will be a forms application
- b. The solution will be bespoke
  - i. It will be specific to Formula 1
  - ii. It will be designed to work specifically for my end user's requirements
- c. The solution will use object oriented programming
- d. The user will be able to work interactively with the system.

### 2. Data Input

- a. The system will allow the user to select an FIA PDF file that is saved locally based on the selection of a race weekend, session details, and data type through combo boxes.
- b. The system will then open the selected timing data.
- c. The user can copy data in the PDF.
- d. The system will read the copied text from the clipboard
  - i. To file, to save data that is already loaded.
  - ii. To a routine to process the data
- e. The system will read settings data from file to establish
  - i. Default values for parameters
  - ii. Infrequently changed parameters
  - iii. Track names and data
  - iv. Driver names
- f. The system will take user inputs to fine-tune or define parameters used for the simulation.
- g. The user will be able to right-click on certain objects to insert a pit stop

### 3. Lap Time archiving

- a. The system will save data from the input sessions to CSV files, which can be accessed at any time to view past lap time data.

- b. This data will be processed to remove the unnecessary data found in the PDF files.
  - c. The CSV files should be in a format that can be read by Microsoft Excel, for analysis.
  - d. The output of data will be tabulated for ease of viewing.
  - e. Data should not be able to be corrupted during viewing.
4. Pace calculations
- a. The system will analyse the timing data to define parameters about a driver's pace. These include:
    - i. Top Speed
    - ii. Base lap pace
    - iii. Option tyre delta
    - iv. Tyre degradation (prime and option)
    - v. Fuel effect
  - b. The system will save the pace parameters data in race and session specific files, which can be used for data analysis.
  - c. The system will set to default values any data which cannot be loaded from the timing data.
  - d. The system will allow the user to edit these values in case they have more accurate information.
5. Strategy Optimisation
- a. A strategy is defined as a set of stints of a defined length, separated by pit stops.
  - b. The system will choose the optimum number of stops, the tyre choice, and the laps on which to pit, based on:
    - i. Automated algorithms for calculating the optimum
    - ii. The number of laps in a race
    - iii. The pace parameters
    - iv. The pit stop loss
    - v. The regulations, which are taken into account in the code
    - vi. Other cars will not be taken into account here.
  - c. The system will set this strategy as the default for this driver
  - d. The user will be able to change the strategy that has been selected as the default.
  - e. Once a race has been simulated, the strategy for one driver can be optimised taking into account other drivers
    - i. This will use iterative methods.
  - f. The user will also be able to manually adjust strategies
6. Race simulation
- a. The system will put all of the drivers' strategies together in a race.
  - b. The system will run a lap-by lap simulation of the race based on the defined pace parameters for each car.
    - i. If pit stops or traffic are encountered, the event will be logged.

- ii. At the end of the lap, all events will be processed, causing a time loss for the car as well as (potentially) other effects.
  - c. The simulation will include:
    - i. Pit stops
    - ii. Overtaking (and the effects of ‘traffic’)
    - iii. Tyre degradation and delta effects
    - iv. Passing lapped cars
    - v. The tyre ‘cliff’
  - d. The simulation will not include
    - i. The effects of weather
    - ii. Safety cars
    - iii. Track evolution through a race
    - iv. The effect of DRS
  - e. The results of the race simulation will be displayed graphically, with the cumulative time (normalised on any selected driver’s average lap time) plotted against number of laps for each driver.
  - f. Overtaking will be simulated using a probabilistic approach
  - g. Each driver will have a different coloured trace on the graph.
  - h. The graph will have adjustable axes to allow the user to change the detail of the view.
  - i. The strategy can be changed from the same window as the graph to see the effects visually.
7. Settings
- a. Settings need to be alterable but do not need to be altered within the running program.
  - b. The units should be in line with FIA standards
    - i. Speed:  $\text{kmh}^{-1}$
    - ii. Time: s
    - iii. Fuel weight: kg
    - iv. Fuel consumption:  $\text{kg.lap}^{-1}$
  - c. Time is increasing in the positive direction
8. Additional processing requirements
- a. The system will implement interfaces to link together different classes
  - b. The system will be able to adapt to new regulation changes in future years.
  - c. The solution will run on Windows 7 PCs
  - d. The solution will take up less than the 8GB of RAM available on the PCs used by the race team.
  - e. There is no need for encryption or data security.
9. Additional output requirements
- a. The system will be displayed in one complete window.
    - i. This window will scale with the size of the screen
    - ii. The window will contain tab controls to viewing different elements
  - b. The system will use simple graphics to prevent the display from becoming jerky.

- c. Additional windows will be used for the timing archives and version information.

## Design

### Overall system design

#### Principles

The system will be split predominantly into three distinct sections. These sessions will be linked together through the main program class, which will store data from all of them.

It will be an object oriented design, due to the large amounts and complex collections of data that need to be stored. There are many types of user interactions that need to take place and all of these need to be handled properly.

As researched earlier, while an MVVM structure is not appropriate for the design overall, I can certainly incorporate elements of it. For example, I am very keen to separate the user interactions from the data, as this means that when I make changes to the data and the way that it is held, for example allowing the 2014 regulations changes to be implemented, it is all located in one place in the code and does not affect the sections that the user interfaces with.

Hence, the design will centre around three different types of class. There will be the forms and user interactions, which will handle predominantly the visualisation and the events from forms and buttons. Then, there will be data and processing classes which define how to analyse the data and what data is stored.

All of these will be linked together by central classes which link the user events to the right combination of data processing. These classes will contain some elements of both other types of class but will largely serve as collections of instructions to be executed when required.

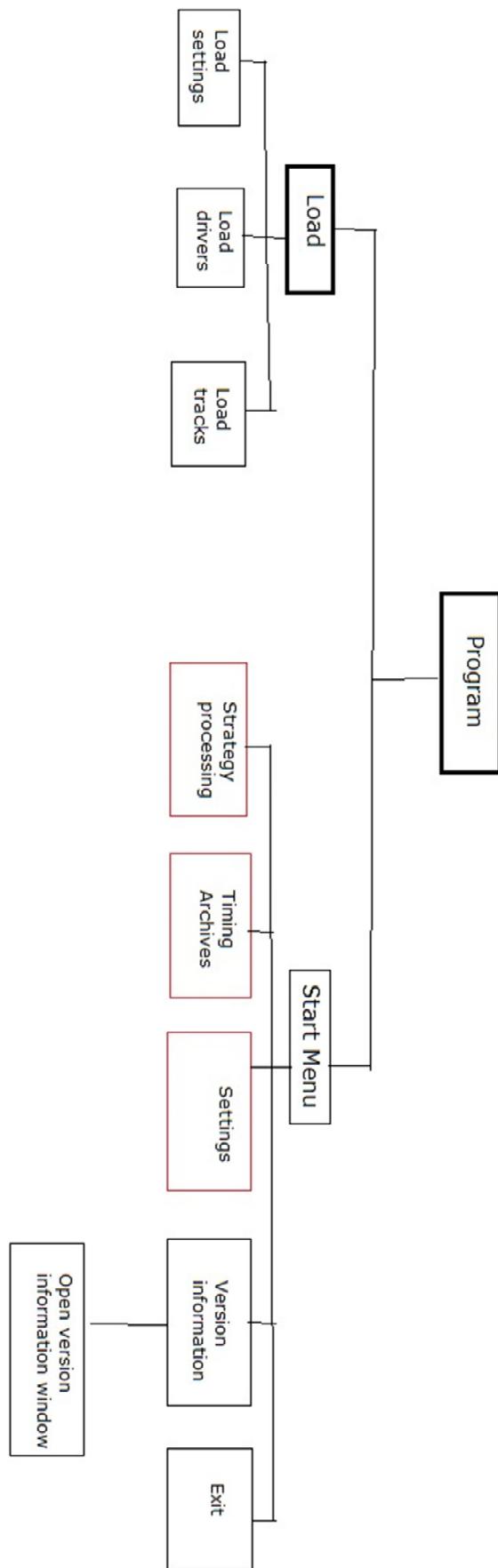
The system will also be split into three main areas. This will make it easier to program and to demonstrate through a structure chart. It will be easier to distinguish between the sections if they are in separate projects.

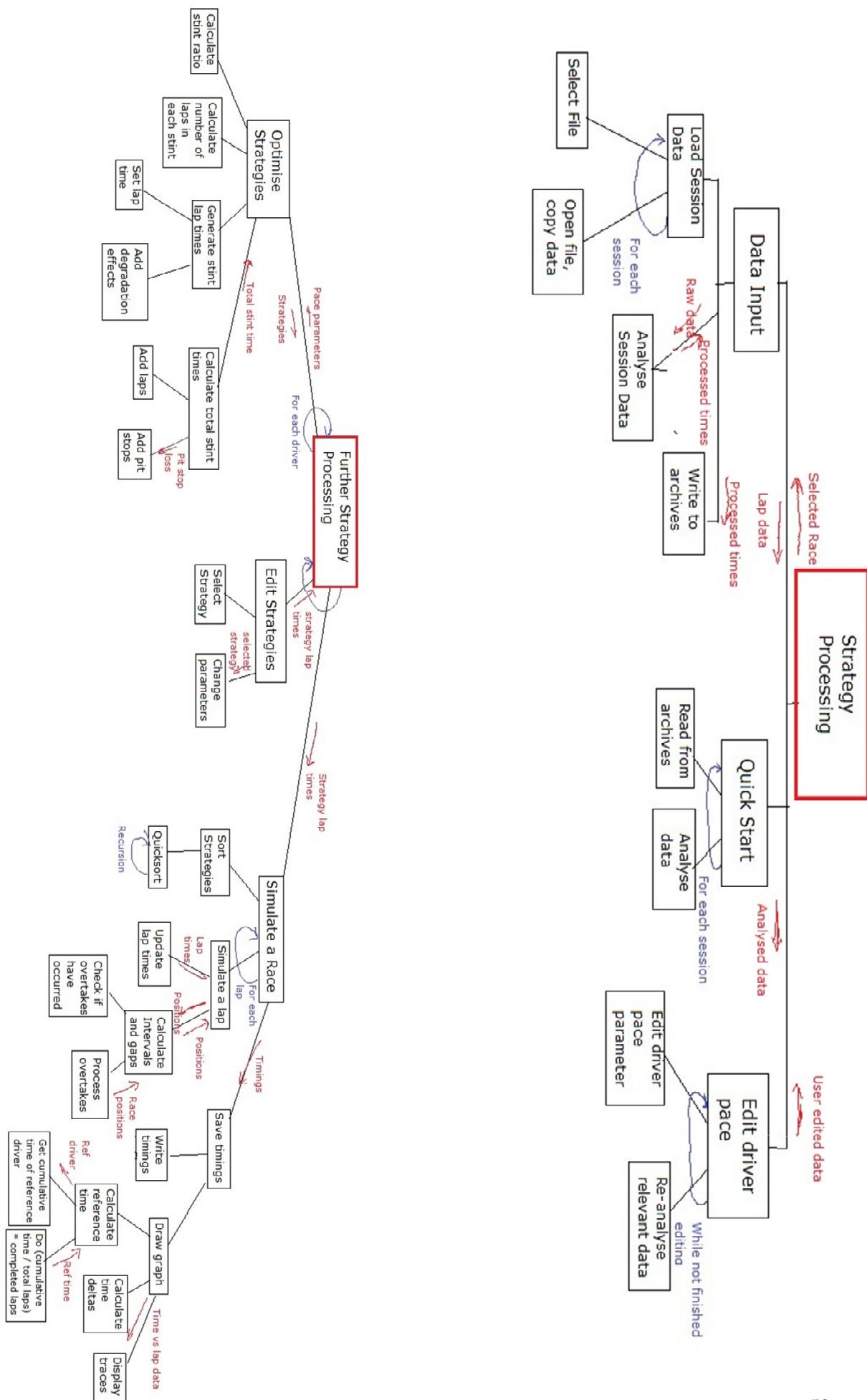
These projects will be the strategy computing, the data archiving, and the settings sections. These are the three main options from the start window. From here the program will diverge as shown in the structure charts.

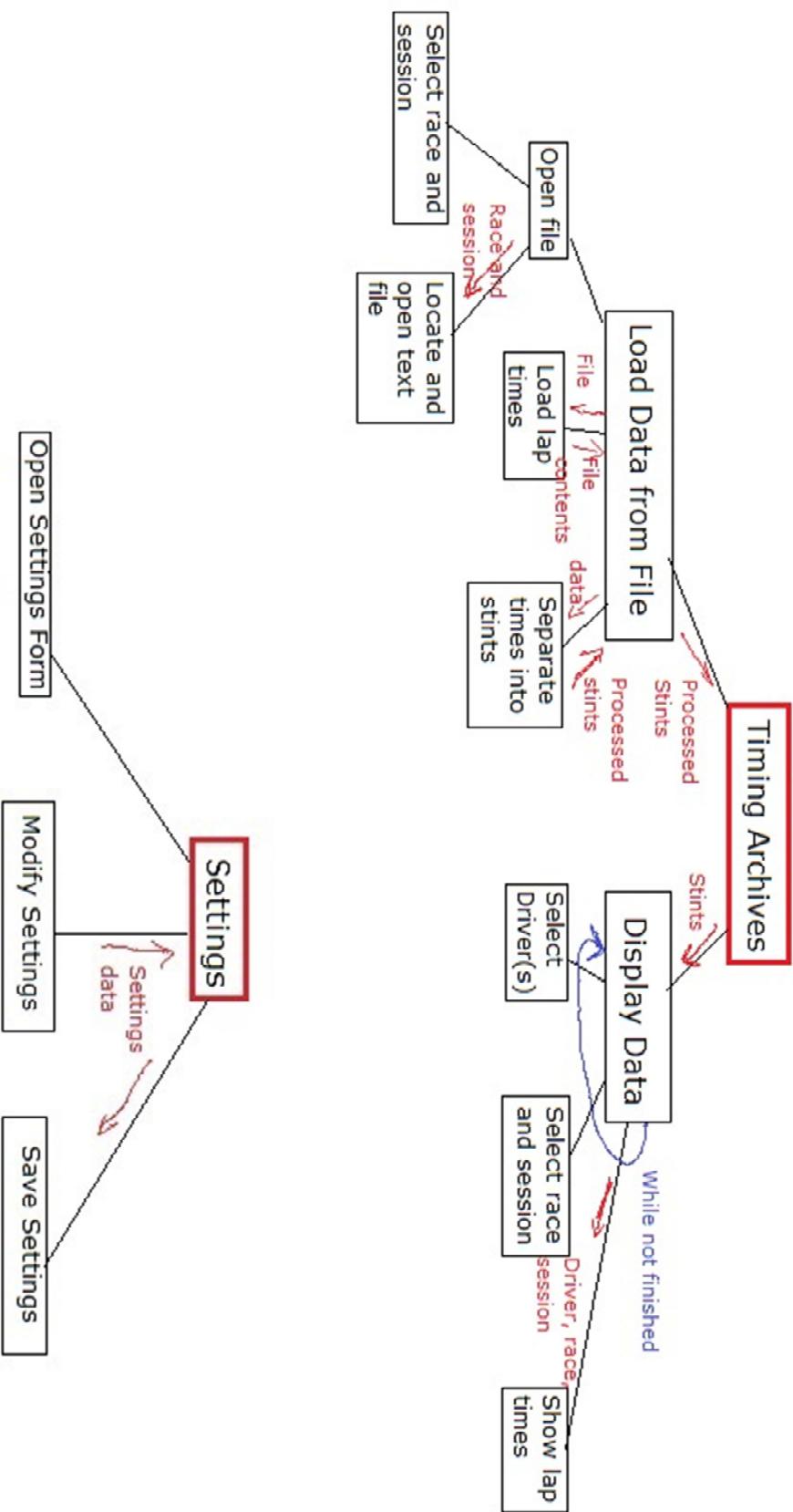
#### Structure Chart

The program is very broad in its processing, so with the aim of fitting the structure charts on to A4 paper, I have broken them down into the separate projects.

The first chart shows the system startup – the settings are loaded and then the system runs the start menu. From here, the user has a series of options.







## Class Inheritance

This section gives an overview of all of the classes, structures, and enums used in the system. It details the inheritance of the classes, and the implementation of interfaces throughout the system. This will be used as a guide to the development of the system. Separating the classes in this way aids readability and will make maintenance easier.

### Public Structures and Enums

- Enum TyreType – Options for the type of tyre that the car could be on – either ‘Prime’ or ‘Option’.
- Struct racePosition – Contains data about the driver found at a particular position and lap in a race.
- Enum DragState – Options for the current state of a window during a drag/drop operation.
- Enum Location – Contains the four window locations, Top, Left, Bottom, Right.
- Enum LineDirection – Contains horizontal and vertical options.
- Enum FillStyles – Options for the way a MyPanel fills a screen.
- Enum AutosizeTypes – Options for the way a MyPanel will stretch to fill available space
- Enum DockTypes – Options for the dock location of a MyPanel within a window.
- Struct axisParameters – Contains data representing the scale and position of an axis on a panel.
- Enum NormalisationType – Options for the way the graph is normalised, either on every lap or on an average lap.
- Struct LapDataPoint – Contains data about a point which is to be plotted on a strategy graph.
- Enum ShowTracesState – Options for how many DriverShowCheckboxes to show when updated.

## Classes

### Data Model

- Namespace ‘Files’ – contains classes for loading and processing data from PDF files.
  - Interface IData – specifies methods required to load data from files
    - Inherits IFile<string>
  - Class TimingData – base class for classes that load data from PDF files
  - Class LapData – class that loads data from a lap-time format PDF file
    - Implements IData
    - Inherits TimingData
  - Class SpeedData – class that loads data from a speed format PDF file
    - Implements IData
    - Inherits TimingData

- Class DataController – class providing methods for managing the loading of data from a PDF file.
- Class Data – contains static fields for constants and frequently-referenced variables in the program.
- Class Functions – contains static methods to operate on data, including validation and sorting.
- Class Settings – contains data representing the settings used throughout the program.
- Class Driver – represents one of the drivers used in the simulation. Contains fields about the driver's performance and methods for calculating lap times and strategies.
- Class Track – represents one of the tracks that are used in the season. Contains data about the track used in the simulations.
- Class Stint – represents a collection of lap times of a given length and tyre type. Contains methods for finding the length of the stint calculating the lap times based on the properties.
  - Implements ICloneable
- Class Strategy – represents a collection of stints and contains methods for running and optimising the strategy used. Also allows the strategy to be modified and re-processed dynamically.
  - Implements ICloneable
- Class Race – contains methods for simulating a race and displaying it on-screen.
- Class PitStop – contains data about the location and timing of a pit stop and provides methods for implementing the time loss associated with the stop.

## View

- Namespace ‘Forms’ – contains the forms used in the system
  - Class VersionWindow – displays information about the system version and copyrights.
    - Inherits Form
- Namespace ‘MyFlowLayout’ – contains classes used in displaying the window to the screen.
  - Class ContentTabControl – displays multiple MyPanels in a tabbed control displayed on the screen
    - Inherits MyPanel
  - Class DragDropController – provides methods for dynamically re-arranging panels on-screen when a drag-drop event occurs.
  - Class FlowLayoutEvents – contains events and corresponding delegates to allow re-layout of the panels on the form.
  - Interface IDockableControl – specifies methods and fields required for a control to be laid out using the flow layout functionality.
  - Class Line – base class for a vertical or horizontal line of defined length and start position.
  - Class LayoutLines – contains methods for finding the maximum size available for a MyPanel and therefore allowing the flow layout functionality.
  - Class MainForm – a custom form supporting flow layout functionality.

- Inherits Form
- Class MainFormIOController – contains methods for showing and hiding all panels included in the project. Supports the flow layout functionality.
- Class MyToolStripButton – represents the base class for a tool strip button with an associated index; contains events on click which contain the button index.
  - Inherits ToolStripButton
- Class MyContextMenu – a custom context menu strip attached to MyPanels to allow the editing of layout.
  - Inherits ContextMenuStrip
- Class MyPanel – a custom panel which can be laid out using the flow layout logic. Contains methods for modifying panel properties and specifies the panel design.
  - Implements IDockableControl
  - Inherits Panel
- Class MyToolbar – a custom toolbar which supports flow layout events and dynamic creation and removal of buttons. Also contains buttons required to support the flow layout logic.
  - Implements IDockableControl
  - Inherits ToolStripContainer
- Class RaceDropDown – an indexed tool strop button that displays the name of the track that the index refers to. Contains a custom event for alerting which button has been pressed.
  - Inherits MyToolStripButton
- Class WindowFlowPanel – a panel displayed on the MainForm which supports flow layout of its MyPanel controls. Contains methods for updating relevant data when panels are added, removed, or re-sized.
  - Inherits Panel
- Namespace ‘Panels’
  - Class AxesWindow – displays the axes parameters relating to the currently displayed graph.
    - Inherits MyPanel
  - Class DataInput – allows the user to enter data from PDF files before commencing parameter analysis.
    - Inherits MyPanel
  - Class DriverSelectPanel – displays a list of drivers that can be selected and unselected, changing the display of traces on the graph.
    - Inherits MyPanel
  - Class InfoPanel – displays basic information about the driver and track currently selected, and provides a running commentary on the program’s functions.
    - Inherits MyPanel
  - Class NewStartPanel – provides a quick interface with the data controllers to load and re-load data, display it, and then simulate a race.
    - Inherits MyPanel

- Class Pace Parameters – displays driver's pace parameters and allows the user to edit them, with a toolbar drop down associated to allow data loading, saving, and updating.
  - Inherits MyPanel
- Class SettingsPanel – displays settings data to the user and allows it to be modified, updated and reset. Contains events for updating other calculated data accordingly when settings are changed.
  - Inherits MyPanel
- Class StrategyGraph – displays a normalised graph of cumulative time for each driver on a common set of axes. Takes inputs from the data collected and the driver select panel. Contains methods for modification of strategies through right-click events on the graph.
  - Inherits MyPanel
- Class StrategyViewer – displays the selected driver's strategies and contains methods for modifying and updating the strategies. An associated toolbar drop-down provides save, load, update, and reset commands.
  - Inherits MyPanel
- Class TimingArchives – allows the user to recall timing data from previous races and seasons. Contains methods to show this data.
  - Inherits MyPanel
- Namespace 'UserControls'
  - Class NormalisedRadioButton – radio button with associated driver index that can be accessed on the click event.
    - Inherits RadioButton
  - Class ShowDriverCheckBox – check box with associated driver index accessible through the click event. Contains methods for re-populating the list of selected drivers on click and setting the checked value according to the list of selected drivers on load.
    - Inherits CheckBox
  - Class MyToolTip – base class for all tooltips in the form. Has customised delays and contains a method for setting the control of the tooltip.
    - Inherits ToolTip
  - Class ParameterTextBox – text box with methods for changing the colour based on input, validation on input, and events to set parameters when text is changed. Also contains a method for retrieving the default value for the box.
    - Inherits TextBox
  - Class TextChangedEventArgs – event arguments created when a parameter text box has its text changed. Contains a driver index, parameter index, and value, which can be updated by any event handlers
    - Inherits EventArgs
  - Class StintButtons – square layout of buttons displayed on a stint panel. Customised to be of a certain size and layout.
    - Inherits TableLayoutPanel

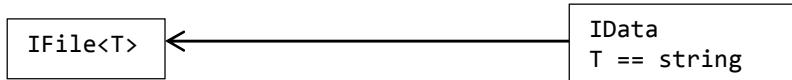
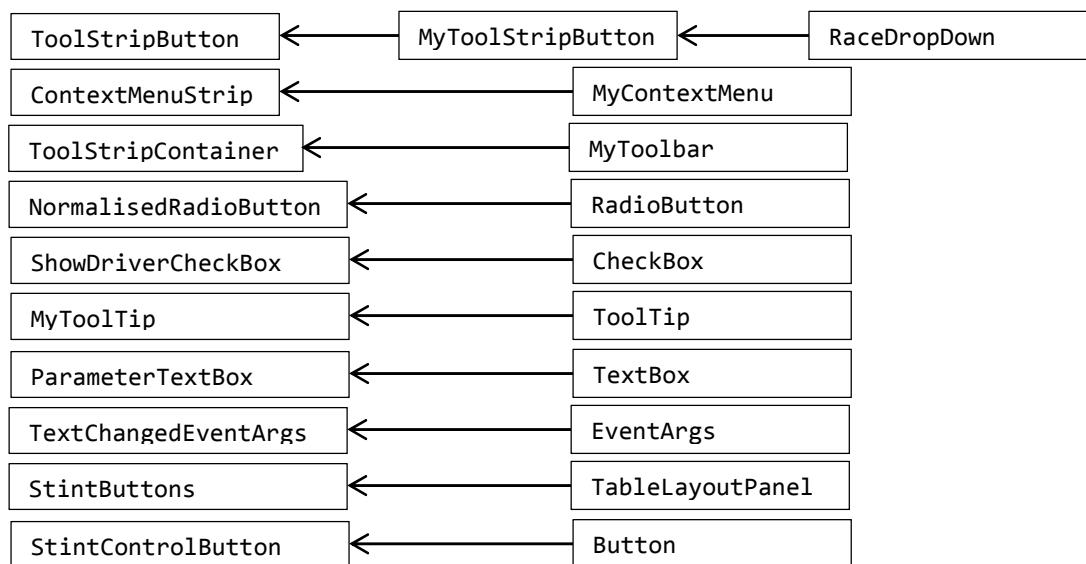
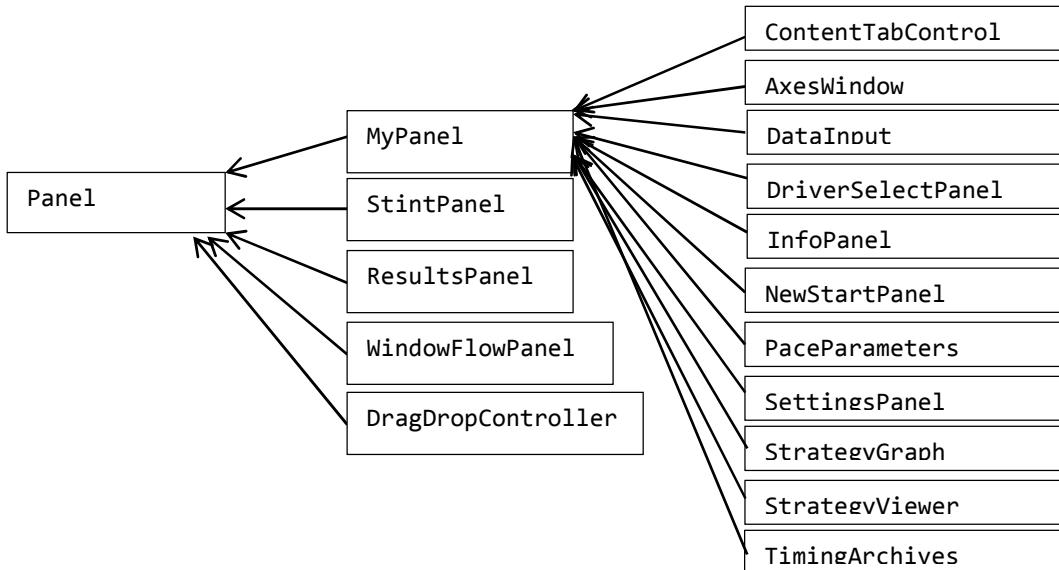
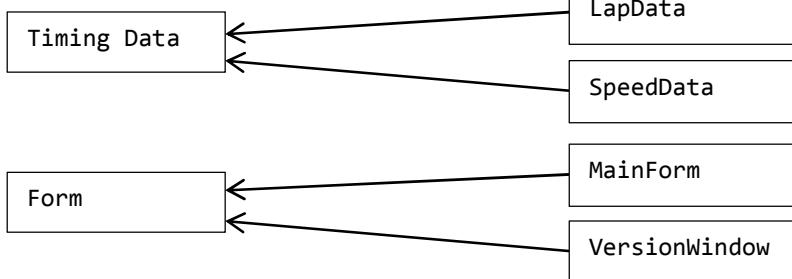
- Class StintControlButton – an indexed button which performs an operation on a stint within a strategy editor. Contains an event which triggers the strategy modifications.
  - Inherits Button
- Class StintPanel – panel designed to show data about a particular race stint from a selected driver. Contains methods to modify that stint when controls within the panel are modified.
  - Inherits Panel
- Class ResultsPanel – panel that shows summary data from a strategy, e.g. the fastest lap, average lap, and a total time.
  - Inherits Panel

### **ViewModel**

- Interface IFile<T> - specifies the methods and fields required for a data controller which has the ability to load from and write data to a file.
- Class MyEvents – General events class containing events and delegates required to link program functions together. It is the crux of the event-driven programming.
- Class PaceParameterData – contains methods required to read pace parameter data from CSV files, and presents this data in an easy format to the TimeAnalysis panel.
  - Implements IFile<float[,]>
- Class PanelControlEvents – contains events specific to hiding and showing panels, triggered when various commands are executed in the main form.
- Class StrategyViewerData – contains methods required to read and write strategy data from CSV files, and presents this data to the Strategy Viewer panel. Acts as a buffer so that update and reset functions are available to the user.
  - Implements IFile<Strategy[]>

### **General**

- Class Program – Contains methods and data required to run the application and organise the graphics objects across multiple forms.

Inheritance Diagram**Interfaces****Classes**

## Description of modular structure of system

This section gives a much more detailed explanation of the important classes in the system. These explanations describe the function of the classes, and some of the methods that are held in them. This is a useful reference for the purpose of classes, and can be read alongside the requirements and the structure chart to understand how data passes through the system and what processing is done at each stage.

### Views (forms)

#### **Start Page**

This is the front page of the system. It handles the user's requests for the type of action they want the system to complete, and also has a small window available for debugging.

It will be designed in the forms designer because this is the simplest way to control the properties and events. It means that it will be easier to make the start page look attractive later on.

#### **Data Input Form**

This form allows the user to select the type of file that they will be inputting. It then runs a few commands required to open this document, before the user can copy the data in and it handles the user's selection of what to do with the data. It can be accessed through the

The form will be dynamically generated because the controls will have to be populated with data at runtime. The code will be more readable if objects are created through the module.

#### **Time Analysis Form**

This form will also be dynamically generated as it will include a large array of text boxes for inputting and displaying parameters pertaining to the drivers' pace. It will be the basis for further race simulation and strategy modification requests. It can be accessed from the data input form or the start menu, or after a 'quick start' function.

The form will require a special class of text box, which will also be user generated. This code will extend the functionality of a text box using a partial class that inherits the TextBox class, adding events and formatting information.

Also in the form will be a custom event handler class. This supplements the programmer-defined events to pass data when a particular event is fired. This is necessary for the updating of parameters.

#### **Strategy view form**

This form will allow the user to select a particular driver and edit their strategy, allowing them to swap stints, change stint lengths, and add or remove stints. The form will display a total stint time so the user knows if their changes are being effective.

The form can also be viewed alongside the race graph form, so that the effect of any changes can be seen clearly.

The strategy form will also be generated dynamically due to the fact that the contents will depend on the selection of a driver from a list box; this means that the population of controls will have to be completed at runtime.

### Race Form

The race form will be an extension of the graph class. It will be dynamically generated based on the data held about the drivers and the race. The form will display axes and information, and display the race data as a graph of time gaps vs. laps. This will be a line graph.

There will be dynamic events on the lines and the axes to allow editing of the view and the strategy, because this allows the user to interact most effectively with the system to improve the strategy.

### Version Information

This is a simple form designed from a template to provide information about the version number, release number, and copyrights in the system.

### Settings

The settings form is very important for controlling how the system deals with errors – it provides default values and also allows the user to manage the folder structure that the system uses. This is essential at start up.

The form will have to handle a lot of data entry, so the most efficient way to display it will be with a tabbed control. This is complicated to generate within programmer-defined code, so it will be modified in the designer.

When the code is released, the settings form will have to be populated on the first start up, because some of the data that is held is essential to the system function.

The settings form will be accessible from the start menu and will provide a direct interface between the user and the data.

### Timing Archives

This will be a large dynamically generated form. It will contain controls for selecting a driver and a session, and then it will display the driver's lap times in that session as well as summary data about it.

The user will have the option to see a graph of this data, and, optionally, another driver's comparison, through the session. The archives will pull data straight from the driver class which contains it, with some intermediate processing for generating graphs and statistics.

This form will be accessible from the start menu and will handle all of the timing and archive data necessary to fulfil this section of the specification.

### Model (data)

#### **Driver**

The driver class will be a dedicated class containing data about the driver and their pace, and also including static methods for calculating the driver's pace parameters.

It will also include things like the colour that the driver's traces will appear on graphs, and their name and team for ease of searching and sorting the system. Each driver will have a single instance of a strategy.

#### **Stint**

A stint is a collection of laps, and this class will contain data about the laps. This will include the lap time, the type of tyre, and for identification purposes, the driver that completed the stint. Both a strategy and a session will be made up of a collection of stints.

The class will also contain simple methods for returning summary information about the stint. This will include the total length, the average degradation throughout the stint, and the fastest lap in the stint.

#### **Track**

The track class is an extension of a user defined structure, as it contains data that is read straight from a file.

#### **Strategy**

The strategy class contains the stints and lap times that make up the plan for a driver's race. The class also contains the routines that are necessary for optimising the strategy; this includes working out the correct number of laps to spend on a particular tyre, and calculating the total time required to complete the strategy.

The strategy class is the only class referenced when working out the strategy that should be implemented by a driver. This can easily be added to or modified in case the system needs to be modified to take into account rain or regulations changes.

#### **Race**

The race class brings together all the driver's strategies and provides methods for running the strategies and simulating the race. It will also create its own instance of a graph for displaying the strategies.

The race class will also interface with files to save data, both as a backup and so that it can be analysed using Microsoft Excel files. This will hopefully mean that other divisions within the company can use the output data effectively.

### ViewModel (data processing)

The system will include some classes that are dedicated to linking the user inputs with processing. These classes will manage all of the data handling and will transfer commands between the necessary classes.

### **Timing Data**

The data that is loaded from the PDF timing files will be in one of two forms. It will either be a lap time file, or a top speed file, and to convert these files into data that can subsequently be processed by the system is a challenge.

There will be several classes dedicated to managing the data input process, including multiple forms. There will also be an interface programmed so that the program treats the data independently of what type it is; both lap data and speed data will implement a data interface that allows archiving and processing with the same command. They will also both inherit a timing data base class for shared methods, such as opening the file, and converting values from strings to the relevant data type.

### **Pace Parameter Data**

This acts as a buffer for the data regarding drivers' pace. The class is populated with data from the drivers, and then presents this data in an easily accessible format to the pace parameter view panel.

This class also controls the writing of data to and the reading of data from files. It allows the data to be read from files and displayed to the user before they confirm that this is the data set they would like to use for optimising the strategies. When data is changed in the pace parameters panel, this class will validate it before it is accepted. The user can choose when to update the data held specific to the drivers, although this must be done before the strategy optimisation is started.

### **Strategy Viewer Data**

Similar to the pace parameter data class, this provides a buffer to the data held on the strategy panel. It presents the data in an accessible format, allows for reading from and writing to files, and validates data entry.

The data can be loaded from and written to the drivers when requested by the user, although in order for the changes to take effect, the user must update the data before a race simulation is started.

### **Static methods**

The functions class will contain methods such as a Quicksort. This will be used whenever the list of strategies needs to be sorted, for example when calculating the grid order at the start of a race.

Any other general functions that could be implemented elsewhere will be in this class. I considered the option of including these files in a separate project, and configuring it to compile as a dynamic link library (dll). However, after investigation into the subject this was deemed unnecessary for the purposes of this project.

## Data dictionary

The data dictionary is similar to the modular system description above. However, this section contains information about the properties of classes – therefore the exact data that will be held in these classes. The data dictionary can be referenced when checking what data is stored where, and how it flows through the program. Most of these properties can be traced through the structure charts too.

### Properties

#### Stint

Property	Description	Type	Example
Session	Value from 1-5 representing the session in which the laps were completed	Int	1
Lap Times	A list of the lap times completed during the stint.	List<Float>	96.551; 96.639; 96.754
Length	The number of laps completed during the stint	Int	12
Tyre	Integer representing the type of tyre used on the car for the stint. This is between 0 and 3 with 0 being prime, 1 being option, and 2 and 3 the wet-weather tyres.	Int	1
Start lap	The lap in the session that the stint began on.	Int	13

#### Driver

Pace	The base lap time that the driver can do. This is a figure in seconds with 3 decimal places.	Float	91.302
Tyre Degradation	The time loss per lap of age on a particular tyre. This will be stored for the prime and option.	Float	0.3; 0.8
Tyre delta	The time gap between the prime and option tyre, the option tyre being faster.	Float	-0.6
Fuel Effect	The time loss per kilogram of fuel carried around a lap.	Float	0.025
Fuel Consumption	The number of kilograms of fuel used per lap.	Float	4.2
Top Speed	The fastest attainable speed for a car, in kilometres per hour to one decimal place.	Float	309.7
Position	The rank of the driver in the current session. It is equal to the number of cars that are faster than him + 1	Int	2
Interval	The time gap between the driver and the car directly in front.	Float	0.805

Gap	The time gap between the driver and the leader	Float	11.200
Lap Time	The latest lap time of the driver in the current session, based on the above parameters.	Float	91.865
Strategy	An instance of the strategy class defining the driver's race	Strategy	
Sessions	The data held about the driver from practice sessions, in the form of a list of stints.	List<Stint>	

**Overtake**

Overtake speed delta	The speed delta required to make an overtake	Float	10.5
Overtake time delta	The pace delta required to make an overtake	Float	0.03
Overtake time loss	The time an overtaken driver will lose during the manoeuvre	Float	0.800
Backmarker	If the driver is trying to pass a backmarker	Bool	True
Backmarker time loss	The time lost by a backmarker when overtaken by a leading car	Float	0.5
Lead Driver	The index of the driver who ends up ahead	Int	0
Second Driver	The index of the driver who is passed	Int	22

**Track**

Name	The referenced name of the event	String	JAPAN
Laps	The number of laps over which the race is run	Int	53
Length of straight	The total length of the overtaking straight, in metres	Int	1200
Pit loss	The time loss caused by making a pit stop	Float	21.0
Fuel consumption	The mass of fuel used per lap	Float	2.5

**Strategy**

List of stints	A collection of stints which make up the entire race distance	List<Stint>	
Time	The total predicted race time	Float	5500
Number of stops	The number of pit stops	Int	3

**Graph**

Data	The numerical data points which make up the graph	Float-array	
Collection of lines	The lines which make up the line graph	List<Line>	
Axes	Two perpendicular lines making up the axes. Also containing data about the scale	User defined structure	

Controls	The buttons and dynamic objects making up the graphing window	List<Control>	
----------	---	---------------	--

## Description of record structure

The system will call upon several different data sets, comprising of text files and database tables. Some of these will be static, i.e. they exist outside of the system and can be modified by the user in a text editor. The others are dynamic, having been created by the system. The user will be able to see this data, as it can be used for producing reports and analysing the data using tools other than those that the system provides.

This provides a short summary of the files that will be used by the system.

### Static Files

The first file that will be called upon is the driver file. This contains a list of the driver names, in race number order, and links each driver with a display colour that can be modified at any time.

Whenever a replacement driver is called upon, or in between seasons, the driver file will have to be updated. Its name and file path can be modified in the settings so if necessary a different file can be used, as long as it is of the same format.

There is also a track file; this is used for populating the track class. It contains a list of all of the tracks used by Formula 1 in a season, and data about the tracks such as the pit stop loss and the number of laps.

Finally, there is a settings file. This links directly to the settings form, and saves all of the data that is necessary to provide default values and to access the data in the first place. The file can be modified either within the system or outside of it.

### Dynamic Files

The system will generate archive files whenever data is entered from a PDF. These will be in a form that is more readable on later analysis, and also easier for the system to read back in later if necessary. There will be one file for every session that is entered, so to ensure proper locating and filing of the data the system will detect and generate a folder structure to store this data in.

It will also generate a backup of the driver parameters and the race positions, should these be required for analysis by hand at any time. I will also be able to use these files during debugging to ensure that the system is performing correctly, before I make the lengthy changes within the system to produce, for example, a graph of the results.

Of course, the system will also make use of the FIA PDF documents, which will have to be stored by the user in their own dedicated folder structure to be read automatically by the system.

### Databases

The system will also be linked to a database, which is where it will read data about the drivers and tracks from. The database link means that further development can include other data from the database, and write to as well as read from the data sets.

It is possible for the database to be stored on a network and therefore be accessible to multiple users within an organisation. In this context, it means that the system can be used to generate a simulation, the results of which are used by many different departments who do not need to learn to use the system, or run the system themselves.

## Validation required

The amount of validation that will be implemented into the program is simply too great to list here. There will be a significant amount of checking and double checking on almost every value because the data that is input does not come from a reliable source, nor is it specific to the current purpose. Hence, the data that it produces may also be ‘garbage’.

It is worth noting that I am generally only validating values to ensure the system functions smoothly. I am assuming that the operators of the system are sufficiently skilled that wildly incorrect values are unlikely. The use of the system should be fast enough that if an incorrect value is mistyped, for example, it can be corrected interactively once the results have been viewed.

Instead, here I will highlight the general principles of the validation that will be implemented by the system, and demonstrate some examples where appropriate.

### Pace parameters

#### **Initial checks**

The pace parameters will be generated completely automatically from the data that is collected from the PDF documents. As a result, it could be very far off what is actually desired from the system, and so there will need to be range checks. These will confirm that data is of roughly the right value to make sure that the system works.

For example, the system will only work if the tyre delta (the difference between the prime and option tyre times) is negative, and the tyre degradation on the hard tyre is less than the degradation on the soft tyre. These checks are therefore implemented, as below (taken from the driver class routine for calculating the option tyre degradation).

The checking also makes sure that the values are within acceptable limits – a degradation of over one second is very large, and is therefore checked for, and if necessary, removed as an anomaly.

```
if((sessionFastestLaps[i].degradation < sessionFastestLaps[i].fastestLap * 0.01)
```

The first line checks that the degradation is less than 1% of the fastest lap time, which is normally about a second. Because some laps are over 110s, and some are under 80, this takes the lap length into account.

```
&& (sessionFastestLaps[i].degradation > tyreDegradation[0]))
```

This is the line to check that the degradation is greater than the prime tyre degradation (tyreDegradation[0]).

```
{ optionDegradation += sessionFastestLaps[i].degradation; sessionsRead++; }
```

If both of these are found to be true, the value for option degradation is updated. The actual commands here involve an averaging system – if the value is acceptable it is added to a running total, and before the value is returned it is divided by the number of acceptable totals. The system averages between sessions and between teammates to ensure the greatest possible level of accuracy – up to four data points can be used to locate the degradation figure.

### **Secondary warnings**

When the values are displayed on a form, if they are more than 25% away from the default value, stored in the settings, they are coloured red. This is to bring it to the attention of the user that the value may not be correct, although the processing will still function in an acceptable manner without the value being changed.

### Strategy Optimisation

The system uses a rational method to calculate the number of laps that should be completed to optimise a stint length. This method involves dividing one floating point number by another, and although both should be rational and the result should be an integer, it is possible that this is not the case. Since the result is the number of laps to be completed on one set of tyres, it needs to be an integer.

As a result, a format check could be implemented on the value. However, this would be lengthy to implement, so instead I have chosen to ‘cast’ the given value to an integer:

```
roundedLength = (int)(value - remainder);
```

### Driver parameters

To give an idea of how the parameters are validated, a list of the validation checks that are completed on each of the properties of the driver class is listed below.

Parameter	Type	Typical	Validation	Explanation
Name	String	“Vettel”	Format check	The value will always be a string. It is likely an internal error has occurred if the value is not a string.
Pace	Float	91.302	No checks	This value is so vital to system function that if the system cannot return a value, it must be input by hand. Outputting a reasonable value would be misleading.
Prime tyre degradation	Float	0.3	> 0 < 0.5% of fastest lap	This must be within these bounds for the calculated stint length to be reasonable.
Option tyre degradation	Float	0.8	> Prime degradation < 1% of fastest lap	To make sure the result is sensible the system must

				use a value within these bounds
Tyre delta	Float	-0.6	< 0	A positive tyre delta will cause the system to fail by trying to create an infinite length stint.
Fuel Effect	Float	0.025	< 0.002% of fastest lap > 0	System may generate zero length stint if less than zero. Results will be unreasonable if greater than certain value
Fuel Consumption	Float	4.2	No validation	Not necessary – value is user defined and if wrong can be easily corrected by user.
Top Speed	Float	309.7	Presence check	If the value is not present it will be set to a default, user defined value. It is essential to system function but a default value is sufficient.
Position	Int	2	> 0 <= noOfDrivers	The position of a driver in a race will be between 1 and the number of drivers.
Interval	Float	0.805	> user defined default	Cars cannot follow each other closer than a certain spacing. This is dependent on the track and conditions, so is a user defined constant.
Gap	Float	11.200	No validation	No processing is done with this value so no validation is necessary
Lap Time	Float	91.865	No checks	It is impossible for the system to logically calculate if the value is wildly incorrect.

### Settings

Because the settings will provide defaults for much of the system, it is essential that the values stored here are correct. If they are in the wrong format or of the wrong value, it is possible that the system will not function properly. As a result, every parameter is validated to ensure that it is correct.

All values in the class will have a presence check as they are all essential to system function. The system-generated code provides a length check of 255 as all of the values must be strings.

Parameter	Type	Typical	Validation	Explanation
Required Pace Delta	Float	0.5	$\neq 0$	If equal to zero, the system will try to divide by zero
Required Speed Delta	Float	10	$\neq 0$	If equal to zero, the system will try to divide by zero
Time Loss when overtaken	Float	0.5	$>$ Time gap in traffic	This ensures that when the loss is applied, an overtake is certain to occur.
Time Loss when lapped	Float	1	No validation	The system will function with any value.
Time Gap in traffic	Float	0.4	$> 0$	In order to follow behind another car, the time gap must be positive.
Default Tyre Delta	Float	-0.8	$< 0$	The system will fail with a positive tyre delta
Default Option Degradation	Float	0.05	$>$ Default Prime Degradation	This will never be less than the prime degradation
Default Prime Degradation	Float	0.02	$> 0$	The system will fail if this is less than 0.
Default Top Speed	Float	300	No validation	Any value can be used for this parameter
Default Fuel Effect	Float	0.035	$> 0, < 0.1$	The system will fail if this is less than zero If it is too large, the system produces unreasonable results.
Default Pace	Float	200	No validation	There are no reasonable bounds that I can justify forcing this value between
Pit Lane Loss	Float	21.2	$> 0$	The system will fail if this is below zero
Default Fuel Load	Float	60	$> 0$	The system divides by this value so it cannot be zero
Default Data Folder	String	..\..\..\	Format check	Must be a valid file path otherwise data will not be read correctly
Name of Driver File	String	“Drivers.txt”	Format check	Must be a valid file name or data will not be read correctly.
Name of	String	“Tracks.txt”	Format check	Must be a valid file

Track File				name or data will not be read correctly.
Track Evolution	Float[]	0.6, 0.4, 0.2, 0, -0.2	No validation (== 0)	The system can and may use any value for this parameter. (The fourth index must be zero as it is the track performance in qualifying, on which everything else is based.)
Track Improvement	Float	0.1	No validation	The system can and may use any value for this parameter.
PDF Reader Name	String	AcroRD32	No validation	It is not possible to validate this because the check that would have to be done requires a list that is protected on most computers.

### Validation Routines

Examples of some of my custom routines for validation are shown below. They will return the nearest boundary value if the value is not in range, and also set a Boolean variable to false if the value is out of the required range. A combination of these three routines can perform any range or length check required, depending on the values that are passed to the routine.

These are static routines in a class of useful functions which can be accessed from anywhere within the program. This will be a useful class to implement in other projects, so it is an excellent example of transferrable code.

I can build my own custom routines to combine permutations of the ‘not equal to’, ‘greater than’, and ‘less than’ routines. This makes it very simple to validate all of the values that I need to in a quick and effective manner. It should also be very clear what the code is doing if all of the subroutines are given proper names. This will aid readability and make sure that the code can be maintained by others if necessary.

### Pseudocode

```

IF value > boundary
THEN
    value ← boundary
    accepted ← false
END IF

```

This routine will prevent a value from being entered that is greater than a given boundary value. It will also set a flag to false so that the system knows an incorrect value has been entered, and can ask for it to be re-entered.

**Program Code**

```
//assign to incorrect value
incorrectValue = (value < notLessThan);

//switch on boolean for speed of execution
if (incorrectValue)
{
    //display a warning message
    string warningMessage = "The field " + field + " cannot be less than " +
Convert.ToString(notLessThan);
    var warning = MessageBox.Show(warningMessage, "Incorrect Value",
MessageBoxButtons.OK, MessageBoxIcon.Warning);

    //set the value
    value = notLessThan;
}

//return the value
return value;
```

The above code implements the Pseudocode in a slightly different structure, which is marginally faster to execute. It also ensures that the Boolean is always assigned to, so there is no chance of a null reference; this improves the robustness of the code.

## File organisation and processing

My system will use three different types of file, and in order to make maintenance easier, all three will be allowed to take their own separate directories.

The system will only handle text files, and will read and write to these files. Wherever this is done within the program, data is passed to or from a dedicated file processing class/object. This makes it very easy to adapt the classes if necessary, so that if file formats are altered or the system needs to write or read a different file, changing the system to handle this is comparatively easy.

The classes will contain methods for writing/reading the file, as well as whatever analysis is necessary.

All of the files will be sequential access because this is very fast and efficient. The data from these files will all be passed into variables in the program. File IO is slow, so the more information that can be stored in main memory, the quicker the program will run.

### Initialisation Files

The system will load three files from a directory on start-up, which provide sufficient data to get the system started.

Firstly, a settings file will be loaded. This must be installed at a location which is hard coded into the program, because otherwise the system cannot load the settings data. The settings file itself contains all of the required file paths and the default values for the rest of the system so it is important that it is loaded first.

Once the settings have been loaded, the system can load data about the tracks and drivers, from the file locations specified in the settings. This data initialises the list of tracks and drivers so that processing can begin.

All three files are currently located in the same directory, which is itself in the same directory as the system's shortcut. This way the files are all organised in the same location and they can be easily accessed manually.

All of the files contain one object per line; the driver file has 22 lines, one for each driver, and each line will populate an instance of a driver. Similarly the track file initialises an instance of a track. The processing is very simple; the system reads the whole file sequentially, line by line. Each line is comma separated, so the line string is split at the ‘,’ character, forming an array of values. These values are converted to the relevant data type and saved, before the next line is read. It is a very fast and simple way of reading data from a file that can be organised simply by humans.

The comma separated format means that the data could, if necessary, be read by Microsoft Excel.

### Data Input Files

The system will input data from PDF files that are located in a specified file location. The settings file defines the directory that these will be located in. Within this specified directory, there must be a folder for every race. The name of these folders must match with the names defined in the track data file, so that the correct file can be identified and loaded when it is selected in the data input window.

These files are located and opened by the system. The user will then have to copy the entire contents of the file for the system to process, which is done in the previously explained manner of populating a class with the data. This class will process the data, and return it to the relevant drivers.

### Data Output Files

#### **Backup/Restore Files**

It is essential for the system that data is saved as it goes along. It is also important for the system to be able to restore from this data, in case a race simulation needs to be run with two different configurations for the results to be compared. It should be possible to save data at the time that a race simulation is run, and re-load the data when it is required.

This functionality will also be useful if the system crashes at any stage. It can be restored to its previous state very easily if the data has been saved.

The backup and restore files will be implemented when the drivers' pace parameters have been calculated, so that these can be backed up or analysed using different tools, and it will again save data when the strategies have been defined. At each of these stages, the user can edit the data manually to see what the result is, and it should be possible to restore the system if the user makes a mistake on data entry.

These files will be processed, again, by defining a class that can perform the read/write operations. They will be located in a dedicated directory in the top level system folder, so that they can be accessed manually by the user if required.

#### **Archive Files**

The archive files contain data that is simply regurgitated from the PDF input files, except read into a slightly different format. These files are located in their own directories with the same structure as for the input files; they can use the same directories if necessary.

These files are processed using the same class that both interpreted data from the PDFs and wrote it out to the files. The system should be able to read in the data from these files and put it straight into the driver classes, so that the rest of the system can be run properly.

### **Data Analysis Files**

When strategies are simulated or a race simulation is run, the system save the results of the simulation in a comma separated file, so that the results can be analysed in other programs if necessary.

This will be particularly important if data from an actual race is loaded, because the engineers may want to look at the times from this in very close detail. Being able to convert the PDF into usable text in seconds will be a very useful tool for the engineers.

As an exception to the rule of one class per file, the data here will be written by the class that generates the race simulation, or the strategy simulation. This is because the iterative nature of these simulations makes it far more efficient to write as the simulation runs; the file will be opened before the simulation runs, and then in each iteration of the loop data will be written.

## Identification of appropriate storage media

### Program Distribution

To distribute the program, a CD is the best method. I can write the installation files, the executables, and any required instructions and help to this CD and it can be kept as a backup copy wherever necessary.

The software can be installed onto a central server and distributed from there onto every machine in the business very quickly, which makes it easy for the end users to access and use.

There will never be any sensitive data on the CD, so there should never be a problem with security this way.

### System Storage

It is worth noting that the storage of setup and timing archive data is best kept to the server as this reduces the number of duplicate copies that need to be kept. It will also mean that the system can be run more quickly in most cases, with data pulled from the server rather than being loaded from the PDFs when the system setup occurs.

Settings data has to be stored locally because it can be modified within any copy of the program.

### Desktop Use

Some data will be stored on the hard drive locally – this will include the backup and results files that are specific to each individual simulation.

The system will predominantly be run on desktop PCs, and due to the small volume of data that requires storage there is no reason that this cannot be kept on the computer's hard drive.

Other systems will need to be available so that the user can upload this data to the centrally accessible server, but such systems do exist currently. The data can be uploaded to the server when necessary so that any other engineers in the company can access the data – for example if an optimal simulation has been reached and the parameters need to be distributed.

### Mobile Use

The system may also be used by the race team on the pit wall. In this case, data will have to be stored on the local network to increase speed, and the system will essentially run in parallel with the systems at base.

The laptops have plenty of internal storage so this will be utilised. Communications links are still available back to base if setups and simulations need to be shared.

It is also possible that a screen capture could be sent over these communications links, so that while only one copy of the system is used by one engineer, it can be seen by many. This will be of use in the strategy planning meetings, where a teleconference is already implemented.

## Identification of processes & suitable algorithms for data transformation

There are some algorithms in the system that will be very complex in terms of logic. I will need to design these step by step before they can be automated and tested, so I started by designing them in pseudocode. These were developed before they were programmed and fully integrated.

### Reading from PDF

One of the key features of the system is the ability to read data from a PDF file into the parameters of the system. There are several steps to be done before it is in the form of data that can be processed to define strategies, so it is split into different subroutines that can each take a section of the problem and solve it.

The first problem that occurs is how to find the relevant lap time data in the PDF – how to recognise the right format. The system must be programmed to understand how the PDF data will appear, and how to recognise a line that has a lap time in it. It will then have to take the line, work out which driver it belongs to, which stint the driver is in, calculate the lap time itself, and then add the lap time to the relevant stint.

As a result, there are a lot of complex steps to be done.

### Finding the right line

An example of data copied from a file is shown below. The data will be read sequentially. Instead of trying to read each line individually, I will use the patterns in the data to find the lines that the system must read.

First Practice Session Lap Times

© 2013 Formula One World Championship Ltd, 6 Princes Gate, London, SW7 1QJ, England.

and related marks are trade marks of Formula One Licensing BV, a Formula One group company.

The F1 FORMULA 1 logo, F1 logo, F1 FIA FORMULA 1 WORLD CHAMPIONSHIP logo, FORMULA 1, FORMULA ONE, F1, FIA FORMULA ONE WORLD CHAMPIONSHIP, GRAND PRIX

the results/data relate and provided that the copyright symbol appears together with the address shown below without prior permission of the copyright holder except for reproduction in local/national/international daily press and regular printed publications on sale to the public within 90 days of the event to which

No part of these results/data may be reproduced, stored in a retrieval system or transmitted in any form or by any means electronic, mechanical, photocopying, recording, broadcasting or otherwise

2013 FORMULA 1 JAPANESE GRAND PRIX - Suzuka

1 S. VETTEL

NO TIME NO TIME

1 P 10:04:19

2 P 2:35.269

3 36:07.465

4 1:35.217

5 1:46.635

6 1:35.049

7 1:44.842

8 1:34.768

9 P 1:51.469

10 12:20.894

11 1:39.014

12 P 1:59.415

13 2:18.419

14 1:38.801

15 1:38.96

For example, we can completely ignore any line before the line “1 S. VETTEL”. This line is important because it defines the driver that the subsequent data represents.

Lines will be read until this type of line is found. I can run a format check on this line to make sure it is in the right format, or, better, I can read the number at the front of the line, and then the name. If the number matches the name of the driver, I know that this is a valid line. At this point we start looking for lap times instead of driver names.

To read the lap times, I need to recognise if the number at the start of the line is the same as the number of laps that have been processed. If this is true, the line will contain a lap time. This can be read.

The following Pseudocode routine is designed to carry out the above steps, and isolate the right line. Once the right line has been found, it is processed in a different routine (demonstrated later).

```

DO WHILE not end of file
  IF driver not found
    THEN
      Get next line of file
    END IF

    Get name of driver from file

    IF name of driver matches a driver name
    THEN
      increment driver index ready for next driver

      start new stint
      set car on an out lap

      DO WHILE line from file contains a lap time
        get next line of file
        get lap number from line from file

        IF lap number from file matches number of laps read
        THEN
          Process data
        ELSE
          finished reading data
        END IF
      LOOP

      IF stint is valid
      THEN
        Add stint
      END IF
    ELSE
      IF finished reading data
        Get next line from file
      END IF

    END IF
  LOOP

```

The routine is further developed below:

**VARIABLES:**

`lineInput` (string containing data from the file. Data is extracted from this.)  
`lapNumber` (integer representing the lap number through a session. This is compared to the number in the file to check the line is correct for reading.)  
`lineStatus` (integer specifying what to do with the current line in the file:  
  0: waiting to read data  
  1: reading data - driver has been found  
  2: finished reading data - waiting to find next driver)  
`driverIndex` (integer specifying the driver that is currently being read)  
`readStatus` (integer specifying the status of the car in the stint:  
  0: Car is in the garage after in-lap.  
  1: Car is on an out-lap.  
  2: Car is on-track and lap times are being added to the stint.)  
`tempStint` (Stint that data is written into temporarily. It is added to the driver's sessions when fully loaded)

```

DO WHILE lineInput != null
  IF lineStatus = 0
  THEN
    lineInput ← getNextLineToInput
    lineNumber ← lineNumber + 1
  END IF

```

```

IF line Contains Name
THEN
    lapNumber ← 0
    lineStatus ← 0
    driverIndex ← driverIndex + 1

    tempStint ← new Stint
    readStatus ← 1

    DO WHILE lineStatus != 2
        lineInput ← getNextLineToInput
        lapNumberFromLine ← getNumberFromLine
        lapNumber ← lapNumber + 1
        IF lapNumberFromLine = lapNumber
        THEN
            Process Data
        ELSE lineStatus ← lineStatus + 1
        END IF
    LOOP

    IF lapsInStint != 0
    THEN
        Add stint to session
    ELSE
        IF lineStatus = 2
        THEN
            lineInput ← getNextLineToInput
        END IF
    END IF
END IF
LOOP

```

The final code for the routine is shown below:

```

public static void collectData()
{
    string lineInput = "";
    int lineNumber = 0;
    int lapNo = 0; //number of laps in driver's stint
    int lineStatus = 0; //0 is no numbers read, 1 is reading, 2 is finished
reading
    int driverIndex = -1;
    int readStatus = 0;

    Stint tempStint;
    do //terminates when file ends.
    {
        if (lineStatus == 0)
            lineInput = getNextLineToInput(lineNumber++);

        if (nameCheck(lineInput) == true)
        {
            lapNo = 0;
            lineStatus = 0;
            ++driverIndex;

            tempStint = new Stint(driverIndex, Program.sessionNumber,
lapNo);
            readStatus = 1;

            do
            {

```

```

        lineInput = getNextLineToInput(lineNumber++);
        if (getNumber(lineInput, 0) == lapNo++)
        {
            useTimeData(lineInput, lapNo - 1,
            driverIndex, ref readStatus, ref tempStint);
        }
        else {lineStatus++;}
    } while (lineStatus != 2); //exits when times are finished

    if (tempStint.lapTimes.Count != 0) {
        Program.drivers[driverIndex].sessions[Program.sessionNumber] += tempStint;
    }
    else
    {
        if (lineStatus == 2)
            lineInput = getNextLineToInput(lineNumber++);
    }
} while (lineInput != null);
}

```

### Processing the Data

To actually retrieve a lap time from the line is still a challenge. The line, some examples of which are shown below, will contain a lap index, may contain a pit marker, and will contain a number with one or two minutes digits, two seconds digits, and three decimal digits.

2 P 2:35.269  
3 36:07.465  
4 1:35.217

I also need to know if the P is present, and then act upon this information to determine if the driver has pitted or not. This routine will be called from within the previous routine, so I can pass a number of variables to and from the process data procedure.

```

Remove lap number from string
IF string contains a P
THEN
    Remove 'P' from string
    IF car was previously on track
    THEN
        Add stint to driver
        Start new stint
    END IF

    Set car in garage
END IF

Get lap time from string

IF car is on track
THEN
    Add lap time to stint
ELSE
    Update car status
END IF

```

This routine is developed below, so that variables are updated as required in the previous routine.

```

Remove lap number from time
IF time contains P
THEN
    Remove P from time

    IF readStatus = 2
    THEN
        Add stint to session
        tempStint ← new Stint
    END IF
    readStatus ← 0
END IF

lapTime ← getLapTime

IF readStatus = 2
THEN
    Add lap to stint
ELSE
    readStatus ← readStatus + 1
END IF

```

The final code for the routine is shown below, with comments to aid with clarity:

```

float lapTime;
//readStatus - 0: waiting for start, 1: on-track, 2: add lap.

time = time.Remove(0, (lapNo >= 10 ? 3 : 2)); //takes away the lap number
if (time[0] == 'P')
{
    time = time.Remove(0, 2); //removes the 'P'

    if (readStatus == 2)
    {
        //pass old stint
        Program.drivers[driverIndex].sessions[Program.sessionNumber] += tempStint;

        //start new tempStint
        tempStint = new Stint(driverIndex, Program.sessionNumber, lapNo);
    }

    //get ready to read new stint
    readStatus = 0;
}

//load the lap time from a string
lapTime = getLapTime(time);

if (readStatus == 2) //if reading
{
    //adds lap time to stint
    tempStint += lapTime;
}
else
{
    //increments readStatus if not already reading
    readStatus++;
}

```

## Overtaking Model

In order to achieve a probabilistic overtaking model, I needed to develop an algorithm that would calculate the chances of an overtake occurring. This is best shown in Pseudocode as below, demonstrating the principles and aims of the routine.

The basic principle of the routine is to find the difference between the required delta, defined in the system settings, and the actual delta, calculated from the race simulation. In order to get a probability, this is scaled based on the magnitude of the required delta, producing a good approximation of the likelihood of an overtake occurring.

For example, if one driver's pace is 0.2s faster than the other, and the required pace delta is also 0.2, then the probability of an overtake occurring is zero (the difference between the values is zero).

However, if one driver was 0.5s faster than the other, the probability would equate to  $0.3/0.2 = 1.5$ . Although this is completely meaningless as a probability in the true sense, it does suggest that if one driver is much faster than another an overtake is very likely. It is worth mentioning that if the speed probability was less than 0.67, there would still be a chance that the overtake would not occur.

There will inevitably need to be refinement on the scale factors and distributions, because a small change to the overtaking model can make a big change to the results of the simulation.

### VARIABLES:

```

Rand (random number used to represent the likelihood of an overtake)
overtake (boolean representing if an overtake will occur)
speedProbability (decimal, the speed component of the total probability)
paceProbability (decimal, the pace component of the total probability)

cumulativeTimeDelta ← behindCumulativeTime - aheadCumulativeTime
paceDelta ← behindLapTime - aheadLapTime
totalDelta ← cumulativeTimeDelta + paceDelta

speedDelta ← behindTopSpeed - aheadTopSpeed

IF requiredPaceDelta > totalDelta
THEN
    paceProbability ← (requiredPaceDelta - paceDelta) /
    Abs(requiredPaceDelta)
ELSE
    paceProbability ← 0
END IF

IF requiredSpeedDelta < speedDelta
THEN
    speedProbability ← (speedDelta - requiredSpeedDelta) /
    Abs(requiredSpeedDelta)
ELSE
    speedProbability ← 0
END IF

probability ← speedProbability * paceProbability

```

```

random ← new random
overtake ← is (probability > random)

return overtake;

```

The final code for the routine is below. In some cases, long declarations have been replaced with simple variable identifiers, for the sake of readability. However, the logic is the same.

```

Random rand = new Random();
bool overtake = false;
float speedProbability = 0;
float paceProbability = 0;

float cumulativeTimeDelta = behindCumulativeTime - aheadCumulativeTime;
float paceDelta = behindPaceDelta - aheadPaceDelta;
float totalDelta = cumulativeTimeDelta + paceDelta;

float speedDelta = behindTopSpeed - aheadTopSpeed;

if (requiredPaceDelta > totalDelta)
{
    paceProbability = (requiredPaceDelta - paceDelta) /
Abs(Program.settings.requiredPaceDelta);
}
else
{
    paceProbability = 0;
}

if (requiredSpeedDelta < speedDelta)
{
    speedProbability = (speedDelta - requiredSpeedDelta) /
Abs(Program.settings.requiredSpeedDelta);
}
else
{
    speedProbability = 0;
}

float probability = speedProbability * paceProbability;
float random = (float)rand.Next(0, 1001) / 1000F;
overtake = (probability > random);

return overtake;

```

### Strategy Modifications

When I modify strategies, various operations will need to be completed on the race stints. One of these is to split the stint in half – which occurs when a stint is ‘added’. The program must split the stint, and then re-assemble all of the unchanged stints with the new stints. There will also be routines to swap stints – re-ordering two different stints without changing them – and one to merge two stints (analogous to removing one).

To tie up the stints and make sure that the strategy consists of the right number of laps, and that the lap times are correct, requires further methods. What is shown below is simply the code that operates on the stints.

These methods are contained within the stint class. The split method is dynamic, but the merge and swap methods are static as they both operate on two stints, therefore cannot be called as a method of one.

The Pseudocode for the three routines is below:

```

BEGIN Split
VARIABLES:
    stintToSplit (stint representing the stint that is to be split)
    splitStints (array of two stints that is the result of the split)
    splitLap (the lap number in on which to split the stints)
    firstStintLength, secondStintLength (integers giving the length of
stints)

    firstStintLength ← splitLap - stintToSplit.startLap
    secondStintLength ← stintToSplit.startLap + stintToSplit.stintLength -
splitLap

    splitStints[0] ← new Stint
    splitStints[1] ← new Stint

    splitStints[0].modified ← true
    splitStints[1].modified ← true

    return splitStints
END

BEGIN Merge
VARIABLES:
    a, b (the stints to be merged)
    mergedStint (the result of the merge operation)
    startLap (the starting lap of the stint)
    endLap (the finishing lap of the stint)

    mergedStint ← a
    startLap ← a.startLap
    endLap ← startLap + a.stintLength + b.stintLength

    mergedStint.lapTimes ← null
    mergedStint.stintLength ← endLap - startLap

    mergedStint.modified ← true

    return mergedStint
END

BEGIN Swap
VARIABLES:
    tempStint (a temporary holder for use in the swap)
    a, b (the stints to be swapped, passed by reference)
    stintAstartLap (the startLap of the earlier stint, before swap)

    stintAstartLap ← a.startLap

    tempStint ← a
    a ← b

```

```

    b <- tempStint

    a.startLap <- stintAstartLap
    b.startLap <- stintAstartLap + a.stintLength
END

```

The fully coded routines are below. The constructor for the stint class, used in the split routine, takes the driver index that it belongs to, the start lap, tyre type, and length, and creates an empty stint with these parameters set. It can be assumed that this line of code creates an empty stint (i.e. without any laps in it).

```

public Stint[] Split(int splitLap)
{
    Stint stintToSplit = this;
    Stint[] splitStints = new Stint[2];
    int firstStintLength, secondStintLength;

    firstStintLength = splitLap - stintToSplit.startLap;
    secondStintLength = stintToSplit.startLap + stintToSplit.stintLength -
        splitLap;

    splitStints[0] = new Stint(stintToSplit.driverNumber,
        stintToSplit.startLap, stintToSplit.tyreType, firstStintLength);
    splitStints[1] = new Stint(stintToSplit.driverNumber, splitLap,
        stintToSplit.tyreType, secondStintLength);

    splitStints[0].modified = true;
    splitStints[1].modified = true;

    return splitStints;
}

public static Stint Merge(Stint a, Stint b) //returns a stint of the correct
length with no lap times included
{
    Stint mergedStint = a;
    int startLap = a.startLap;
    int endLap = startLap + a.stintLength + b.stintLength;

    mergedStint.lapTimes.Clear();
    mergedStint.stintLength = endLap - startLap;

    mergedStint.modified = true;

    return mergedStint;
}

public static void Swap(ref Stint a, ref Stint b)
{
    Stint tempStint;
    int stintAstartLap = a.startLap;

    tempStint = a;
    a = b;
    b = tempStint;

    a.startLap = stintAstartLap;
    b.startLap = stintAstartLap + a.stintLength;
}

```

## User interface design (HCI) rationale

### Planning

The system will be designed with multiple forms. Each form will roughly represent an entity; after the start page, there will be a settings form to control the settings, a timing archives form to control the archives, and then the main strategy processing. This will comprise four forms, a data input form, which represents a PDF and controls the input of timing data, a timing analysis form, which represents all of the drivers, a strategy form, which represents the strategies, and a graph.

The graph will represent both strategies and races, and will come in different types as for the current system – a ‘pit now’ graph and a ‘cumulative time’ graph.

All of the forms will have the standard minimise, restore and close buttons, as well as user controls to affect the display of the form, or for data entry. The forms will also contain controls to move between forms.

To summarise, the system will consist of multiple GUI view objects, which are populated by data from the system ‘model’. The interface between the model and the view, known as the view-model, controls the processing and the running of the system. This is not a full MVVM structure but represents one, and is an efficient way of organising the user interface and the data within one project.

### Sample of planned data capture and entry designs

To a large extent, the system does not require any manual data input. This is because it was identified as one of the flaws of the current system. Instead, the user points the system in the direction of the data, and the system then goes and fetches it.

There are some windows that will require actual user data entry, which are described below.

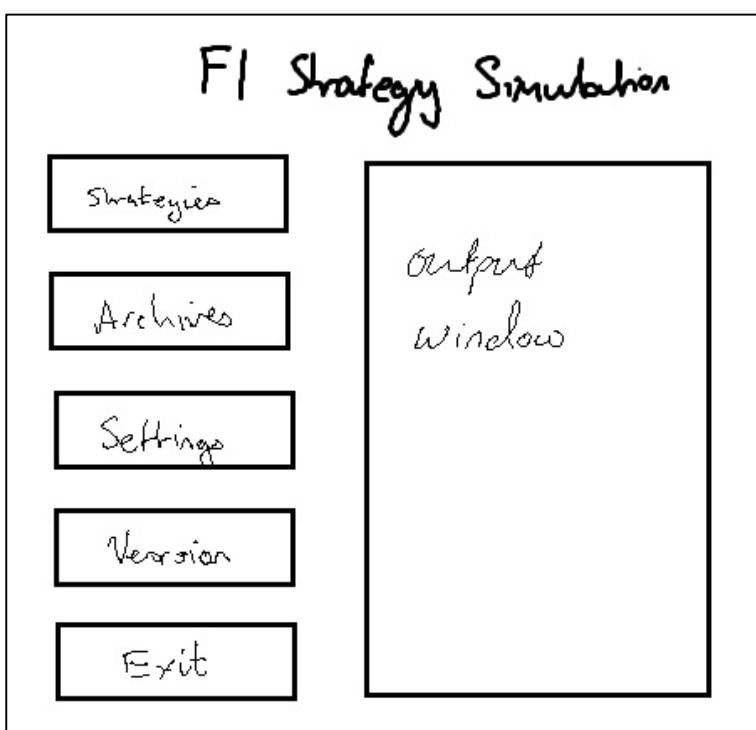
The nature of the system means that some windows that contain calculated (output)

data will also require input data. These are dealt with in a separate section.

### **Start Page**

The start window will be the only page to contain a title on the page. I feel that it looks better with the title in place.

The buttons on the left all refer to separate branches of the structure chart, and will



be the same size and shape. This will make it clear and easy to decide what the user wants to do.

On the right is an output window. This will display a commentary of what the system is doing, so that the user knows what has been done recently if, for example, they return to it after a break, or if someone else uses the system in the meantime.

The form will have the usual three buttons in the top corner, with exit closing the window. When the window is closed, the system will also be closed, therefore this window controls the running of the system.

### Data Input Window

The diagram shows a window titled "Data Input" (1). It contains three dropdown menus (2) labeled "Race:", "Session:", and "Data Type:". Each dropdown is accompanied by three buttons: "Confirm Entry", "Add new file", and "Get parameters". There are also three numbered labels: "3" at the bottom center and "4" on the right side.

This is the window from which the user will select the type of data that they want to process. Once the combo boxes have been populated, the system can open the specified file, ready for the user to load data from it. This window appears when the 'Strategies' option is selected from the main menu.

1: The title bar – includes the default three buttons for the window, all of which have functionality, and the name of the window. This will be displayed on every form, and will replace having an actual title on the form. I hope that this will make the windows smaller and more efficient.

2: Labels – the labels guide the user into what they should select from the combo box.

3: Combo boxes – these contain all of the available options for this parameter. Once all three have been defined, it is possible to locate the file that requires opening, and work out its name, so that it can be opened for the user to copy the data.

4: The buttons allow the user to confirm that the data has been entered, add a new session to the loaded data, and advance the system by processing the data. There is validation behind these buttons so that the system warns the user when they try to advance without loading all of the required data. This comes up as a warning message that must be dismissed and rectified, or ignored if necessary.

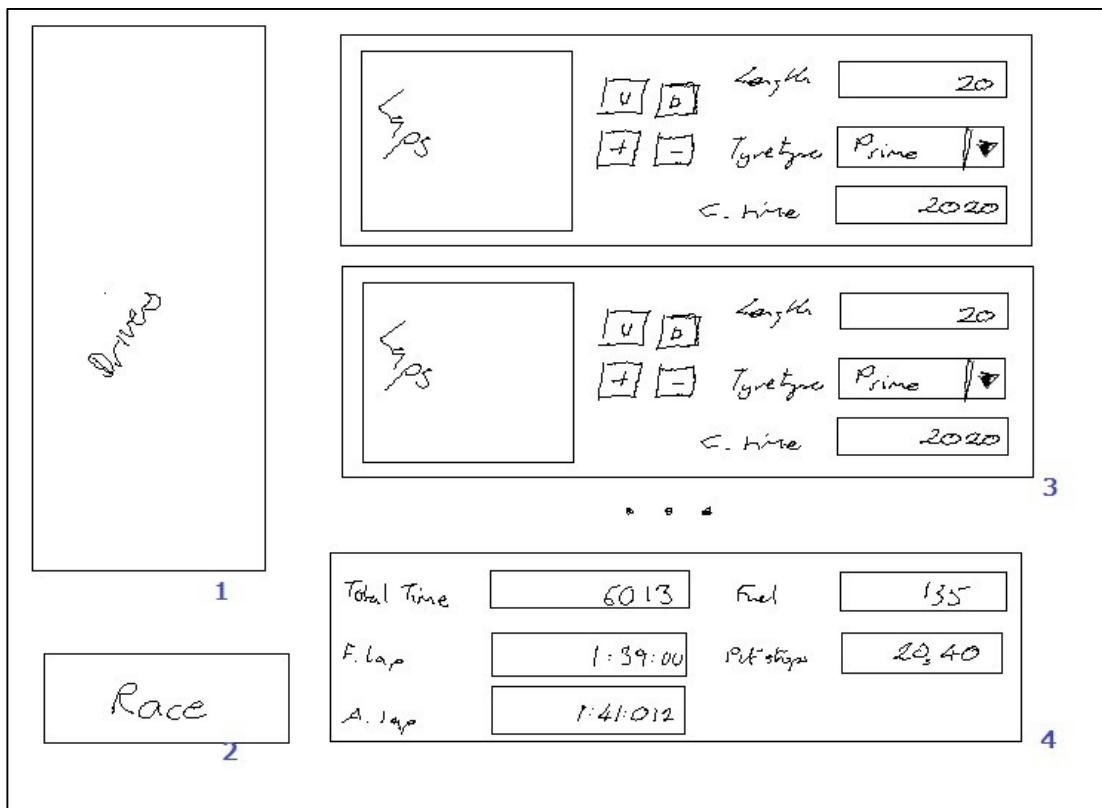
**Input/Output Windows****Time Analysis**

The screenshot shows a window titled 'Time Analysis'. At the top right, there is a number '4'. On the left side, there is a vertical column of numbers from 1 to 10. The main area contains a grid of 10 columns and 10 rows of text boxes. Some cells are highlighted in yellow or blue. At the bottom of the window, there are two buttons: 'Refresh' and 'Get Strategies'. To the right of the 'Get Strategies' button is a number '2'.

The time analysis window displays the calculated driver parameters. These have been calculated from the data that was input from files. The majority of the window will be a table of text boxes containing the relevant data, with driver names and headings on the left and top sides respectively. The form is dynamically generated to handle the large number of text boxes that will be present.

- 1: The driver names and numbers will be listed down the left side. They will be ordered in race number order.
- 2: The buttons at the bottom of the form allow the form to be refreshed, resetting data to its defaults, or allow the user to run a strategy simulation, taking the parameters and working out the optimum strategies to implement.
- 3: The text boxes will display the data that was loaded. They will have two layers of validation – if the data is wildly out of the expected range, a default value is used and the cell is coloured yellow. If it could be out of range but is within acceptable limits, the font will be red. The text boxes allow the user to enter data into them too, so it is possible for the user to change data by overwriting the value in the text box. In this instance, it may still have red font, but the cell will turn blue, to signify that the value has been manually changed.
- 4: The headings are labels, containing the name of the data that is included in the cells directly below.

## Strategy Viewer



The strategy viewer window will show the strategy that has been chosen for any selected driver. For each stint in the race it displays a standardised panel, filled with details about the stint. It can only display one driver at a time, but updates automatically according to the selection in the ‘drivers’ list box. The form will be dynamically generated to allow different numbers of stints to be accommodated.

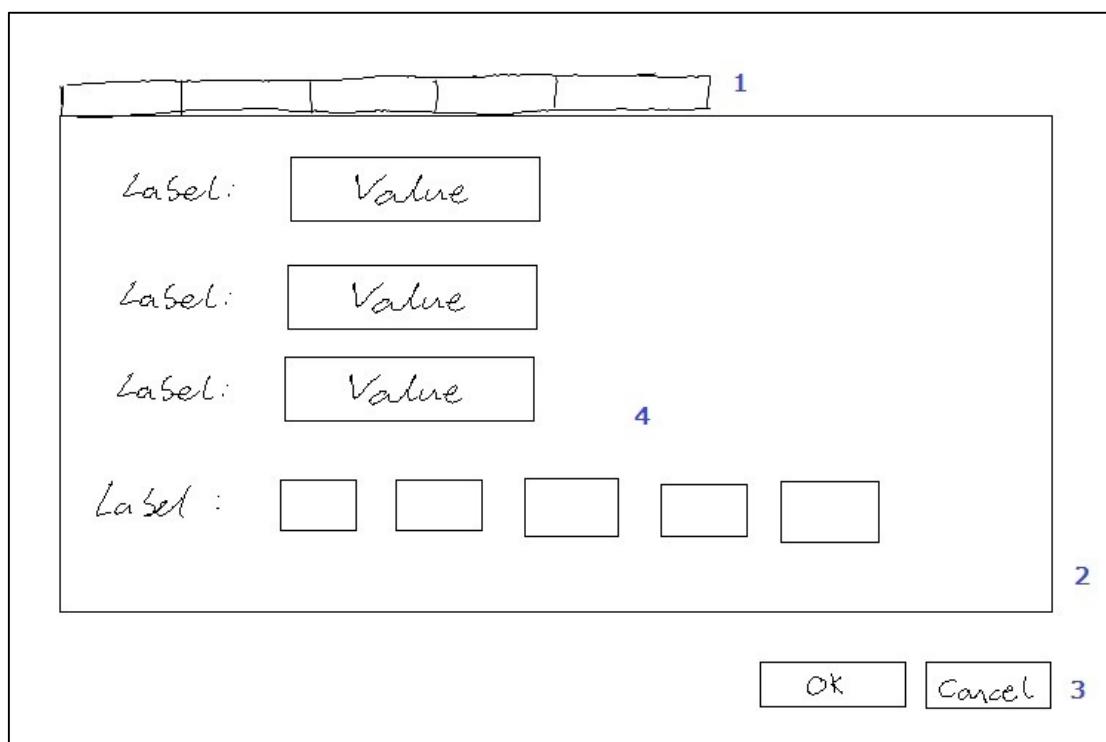
1: The list box is populated with all of the driver names. The selected driver defines what stints are displayed. If no driver is selected, no stints will be displayed.

2: The ‘race’ button will run a race simulation – producing a graph of the results (see below). It combines all of the driver strategies as they are defined in this window, so edits made by the user will be carried into the race simulation.

3: The standardised stint panels display a list of the lap times, a collection of buttons, and the length, tyre type, and total time for the stint. The tyre type and stint length can be altered by the user and the stint will update accordingly. The buttons, of which there are usually four, allow stints to be added, removed, and moved up and down. These invoke the stint modification methods seen in the Pseudocode section.

4: The window will also display a summary panel for the stints. This will contain details of the total time that the race will take this driver, the fastest and average lap, and the list of stops and fuel required. This will help to differentiate between a faster and a slower strategy. It will, again, update automatically when the user selects a different driver or changes the strategy.

## Settings



The settings form displays settings data and allows the user to change it. When the settings form is closed with the 'ok' command, data is updated in the settings, and then simulations can be re-run instantly to see the effects.

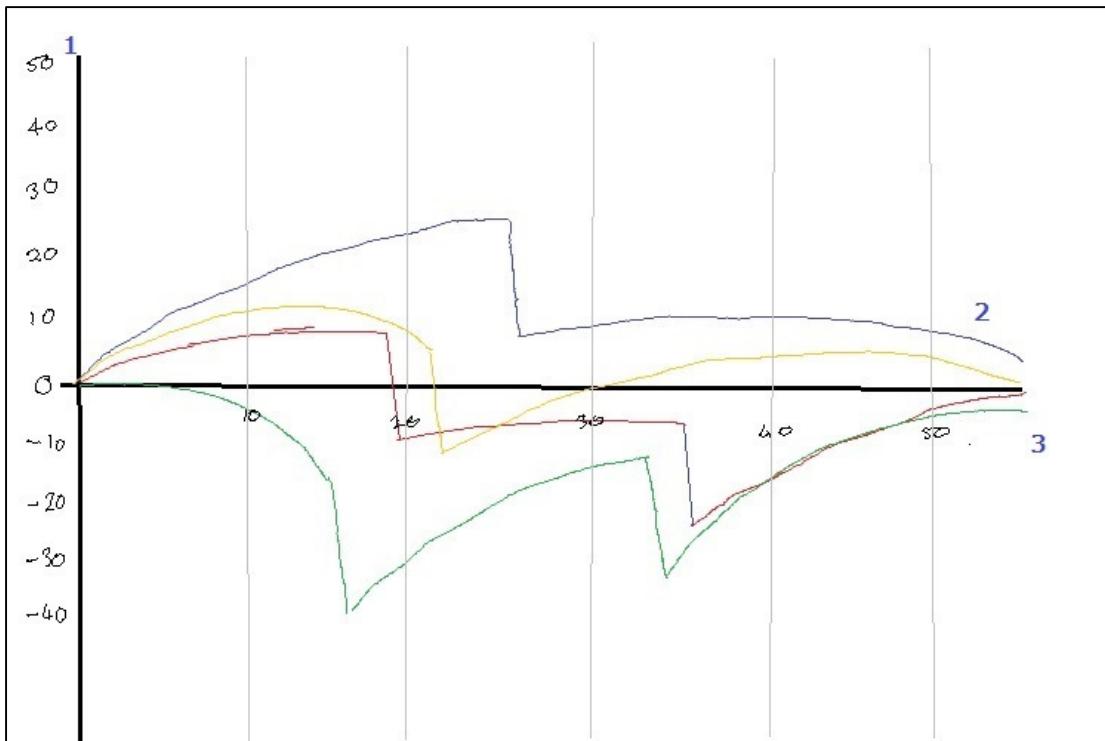
Because there is so much data to display on this form, it is separated into a tabbed control. As well as making the form look nicer, it means that the data is better organised; data with similar relevance is collected in to the same place, which is a more user-friendly layout. The form will not be dynamically generated because visualising the layout and working with the tabs is easier to manage through the designer. The number of events on the form is also limited, which means that for readability purposes it is not necessary to keep the form dynamically generated.

1: The tabbed control – there will be a series of tabs, each named according to the data contained within them. Selecting the tabs displays different forms of data.

2: The panel size – the form will be the same size regardless of the tab that is selected, so that when organising the screen the user can have the most flexibility possible.

3: The standard OK and Cancel buttons retain their normal effects. The OK button validates the input and updates the settings data with any changes. The cancel button will close the form without making any changes.

4: The contents of the tabbed panels will normally be two columns of labels and text boxes. The labels define the data that will be displayed in the text boxes. There will be some cases when multiple values need to be displayed, for example in the settings for track evolution. In this situation, there will be multiple text boxes generated.

Sample of planned valid output designs**Graph**

The graph will be used to display both strategies and the race simulation. In the first case, it will show only the selected drivers, and these drivers will not interact with each other. In the second, it will show all of the drivers (with options to hide some), and overtakes, lapped car interactions, and traffic effects will all be simulated.

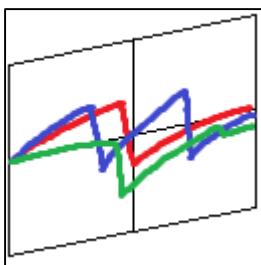
Each driver will have their own colour on the graph, which can be displayed for ease of identification. The axes will be scalable so that a particular part of the graph can be seen more clearly if necessary. This will help the user to see the exact amount of detail that they want to see.

1: the vertical axis will represent a time delta from the normalised time. The normalised time will be a straight line along the horizontal axis; if the traces are ahead of the normalised time (i.e. their time is less than the normalised time) they will be above the axis, if not they will be below. In the sketch shown above, the graph is normalised on the average time for the red car. Blue and yellow are both ahead for a long way, with green behind.

2: the traces will be simple line graph plots of cumulative time against laps. The gradient can therefore be taken as the driver's current lap time, with flat being equal to the normalised lap. They will be made up of a collection of lines from the point on one lap to the point on the next, giving the illusion of a curve.

3: The horizontal axis represents laps. This will start at one, and terminate at the number of laps in the race.

## Icon



The icon for the system has been chosen to be as simple as possible, while still clearly showing what the system is for. It has enough colour to be clear, even at small scales.

This will be displayed on shortcuts, on the windows of the program, and on the task bar when the program is running.

## Summary

There are some functions and forms that have not been included here because they either do not input/output data, or because to include them would add complexity to the drawings. For now, the drawings give a clear overview of the way the system works and how it should behave, which is sufficient for designing the windows and arranging the data input/output into the right locations.

## Description of measures planned for security and integrity of data

### Data Protection

The data that is stored within the system is not personal data, and therefore the data protection act does not apply. It can be transferred globally without encryption, therefore it does not need to be secured or password protected.

### User Access Rights

Because of the nature of the work that is done to the system, it is feasible that the software could contain differing levels of access rights. However, it would be more efficient for there to be one universal (administrator) level, and if data then needs to be transferred it can be copied and distributed. The system outputs will allow this to take place, rather than implementing a complex access rights system.

### Data Integrity

The data that is stored associated with the system could be corrupted if it is modified by users. This will occur in one of two ways:

- Data is edited in files that the program uses to run, and is not saved in the correct format.
- Data is edited in files that the system has output.

In the first case, the system will show an error and the user will have to correct the error in the file. Adequate support will be provided in the user manual for this. In the second case, the edits will not affect the system at runtime, but data integrity will still be compromised. However, because data is modified outside of the control of the system, it is not a problem that the system can feasibly deal with.

### Summary of Measures

The system does not require any special consideration to the security and integrity of data. While both can be compromised, the defences that the system could implement

will be of a lesser quality than the defences on PCs and networks that the system is held on. As such, any measures that I can implement are largely irrelevant.

In order to ensure that data inconsistencies do not cause hidden errors in the program, there will be format checks on input of all files. If errors are found in the format that may cause the system to fail, or function improperly, a warning will be displayed so that the user can correct the issue.

## Description of measures planned for system security

The system itself requires very little additional security. Password protection and data encryption are not required, as requested by the client.

The data that is held in the system is not of sufficient importance to require extensive security in the system – much of it is in the public domain. The data that is not in the public domain is that which is loaded during use of the system – always by employees of Red Bull Racing/Red Bull Technology – and it will come from other sources within the company.

Any user who gains access to the system could potentially gain access to sensitive data. In this event, however, much more rigorous defences will have been breached, so measures within the system will not be a significant challenge.

The system itself will have a maximum useful lifespan of a few years, and will probably require updating every few months. As a result, users that gain access to the system unlawfully will not be able to gain a significant competitive advantage over the Red Bull team through use of the system.

## Overall test strategy

### *Test data and expected results*

The system will implement a lot of features, and as a result making sure that it works correctly is a vital part of the design and development. There is a lot of reliance on data being reasonable – if an error occurs during input it can corrupt data that is generated all the way through to the race simulation, and, ultimately, invalidate this simulation.

As a result, there must be sufficient testing at each stage to make sure that the data that is presented is reasonable and correct. This will have to be done in a variety of different ways, depending on the stage of development and the complexity of the routines involved.

The test strategies that I plan to implement are described below.

### During Development

#### **Bottom Up Testing**

There are sections of the program that are programmed as individual modules, and will be tested during their development. An example of this is the routine which processes data that has been input from the PDFs – this will be programmed separately and tested with all of the PDFs that I have to ensure that they worked properly. By programming it separately, I can trace any errors through a much smaller piece of code than the whole program.

The routine will initially be programmed to select just the lap times and output them. This will prove that the system can identify lines which included a lap time and convert them into the floating-point lap time.

Then, I developed this to identify the drivers, and the system will be able to output the driver and their lap times. At this point, it should also be able to save the lap times to the drivers' individual list of lap times.

Finally, I will make sure that the program deals correctly with the 'P' pit marker, by splitting the input data into stints. The program will check these stints by getting the fastest lap, average lap, and the total time of the laps. I will output this summary data, which will be checked for a few stints against hand calculations. Once the summary data had been outputted correctly, I arranged the system to save the data to the drivers' individual list of stints, which forms the basis of the data analysis.

Similar methods of testing will be used for the system's calculation of timing data once the driver's stints had been loaded, and for testing the routine that calculates the optimum strategy.

The algorithms that are used will be, in reality, too complex and long to try and trace by hand to produce hand-calculations. Instead, I will make sure that the results are reasonable, and by testing each routine, I should be able to confirm that all of the data is correct and reliable as it passes through the system.

### **Integration Testing**

Having completed the bottom-up testing on all of the routines which handled data, implementation testing will be required when writing the interfaces to link them together.

This will be done by building the system up step-by-step and checking that the routines all used consistent terminology, variable names, and interfaced together properly. The compiler help will be utilised at this stage for identifying unassigned or unused variables, and for debugging issues when they do occur.

This implementation testing will use data sets that had been produced using previously tested sections of code. For example, the interface between loading data and processing it into the 'pace parameters' will use the PDF data. The pace parameter data will be saved and input as test data for the calculation of optimum strategies. This method used realistic test data and made sure that all routines were

correctly linked. It does not necessarily imply that the values are correct, but checks that they are reasonable.

### System Testing

Once the implementation is largely completed, I needed to fully test the system from start to finish making sure there were no unusual bugs present. For this, the test data will be the PDFs from the 2013 season, which will be sufficient to prove that the system can work in all reasonable situations.

### **White Box Testing**

Due to the limited selection of data that could feasibly be input, it would be reasonable to assume that by testing all of the available test data (one season's worth of files), will run every required route through the system. This can be classed as 'white box testing'.

To test this, I will run the system for every PDF file from the 2013 season, and make sure that the system will produce a race simulation from this data. I will then take one specific data set and test all of the buttons and inputs on the strategy form, and the same for the race form. Finally, I will make sure that the system can load data from the timing archives for every race from the season.

This collection of test data will contribute to running every necessary piece of code in the system. There will be boundary and erroneous tests, but the system would be expected to fail in the erroneous tests because it was not efficient to implement the validation to prevent this. Otherwise, the system should continue to produce a race simulation, although it may not be completely accurate.

### **Black Box Testing**

The final tests before the system is sent to the user for their own acceptance testing will ensure that the program meets the specification. At this stage, all of the specification points will have tests devised to make sure that the program meets them effectively, and then the tests will be run. The results will be compared with the expected test data and if the system passes the tests it will be passed on to the client.

The complete tests will be listed in the testing section as the test data and expected results will depend on the way that the system is implemented.

### Test Datasets

#### **Strategy Optimisation**

To make sure that the strategy optimisation works correctly, I have written the algorithm to optimise a strategy for given parameters in Microsoft Excel. This has allowed me to produce test data for two purposes.

Firstly, I can check that the correct stint lengths are selected given the input parameters for the strategies. The system will calculate the optimum lengths

depending on the pace parameters and number of stints. The program will do this for every permutation of numbers of stints.

Secondly, the system will choose the strategy which takes the least amount of time, and make it the default strategy for the given driver. The excel program will also calculate the time required to complete a strategy, thus confirming that the system chooses the strategy that takes the least amount of time.

The datasets are below. These are copied straight from the excel system. The optimum strategy is a two-stop strategy with two prime stints of 23 laps, and one option stint of 12 laps. This will be checked against the system's output during the testing phase.

Data Set	Prime Stints	Option Stints	Ideal Prime Laps	Act Prime Laps	Act Option Laps	Deg Time	Pit Time	Fuel Time	Total Added Time	Race Time
1	1	1	41.25	41	17	109.2	21	85.55	215.75	11862.15
2	1	2	31.20	31	13.5	75.525	42	85.55	203.075	11849.48
3	2	1	23.57	23	12	60.8	42	85.55	188.35	11834.75
4	2	2	19.50	19	10	45.2	63	85.55	193.75	11840.15
5	1	3	24.50	25	11	53.1	63	85.55	201.65	11848.05
6	3	1	16.50	16	10	41.5	63	85.55	190.05	11836.45

## Implementation

### Index of Complex Code Structures

The table below gives an overview of the location of complex features within the program. All of the pages given are examples; all of the features can be found multiple times within the program.

Feature	Feature	Page	Line
Indentation of code, code spacing, comments	See All		
Using local variables	GetFileName	116	
Use of meaningful variable names	Fields for strategy class	162-163	
Use of constants	Control spacing variables in PaceParameters class	248	
Selection + iteration statements:	Switch If For Foreach While	259 109 132 105 135	7 17, 20, 35 2, 4 21, 30 2, 5, 11
Procedures + functions	See All		
Parameter passing	Value Reference Classes by pointer Out	105 157 169 123	7 35 40 36
User defined records	Structures Settings Class <sup>3</sup>	141 149	46 5
Arrays	1D 2D List Dictionary	248 248 104 170	13, 14, 16, 23 24, 26 19 35
File handling routines	Reading Writing Creating Directories Error Handling	297 297 119 306	65 21 22 1
Validation	Calls Routines	254 137	30 17
Modules	See namespaces throughout		
Regular Expressions (REGEX)	Not required by specification		
Encryption	Not required by specification		
Search routines	ConvertToDriverIndex <sup>4</sup>	114	16

---

<sup>3</sup> The settings class is an extended record structure with methods for reading and writing that structure to and from file.

Feature	Feature	Page	Line
	Binary Search	136	38
Sort routines	Quicksort	134	51
	Miniature insert-sort	137	55
	Insertion Sort	135	47
Recursion	Quicksort	134	51
	Binary Search	136	38
Graphics and drawing routines	CumulativeTimeGraph	255	65
Linking to a database (or other external software)	Writing to csv	197	17
	Loading from database	123	41
Dynamic objects	MyPanel <sup>5</sup>	200	54
Advanced controls e.g. ListView, Panel	Panel	200	54
	List Box	273	8
	Toolbar	207	31
	Right-click menu	198	8
Other data structures e.g. graphs, trees, queues, stacks	Line	174	45
	Graph axis	255	48
Object oriented programming	Classes – see all		
	Interface	291	14
	Inheritance – see diagram		
	Overriding	213	17
Bespoke processing/ calculation routines:	OptimisePrime <sup>6</sup>	158	64
	CalculateOvertake <sup>7</sup>	144	48
	SimulateRace <sup>8</sup>	142	57
	GetNormalisedTraces <sup>9</sup>	259, 260	35, 22
Other complexity:	Generic Type Function	134	51
	Operator overloading	151	23, 29
	General overloading	123	1, 18
	Click-drag panel locating	170	29
	Dynamic panel ordering	221	39
	Text processing	109	4, 57
	Lambda Expressions	210	1, 7, 13, etc.
	Extension Methods	133	8
	Custom Event Args	280	57

<sup>4</sup> ConvertToDriverIndex is a routine that finds a driver index based on the race number of that driver (race numbers cannot be sorted as they are a property of the driver class and we require the array index of the driver).

<sup>5</sup> MyPanel and all subclasses are entirely dynamically generated.

<sup>6</sup> OptimisePrime uses an algorithm generated by calculus to pick the optimum prime stint length.

<sup>7</sup> CalculateOvertake uses a probabilistic model to work out if an overtake will occur between two cars.

<sup>8</sup> SimulateRace runs a lap-based simulation of all cars in a race situation.

<sup>9</sup> GetNormalisedTraces normalises a cumulative time line based on either an average time or a different trace. It allows cars to cycle on the graph when they are lapped, showing their true track position.

## Program listing

### StratSim

```

using StratSim.Model;
using StratSim.Model.Files;
using StratSim.View.Forms;
using StratSim.View.MyFlowLayout;
using StratSim.View.Panels;
using StratSim.ViewModel;
using System;
using System.Collections.Generic;
using System.Windows.Forms;

namespace StratSim
{
    public static class Program
    {
        /// <summary>
        /// The main entry point for the application.
        /// </summary>

        static List<MainForm> mainForms;

        public static VersionWindow VersionInformation;

        public static InfoPanel InfoPanel;
        public static SettingsPanel SettingsPanel;

        public static NewStartPanel NewStartPanel;
        public static DataInput DataInput;
        public static CumulativeTimeGraph Graph;
        public static StrategyViewer StrategyViewer;
        public static PaceParameters TimeAnalysis;
        public static DriverSelectPanel DriverSelectPanel;
        public static ContentTabControl ContentTabControl;
        public static AxesWindow AxesWindow;
        public static TimingArchives TimingArchives;

        static Data myModel;
        static MyEvents events;

        static int numberOfforms;

        [STAThread]
        static void Main()
        {
            Application.EnableVisualStyles();
            Application.SetCompatibleTextRenderingDefault(false);
            SetupStaticClasses();
            mainForms.Add(new MainForm(numberOfforms++));

            Application.Run(mainForms[0]);
        }

        static void SetupStaticClasses()
        {
            myModel = new Data();
            events = new MyEvents();
            mainForms = new List<MainForm>();
        }

        public static void ShowVersionInfo()
        {
            VersionInformation = new VersionWindow();
            VersionInformation.Show();
        }
    }
}

```

```

}

/// <summary>
/// Removes a form from the list of forms when it is closed.
/// </summary>
/// <param name="formIndex">The index of the form that is closed.</param>
public static void FormClosed(int formIndex)
{
    mainForms.RemoveAt(formIndex);
    numberOForms--;

    if (numberOForms == 0)
    {
        Exit();
    }
}

public static void AddEventToForms(MainForm.PanelDroppedEventHandler
panelDroppedEventHandler)
{
    foreach (MainForm f in mainForms)
    {
        f.PanelDroppedOnForm += panelDroppedEventHandler;
    }
}

public static void RemoveEventFromForms(MainForm.PanelDroppedEventHandler
panelDroppedEventHandler)
{
    foreach (MainForm f in mainForms)
    {
        f.PanelDroppedOnForm -= panelDroppedEventHandler;
    }
}

/// <summary>
/// Opens a panel in a new window
/// </summary>
/// <param name="PanelToAddToNewWindow">The panel to open in a new
window</param>
/// <param name="Form">The original form on which it was displayed</param>
public static void OpenInNewWindow(MyPanel PanelToAddToNewWindow, MainForm
Form)
{
    mainForms.Add(new MainForm(numberOForms));
    Form.IOController.RemovePanel(PanelToAddToNewWindow);
    mainForms[numberOForms].IOController.AddPanel(PanelToAddToNewWindow);
    PanelToAddToNewWindow.OnOpenedInDifferentForm(mainForms[numberOForms]);
    mainForms[numberOForms].Show();
    numberOForms++;
}

/// <summary>
/// Loads driver timing data from the saved files automatically.
/// </summary>
/// <param name="raceIndex">The index of the race data to load</param>
public static void PopulateDriverDataFromFiles(int raceIndex)
{
    Program.InfoPanel.WriteData("Commencing quick start");
    Data.RaceIndex = raceIndex;

    for (int sessionIndex = 0; sessionIndex <= 3; sessionIndex++)
    {
        var d = new LapData();
        Data.SessionIndex = sessionIndex;
        d.RetrieveArchiveData(sessionIndex, raceIndex);
        Data.sessionDataLoaded[sessionIndex] = true;
    }
}

```

```
    }

    var s = new SpeedData();
    s.RetrieveArchiveData(3, raceIndex);
    Data.sessionDataLoaded[4] = true;

    Program.InfoPanel.WriteData("Data loaded and ready for analysis");
}

public static void Exit()
{
    Application.Exit();
}
}
```

StratSim.Model.Files

```

namespace StratSim.Model.Files
{
    /// <summary>
    /// Controls the loading of data from a file and the processing of the data
    /// in the file
    /// </summary>
    class DataController
    {
        ISessionData dataToAnalyse;

        public DataController()
        { }

        /// <summary>
        /// Controls the processing of data loaded from a timing data file.
        /// </summary>
        /// <param name="data">The complete file contents</param>
        /// <param name="fileType">The type of data contained by the file</param>
        public void ProcessData(string data, TimingDataType fileType)
        {
            Program.InfoPanel.WriteData("Data Loaded");
            dataToAnalyse = GetdataType(data, fileType);

            dataToAnalyse.AnalyseData();
            dataToAnalyse.WriteArchiveData();

            Data.sessionDataLoaded[Data.SessionIndex + (int)fileType] = true;
            Program.InfoPanel.WriteData("Data from " +
Data.SessionNames[Data.SessionIndex] + " processed.");
            Program.DataInput.ResetPanel(Data.RaceIndex, Data.SessionIndex + 1);
        }

        /// <summary>
        /// Gets the type of data that is to be loaded from the given file type
        /// </summary>
        /// <param name="data">The loaded data string that will be used to generate
        information</param>
        /// <param name="fileType">The type of file that was loaded</param>
        /// <returns>A populated instance of the correct class for the type of data
        that was passed</returns>
        ISessionData GetdataType(string data, TimingDataType fileType)
        {
            ISessionData dataClassToReturn;

            if (fileType == TimingDataType.LapTimeData)
            {
                dataClassToReturn = new LapData(data);
            }
            else
            {
                dataClassToReturn = new SpeedData(data);
            }

            return dataClassToReturn;
        }

        public TimingData ThisData
        {
            get { return (TimingData)dataToAnalyse; }
            set { dataToAnalyse = (ISessionData)value; }
        }
    }
}

```

```

using StratSim.ViewModel;

namespace StratSim.Model.Files
{
    /// <summary>
    /// Specifies that the class has methods for interpreting and writing data
    /// loaded from PDF files
    /// </summary>
    public interface ISessionData : IFileController<string[]>
    {
        void AnalyseData();
        void WriteArchiveData();
        void RetrieveArchiveData(int sessionNumber, int raceNumber);
    }
}

using System;
using System.IO;

namespace StratSim.Model.Files
{
    /// <summary>
    /// Processes timing data about the lap times of drivers from a PDF file.
    /// </summary>
    public class LapData : TimingData, ISessionData
    {
        private string lapDataAsString;
        private string[] lapDataAsArray;

        public LapData(string passedData)
        {
            lapDataAsString = passedData;
            lapDataAsArray = ConvertToArray(lapDataAsString);
        }

        public LapData()
        {
        }

        public void AnalyseData()
        {
            CollectAndProcessData(Data.SessionIndex);
        }

        public void WriteArchiveData()
        {
            string filePath = GetTimingDataDirectory(Data.RaceIndex) +
                GetFileName(Data.SessionIndex, Data.RaceIndex, TimingDataType.LapTimeData);
            WriteToFile(filePath);
        }

        public void RetrieveArchiveData(int sessionNumber, int raceNumber)
        {
            string filePath = GetTimingDataDirectory(raceNumber) +
                GetFileName(sessionNumber, raceNumber, TimingDataType.LapTimeData);
            ReadFromFile(filePath);
        }

        public string[] FileData
        {
            get
            {
                return lapDataAsArray;
            }
        }
    }
}

```

```

        }

    public void CollectAndProcessData(int sessionIndex)
    {
        string lineInput = "";
        int lineNumber = 0;
        int lapNo = 0; //number of laps in driver's stint
        int lineStatus = 0; //0 is no numbers read, 1 is reading, 2 is finished
reading
        int driverIndex = -1;
        int readStatus = 0;

        Stint tempStint;
        do //terminates when file ends.
        {
            if (lineStatus == 0)
                lineInput = GetNextLineToInput(lineNumber++, FileData);

            if (NameCheck(lineInput))
            {
                lapNo = 0;
                lineStatus = 0;
                ++driverIndex;

Data.Drivers[driverIndex].RaceWeekendSessionStints[sessionIndex].Clear();
tempStint = new Stint(Data.Drivers[driverIndex], sessionIndex,
lapNo);
readStatus = 1;

            do
            {

                lineInput = GetNextLineToInput(lineNumber++, FileData);
                if (GetNumber(lineInput, 0) == lapNo++)
                {
                    UseTimeData(lineInput, lapNo - 1, driverIndex, sessionIndex,
ref readStatus, ref tempStint);
                }
                else { lineStatus++; }
            } while (lineStatus != 2); //exits when times are finished

            if (tempStint.lapTimes.Count != 0) {
Data.Drivers[driverIndex].RaceWeekendSessionStints[Data.SessionIndex] += tempStint;
}

            else
            {
                if (lineStatus == 2)
                    lineInput = GetNextLineToInput(lineNumber++, FileData);
            }
//end if

        } while (lineInput != null);
    }

    void UseTimeData(string time, int lapNo, int driverIndex, int sessionIndex,
ref int readStatus, ref Stint tempStint)
{
    float lapTime;
//readStatus - 0: waiting for start, 1: out-lap, 2: add lap to stint.

    time = time.Remove(0, (lapNo >= 10 ? 3 : 2)); //takes away the lap number
    if (time[0] == 'P')
    {
        time = time.Remove(0, 2); //removes the 'P'
}
}

```

```

        if (readStatus == 2)
        {
            //pass old stint
            Data.Drivers[driverIndex].RaceWeekendSessionStints[sessionIndex] += tempStint;

            //start new tempStint
            tempStint = new Stint(Data.Drivers[driverIndex], sessionIndex,
lapNo);
        }

        //get ready to read new stint
        readStatus = 0;
    }

    //load the lap time from a string
    lapTime = GetLapTime(time);

    if (readStatus == 2) //if reading
    {
        //adds lap time to stint
        tempStint += lapTime;
    }
    else
    {
        //increments readStatus if not already reading
        readStatus++;
    }
}

public void WriteToFile(string fileName)
{
    string lineInput = "";
    int lineNumber = 0;
    int lapNumber = 1;
    int lapType = 2;
    int driverIndex = 0;
    float lapTime = 0;

    StreamWriter w = new StreamWriter(fileName);

    while ((lineInput = GetNextLineToInput(lineNumber++, FileData)) != null)
        //While there is data to read
    {
        if (NameCheck(lineInput)) //If a driver's name is found
        {
            lapNumber = 1;
            lapType = 2;
            w.WriteLine(Data.Drivers[driverIndex++].DriverName);
        }

        else //reading times
        {
            if (GetNumber(lineInput, 0) == lapNumber)
            {
                lineInput = lineInput.Remove(0, (lapNumber >= 10 ? 3 : 2));
                if (lineInput[0] == 'P')
                {
                    lineInput = lineInput.Remove(0, 2);
                    lapType = 2;
                }
                lapTime = GetLapTime(lineInput);
                lapNumber++;
            }

            try
            {

```

```

        w.WriteLine(lapTime);
        w.Write(",");
        w.WriteLine(lapType);
    }
    catch
    {
        w.WriteLine("0,2");
    }

    if (lapType == 2) //if in-lap, cycle the lap status so the next
lap is an out-lap
        { lapType = 0; }
        else
        { lapType = 1; }
    }
}
w.Dispose();
}

public void ReadFromFile(string fileName)
{
    if (File.Exists(fileName))
    {
        string line = "";
        string[] lineParts;
        float lapTime = 0;
        int lapType = 0;
        int lapNumber = 0;

        int driverIndex = -1;
        Stint tempStint = null;

        StreamReader r = new StreamReader(fileName);

        do
        {
            line = r.ReadLine();

            if (IsDriverName(line))
            {
                if (driverIndex >= 0 && tempStint.lapTimes.Count > 0) {
Data.Drivers[driverIndex].RaceWeekendSessionStints[Data.SessionIndex] += tempStint;
}
                driverIndex++;
                lapNumber = 0;

                Data.Drivers[driverIndex].RaceWeekendSessionStints[Data.SessionIndex].Clear();
                tempStint = new Stint(Data.Drivers[driverIndex],
Data.SessionIndex, lapNumber);
            }
            else
            {
                lineParts = line.Split(',');
                if (lineParts.Length == 2)
                {
                    if (lineParts[0] != "") 
lapTime = float.Parse(lineParts[0]);
                    if (lineParts[1] != "") 
lapType = int.Parse(lineParts[1]);
                }
            }
        }
        lapNumber++;

        if (lapType == 1)
tempStint += lapTime;
    }
}

```

```

        if (lapType == 2 && tempStint.lapTimes.Count > 0)
        {

            Data.Drivers[driverIndex].RaceWeekendSessionStints[Data.SessionIndex] += tempStint;
            tempStint = new Stint(Data.Drivers[driverIndex],
Data.SessionIndex, lapNumber);
        }
    }

} while (!r.EndOfStream);
r.Dispose();
}

/// <summary>Gets a lap time from a specified string</summary>
/// <param name="time">A string with the time in mm:ss:000 format</param>
/// <returns>The number of seconds represented by the time</returns>
public float GetLapTime(string time)
{
    float lapTime = 0;
    int decimalPlaces = 3;

    if (time[1] != ':')
    {
        lapTime += CovertCharToInt(time[0]) * 600; //counts tens of minutes if they are present
        time = time.Remove(0, 1); //removes tens of minutes
        decimalPlaces = 2;
    }

    //add minutes, 10 x seconds, seconds
    lapTime += CovertCharToInt(time[0]) * 60;
    lapTime += CovertCharToInt(time[2]) * 10;
    lapTime += CovertCharToInt(time[3]);

    //add thousandths
    for (int characterIndex = 1; characterIndex <= decimalPlaces;
characterIndex++)
    {
        try
        { lapTime += CovertCharToInt(time[4 + characterIndex]) *
(float)Math.Pow(10, -characterIndex); }
        catch
        { }
    }

    return lapTime;
}
}

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.IO;

namespace StratSim.Model.Files
{
    /// <summary>
    /// Processes data about driver top speeds from a PDF file
    /// </summary>

```

```

public class SpeedData : TimingData, ISessionData
{
    private string speedDataAsString;
    private string[] speedDataAsArray;

    public SpeedData(string passedData)
    {
        speedDataAsString = passedData;
        speedDataAsArray = ConvertToArray(speedDataAsString);
    }

    public SpeedData()
    {
    }

    public void AnalyseData()
    {
        CollectAndProcessData();
    }

    public void WriteArchiveData()
    {
        string filePath = GetTimingDataDirectory(Data.RaceIndex) +
GetFileName(Data.SessionIndex, Data.RaceIndex, TimingDataType.SpeedData);
        WriteToFile(filePath);
    }

    public void RetrieveArchiveData(int sessionNumber, int raceNumber)
    {
        string filePath = GetTimingDataDirectory(raceNumber) +
GetFileName(sessionNumber, raceNumber, TimingDataType.SpeedData);
        ReadFromFile(filePath);
    }

    public string[] FileData
    {
        get
        {
            return speedDataAsArray;
        }
        set
        {
            speedDataAsArray = value;
        }
    }

    public void CollectAndProcessData()
    {
        string lineInput = "";
        int lineNumber = 0;
        int rank = 0;
        int processedDrivers = 1;
        int driverNumber, driverIndex;
        bool readData = true;
        float driverTopSpeed;

        lineInput = GetNextLineToInput(lineNumber++, FileData);
        do
        {
            driverTopSpeed = Data.Settings.DefaultTopSpeed;
            readData = true;

            rank = GetNumber(lineInput, 0);

            //check ranking against processed drivers
            if (rank != processedDrivers)

```

```

        { readData = false; }
    else
    {
        //remove the driver's ranking.
        lineInput = lineInput.Remove(0, (processedDrivers >= 10 ? 3 : 2));

        processedDrivers++;

        //check the driver's name.
        readData = NameCheck(lineInput);
    }

    driverNumber = GetNumber(lineInput, 0);
    driverIndex = Driver.ConvertToDriverIndex(driverNumber);

    if (readData)
    {
        driverTopSpeed = AssignDriverSpeedData(lineInput, driverIndex);
        Data.Drivers[driverIndex].TopSpeed = driverTopSpeed;
    }

} while ((lineInput = GetNextLineToInput(lineNumber++, FileData)) != null);
}

float AssignDriverSpeedData(string speed, int driverIndex)
{
    float driverTopSpeed = 0;
    int decimalPointPosition = 0;
    int characterIndex = 0;

    speed = speed.Remove(0, Data.Drivers[driverIndex].DriverName.Length +
(driverIndex >= 9 ? 6 : 5) + 1);
    speed = speed.Substring(0, 5);

    decimalPointPosition = speed.LastIndexOf('.');
    for (int orderOfMagnitude = decimalPointPosition - 1; orderOfMagnitude >=
0; --orderOfMagnitude)
    {
        //Sums the orders of magnitude of the top speed
        driverTopSpeed += CovertCharToInt(speed[characterIndex++]) *
(float)Math.Pow(10, orderOfMagnitude);
    }
    //Adds the value of the first decimal point
    driverTopSpeed += (float)(CovertCharToInt(speed[decimalPointPosition + 1]) *
0.1);

    return driverTopSpeed;
}

public void WriteToFile(string fileName)
{
    string lineInput = "";
    int lineNumber = 0;
    bool readData;
    int rank = 0;
    int processedDrivers = 1;
    int driverNumber, driverIndex;
    float topSpeed = Data.Settings.DefaultTopSpeed;

    StreamWriter w = new StreamWriter(fileName);

    while ((lineInput = GetNextLineToInput(lineNumber++, FileData)) != null)
    {
        readData = true;

        rank = GetNumber(lineInput, 0);

```

```

//check ranking against processed drivers
if (rank != processedDrivers)
{ readData = false; }
else
{
    //remove the driver's ranking.
    lineInput = lineInput.Remove(0, (processedDrivers >= 10 ? 3 : 2));

    processedDrivers++;

    //check the driver's name.
    readData = NameCheck(lineInput);
}

driverNumber = GetNumber(lineInput, 0);
driverIndex = Driver.ConvertToDriverIndex(driverNumber);

if (readData)
{
    topSpeed = AssignDriverSpeedData(lineInput, driverIndex);

    w.Write(driverIndex);
    w.Write(",");
    w.Write(Data.Drivers[driverIndex].DriverName);
    w.Write(",");
    w.WriteLine(topSpeed);
}
}

w.Dispose();
}

public void ReadFromFile(string fileName)
{
    if (File.Exists(fileName))
    {
        string line = "";

        StreamReader r = new StreamReader(fileName);

        do
        {
            line = r.ReadLine();
            SetDriverSpeed(line);
        } while (!r.EndOfStream);
    }
}

public void SetDriverSpeed(string lineContainingDataFromFile)
{
    string[] lineParts;

    float driverSpeed = Data.Settings.DefaultTopSpeed;
    int driverIndex = 0;

    lineParts = lineContainingDataFromFile.Split(',');
    driverSpeed = float.Parse(lineParts[2]);
    driverIndex = int.Parse(lineParts[0]);

    Data.Drivers[driverIndex].TopSpeed = driverSpeed;
}
}
}

```

```

using System.Collections.Generic;
using System.IO;

namespace StratSim.Model.Files
{
    public enum TimingDataType {LapTimeData = 0, SpeedData = 1}

    /// <summary>
    /// Base class for types of timing data processed by the system
    /// </summary>
    public class TimingData
    {
        public TimingData()
        {}

        /// <summary>
        /// Gets the full file name with extension for the given session, track, and
        data type
        /// </summary>
        /// <param name="sessionNumber">The session to find the file for</param>
        /// <param name="raceIndex">The index of the race to find the file for</param>
        /// <param name="dataType">The type of data being processed</param>
        /// <returns>The complete file name with extension for locating the PDF file
        containing the required data.</returns>
        public string GetFileName(int sessionNumber, int raceIndex, TimingDataType
        dataType)
        {

            string fileName = "";

            fileName += Data.Tracks[raceIndex].name;
            fileName += " ";
            fileName += Data.SessionNames[sessionNumber];
            fileName += " ";

            if (dataType == TimingDataType.LapTimeData)
            {
                fileName += "Lap Times.txt";
            }
            else
            {
                fileName += "Speed Trap.txt";
            }

            return fileName;
        }

        /// <summary>
        /// Converts a character value to an integer
        /// </summary>
        /// <param name="c">The character to be changed</param>
        /// <returns>An integer value that is the same as the textual representation
        held by the character</returns>
        public static int CovertCharToInt(char c)
        {
            return ((int)c - 48);
        }

        /// <summary>
        /// Gets a driver number from a line of text containing the driver number.
        /// </summary>
        /// <param name="line">The string containing the number</param>
        /// <param name="startPosition">The start position of the number in the
        line</param>
        /// <returns>The driver number as an integer from the line</returns>
        public static int GetNumber(string line, int startPosition)
        {
    
```

```

        int number = 0;

    try
    {
        if (line[startPosition + 1] != ' ')
        {
            number += CovertCharToInt(line[startPosition]) * 10;
            line = line.Remove(startPosition, 1); //takes out first character
        }

        number += (CovertCharToInt(line[startPosition]));

        return number;
    }
    catch
    {
        return -1;
    }
}

/// <summary>
/// Converts data from a string including line spacing characters to an array
of each line.
/// </summary>
/// <param name="data">The string of data to be converted to a string</param>
/// <returns>An array of strings representing each line</returns>
public static string[] ConvertToArray(string data)
{
    List<string> temp = new List<string>();
    string[] arrayToReturn;
    string stringToAdd = "";
    int charsProcessed = 0;

    do
    {
        if (data[charsProcessed] == '\r')
        {
            if (data[charsProcessed + 1] == '\n') //If a line break has been
found
            {
                temp.Add(stringToAdd);
                stringToAdd = "";
                charsProcessed += 1;
            }
        }
        else { stringToAdd += data[charsProcessed]; }
    } while (++charsProcessed <= data.Length - 1);

    //Adds the strings to a temporary list
    temp.Add(stringToAdd);

    //sets up an array of the strings
    arrayToReturn = new string[temp.Count + 1];
    int stringIndex = 0;

    //populates the array of strings
    foreach (string s in temp)
    {
        arrayToReturn[stringIndex++] = s;
    }

    return arrayToReturn;
}

/// <summary>

```

```

    ///> Checks if the line contains a driver's name and number, and if it is a
    ///> valid combination
    ///> </summary>
    ///> <param name="testLine">The line to check for names</param>
    ///> <returns>True if the line contains a driver name and matching
    number</returns>
    public static bool NameCheck(string testLine)
    {
        int driverIndex, driverNumber;
        int startOfName, nameLength;
        string driverName = "";

        try
        {
            //gets the number of the driver. Stored as string so converts from ascii
            driverNumber = GetNumber(testLine, 0);
            driverIndex = Driver.ConvertToDriverIndex(driverNumber);

            if (driverIndex < 0 || driverIndex >= Data.NumberOfDrivers) { return
false; }
        }
        catch { return false; }

        startOfName = (driverNumber >= 10 ? 6 : 5);
        nameLength = Data.Drivers[driverIndex].DriverName.Length;

        if (testLine.Length >= (startOfName + nameLength))
        {
            driverName = testLine.Substring(startOfName, nameLength);
        }

        try { if (driverName != Data.Drivers[driverIndex].DriverName) { return
false; } }
        catch { return false; }

        return true;
    }

    ///> Checks if the specified value is a valid driver name
    ///> </summary>
    ///> <param name="name">The name to test</param>
    ///> <returns>True if the name is valid</returns>
    public bool IsDriverName(string name)
    {
        bool isName = false;

        foreach (Driver d in Data.Drivers)
        {
            if (name == d.DriverName)
                isName = true;
        }

        return isName;
    }

    ///> Gets a single string line from a string array of data
    ///> </summary>
    ///> <param name="lineNumber">The array index to retrieve</param>
    ///> <param name="fileData">The array to retrieve data from</param>
    ///> <returns>Empty string if the array index is out of range</returns>
    public string GetNextLineToInput(int lineNumber, string[] fileData)
    {
        string lineToReturn;

        try

```

```
{  
    lineToReturn = fileData[lineNumber];  
    return lineToReturn;  
}  
catch  
{  
    return "";  
}  
}  
  
/// <returns>The file path of the directory containing the required  
file</returns>  
public string GetTimingDataDirectory(int raceIndex)  
{  
    string directoryName = Data.Tracks[raceIndex].name;  
  
    string baseDirectory = Data.Settings.DataFilePath;  
    string dataPath = baseDirectory + "/RaceData/";  
    dataPath += directoryName;  
  
    if (!Directory.Exists(dataPath)) { Directory.CreateDirectory(dataPath); }  
  
    dataPath += "/";  
  
    return dataPath;  
}  
}  
}  
}
```

StratSim.Model

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

using StratSim.View.Panels;

namespace StratSim.Model
{
    /// <summary>
    /// Enumerated options for the type of tyre used by a driver
    /// </summary>
    public enum TyreType { Prime, Option }

    /// <summary>
    /// class containing static data elements used within the program
    /// </summary>
    public class Data
    {
        /// <summary>
        /// Initialises the data class and initialises elements within it.
        /// </summary>
        public Data()
        {
            settings = new Settings();
            drivers = Driver.InitialiseDrivers(out NumberOfDrivers);
            tracks = Track.InitialiseTracks(out NumberOfTracks);
        }

        /// <summary>
        /// The number of drivers registered to take part in a race
        /// </summary>
        public static int NumberOfDrivers;
        /// <summary>
        /// The number of races in a season
        /// </summary>
        public static int NumberOfTracks;

        static string[] sessionNames = new string[5] { "First Practice Session",
"Second Practice Session", "Third Practice Session", "Qualifying", "Race" };
        static Driver[] drivers;
        static Track[] tracks;
        static Settings settings;
        static Race race;

        static int sessionNumber = 0;
        static int raceIndex = 0;
        static int driverIndex = 0;
        public static bool[] sessionDataLoaded = new bool[5];//0-3, 4 is speedData.

        /// <returns>The number of laps run in the race denoted by the current race
index</returns>
        public static int GetRaceLaps()
        {
            return tracks[raceIndex].laps;
        }

        /// <summary>
        /// Gets an array representing the names given to the sessions of the race
weekend.
        /// </summary>
        public static string[] SessionNames
        { get { return sessionNames; } }
    }
}

```

```

    ///<summary>
    /// Gets or sets an array of the drivers currently taking part in an event.
    ///</summary>
    public static Driver[] Drivers
    {
        get{ return drivers; }
        set {drivers = value;}
    }
    ///<summary>
    /// Gets or sets an array of the race tracks currently loaded
    ///</summary>
    public static Track[] Tracks
    {
        get{ return tracks; }
        set { tracks = value; }
    }
    ///<summary>
    /// Gets or sets a collection of data used for generic settings and default
    information.
    ///</summary>
    public static Settings Settings
    {
        get { return settings; }
        set{ settings = value;}
    }
    ///<summary>
    /// Gets or sets the race simulation used in the program.
    ///</summary>
    public static Race Race
    {
        get{ return race; }
        set{ race = value;}
    }

    ///<summary>
    /// Gets or sets the current race index. Setting the race index shows track
    information
    /// in the info panel and clears all timing data from the driver classes.
    ///</summary>
    public static int RaceIndex
    {
        get{ return raceIndex; }
        set
        {
            raceIndex = value;
            Program.InfoPanel.ShowTrackInfo = true;
            foreach (Driver d in drivers)
            {
                d.ClearSessions();
            }
        }
    }
    ///<summary>
    /// Gets the track represented by the current race index.
    ///</summary>
    public static Track CurrentTrack
    { get{ return tracks[raceIndex];}}
    ///<summary>
    /// Gets or sets the current driver index. This is the normalised driver on
    any graph
    /// and the driver whose details are displayed in the info panel.
    ///</summary>
    public static int DriverIndex
    {
        get { return driverIndex; }
        set {
            driverIndex = value;
    }
}

```

```

        Program.InfoPanel.ShowDriverInfo = true;
    }
}
public static int SessionIndex
{
    get { return sessionNumber; }
    set { sessionNumber = value; }
}
/// <summary>
/// Gets the driver represented by the current driver index.
/// </summary>
public static Driver CurrentDriver
{ get { return drivers[driverIndex]; } }
}

using System;
using System.Collections.Generic;
using System.Drawing;
using System.Linq;

namespace StratSim.Model
{
    /// <summary>
    /// Contains fields representing data about how the driver will perform in a
    race,
    /// and methods for calculating this data from raw lap times.
    /// </summary>
    public class Driver
    {
        //properties
        public int DriverNumber;
        public int DriverIndex;
        public string Team;
        public string DriverName;
        public List<Stint>[] RaceWeekendSessionStints;

        public Strategy SelectedStrategy;
        public Color LineColour;

        public float PitStopLoss;
        public float TopSpeed;
        public float TyrePaceDelta;
        Dictionary<TyreType, float> tyreDegradation = new Dictionary<TyreType,
float>(2);
        public float LowFuelPace; //low fuel lap time
        public float FuelEffectPerKilo;
        public float FuelConsumptionPerLap;
        public float FuelLoadInP2;

        public Dictionary<TyreType, float> TyreDegradation
        {
            get { return tyreDegradation; }
            set { tyreDegradation = value; }
        }

        /// <summary>
        /// Creates a new instance of the Driver class
        /// </summary>
        /// <param name="passDriverIndex">The index of the driver to be
        created</param>
        /// <param name="properties">A string array containing the driver properties,
        loaded
        /// from the driver text file.</param>
    }
}

```

```

    public Driver(int passDriverIndex, string driverName, string driverTeam, int
number, Color passColour)
{
    DriverIndex = passDriverIndex;

    DriverName = driverName;
    DriverNumber = number;
    Team = driverTeam;
    LineColour = passColour;
    FuelLoadInP2 = Data.Settings.DefaultP2Fuel;
    InitialiseSessions();
}

/// <summary>
/// Initialises a driver with specified parameters.
/// </summary>
/// <param name="parameters">An array of 8 parameters to be loaded</param>
public Driver(float[] parameters)
{
    TopSpeed = parameters[0];
    TyrePaceDelta = parameters[1];
    tyreDegradation[TyreType.Prime] = parameters[2];
    tyreDegradation[TyreType.Option] = parameters[3];
    LowFuelPace = parameters[4];
    FuelEffectPerKilo = parameters[5];
    FuelConsumptionPerLap = parameters[6];
    FuelLoadInP2 = parameters[7];
}

/// <summary>
/// Loads all drivers from the database
/// </summary>
/// <param name="numberOfDriversInSeason">A variable to which the number of
drivers found is assigned</param>
/// <returns>The populated array of drivers</returns>
public static Driver[] InitialiseDrivers(out int numberOfDriversInSeason)
{
    Driver[] arrayOfDrivers;
    int numberOfDrivers;

    var stratSimDataSet = new StratSimDataSet();
    var driverTableAdapter = new
StratSimDataSetTableAdapters.DriversTableAdapter();
    driverTableAdapter.Fill(stratSimDataSet.Drivers);
    StratSimDataSet.DriversRow row;

    numberOfDrivers = stratSimDataSet.Drivers.Count;
    arrayOfDrivers = new Driver[numberOfDrivers];

    string name, teamName;
    int driverNumber;
    Color lineColour;

    for (int driverIndex = 0; driverIndex < numberOfDrivers; driverIndex++)
    {
        row = stratSimDataSet.Drivers[driverIndex];
        name = row.DriverName;
        teamName = row.Team;
        driverNumber = row.DriverNumber;
        lineColour = Color.FromName(row.LineColour);
        arrayOfDrivers[driverIndex] = new Driver(driverIndex, name, teamName,
driverNumber, lineColour);
    }

    numberOfDriversInSeason = numberOfDrivers;
    return arrayOfDrivers;
}

```

```

void InitialiseSessions()
{
    RaceWeekendSessionStints = new List<Stint>[5];
    for (int sessionIndex = 0; sessionIndex <= 4; sessionIndex++)
    {
        RaceWeekendSessionStints[sessionIndex] = new List<Stint>();
    }
}

/// <summary>
/// Converts a driver number to the corresponding driver index
/// </summary>
/// <param name="driverNumber">The driver number to be found</param>
/// <returns>The index of the driver who currently holds the driver number.
/// Returns -1 if the driver number does not exist.</returns>
public static int ConvertToDriverIndex(int driverNumber)
{
    int driverIndex = 0;
    bool exitLoop = false;
    do
    {
        exitLoop = (Data.Drivers[driverIndex++].DriverNumber == driverNumber);

        if (driverIndex >= Data.NumberOfDrivers) //driver number not found
        {
            exitLoop = true;
            driverIndex = 0;
        }
    }
    while (!exitLoop);

    return driverIndex - 1;
}

/// <summary>
/// Gets the difference between the prime and option tyre pace for this
driver.
/// </summary>
/// <returns>The fastest option tyre lap - fastest prime tyre lap across all
sessions</returns>
public float GetTyreDelta()
{
    float[] sessionFastestLap = new float[2];
    float stintFastestLap = Data.Settings.DefaultPace;
    float tyreDelta;

    sessionFastestLap[0] = Data.Settings.DefaultPace;
    sessionFastestLap[1] = Data.Settings.DefaultPace;

    try
    {
        //Gets the fastest lap in FP1
        foreach (Stint s in RaceWeekendSessionStints[0])
        {
            stintFastestLap = s.FastestLap();
            if (stintFastestLap < sessionFastestLap[0])
                sessionFastestLap[0] = stintFastestLap;
        }

        //Gets the fastest lap in FP2
        foreach (Stint s in RaceWeekendSessionStints[1])
        {
            stintFastestLap = s.FastestLap();
            if (stintFastestLap < sessionFastestLap[1])
                sessionFastestLap[1] = stintFastestLap;
        }
    }
}

```

```

        //Calculates the difference between these two times, taking into account
        the track evolution.
        tyreDelta = (sessionFastestLap[1] - Data.Settings.TrackEvolution[1]) -
        (sessionFastestLap[0] - Data.Settings.TrackEvolution[0]);
    }
    catch
    {
        //return default tyre delta
        tyreDelta = Data.Settings.DefaultCompoundDelta;
    }

    //check within sensible limits.
    if ((tyreDelta < -(sessionFastestLap[1] * 0.02)) || (tyreDelta >= 0))
    {
        tyreDelta = Data.Settings.DefaultCompoundDelta;
    }

    return tyreDelta;
}
/// <summary>
/// Calculates the effect of fuel for this driver
/// </summary>
/// <returns>The time loss per kilogram of fuel carried in the car</returns>
public float GetFuelEffect()
{
    float[] sessionFastestLap = new float[2];
    float stintFastestLap = Data.Settings.DefaultPace;
    float fuelEffect;

    sessionFastestLap[0] = Data.Settings.DefaultPace; //FP2
    sessionFastestLap[1] = Data.Settings.DefaultPace; //FP3

    try
    {
        //Gets the fastest lap from FP2
        foreach (Stint s in RaceWeekendSessionStints[1])
        {
            stintFastestLap = s.FastestLap();
            if (stintFastestLap < sessionFastestLap[0])
                sessionFastestLap[0] = stintFastestLap;
        }

        //Gets the fastest lap from FP3
        stintFastestLap = Data.Settings.DefaultPace;
        foreach (Stint s in RaceWeekendSessionStints[2])
        {
            stintFastestLap = s.FastestLap();
            if (stintFastestLap < sessionFastestLap[1])
                sessionFastestLap[1] = stintFastestLap;
        }

        //Finds the difference between these two times, taking into account the
        track evolution
        fuelEffect = ((sessionFastestLap[0] - Data.Settings.TrackEvolution[1]) -
        (sessionFastestLap[1] - Data.Settings.TrackEvolution[2])) / FuelLoadInP2;
    }
    catch
    {
        //return default
        return Data.Settings.DefaultFuelEffect;
    }

    //Checks within sensible bounds.
    if ((fuelEffect > (sessionFastestLap[1] * 0.002)) || (fuelEffect <= 0))
    {
        fuelEffect = Data.Settings.DefaultFuelEffect;
    }
}

```

```

        }

        return fuelEffect;
    }
/// <summary>
/// Calculates the degradation per lap on the prime tyre
/// </summary>
/// <returns>The seconds per lap loss from new on the prime tyre</returns>
public float GetPrimeDegradation()
{
    /*finding prime tyre degradation
     * Find the longest stint in FP1
     * find the average degradation throughout this stint
     * correct for fuel effects.
     * return default if negative*/

    int stintLength = 0;
    int stintIndex = 0;
    int longestStintLength = 0;
    int longestStintIndex = 0;
    bool canReadFP1 = true;
    bool validate = true;

    float rawPrimeDegradation = 0;
    float primeDegradation = 0;

    //Checks that sessions are present in FP1
    if (RaceWeekendSessionStints[0].Count == 0) { canReadFP1 = false; }

    if (canReadFP1)
    {
        //Finds the longest stint in FP1
        foreach (Stint s in RaceWeekendSessionStints[0])
        {
            stintLength = s.lapTimes.Count;
            if (stintLength > longestStintLength)
            {
                longestStintLength = stintLength;
                longestStintIndex = stintIndex;
            }

            stintIndex++;
        }
    }
    else
    {
        primeDegradation = Data.Settings.DefaultPrimeDegradation;
        validate = false;
    }

    //Avoids a null reference exception
    if (longestStintIndex < RaceWeekendSessionStints[0].Count &&
longestStintIndex >= 0)
    {
        //Sets the raw value for prime tyre degradation by calculating
degradation from the longest stint.
        rawPrimeDegradation =
RaceWeekendSessionStints[0][longestStintIndex].AverageDegradation();
    }
    else
    {
        primeDegradation = Data.Settings.DefaultPrimeDegradation;
        rawPrimeDegradation = 0;
        validate = false;
    }

    //Adjusts degradation for the calculated fuel effect.
}

```

```

        rawPrimeDegradation -= (FuelEffectPerKilo * FuelConsumptionPerLap);

        if (validate) //If the value has been changed
        {
            //Checks within reasonable bounds.
            if (rawPrimeDegradation > 0 && rawPrimeDegradation <
RaceWeekendSessionStints[0][longestStintIndex].FastestLap() * 0.003)
            {
                primeDegradation = rawPrimeDegradation;
            }
            else
            {
                primeDegradation = Data.Settings.DefaultPrimeDegradation;
            }
        }

        return primeDegradation;
    }
    /// <summary>
    /// Contains data about the two fastest laps from any given session, used
    /// for calculating the time loss per lap on the option tyre
    /// </summary>
    struct fastestLaps
    {
        public int fastestLapIndex;
        public int secondFastestLapIndex;
        public float fastestLap;
        public float secondFastestLap;
        public float effectOffFuel;
        public float degradation;
    };
    public float GetOptionDegradation()
    {
        /*finding the option tyre degradation:
         * for FP2 and FP3 in stint containing the fastest lap
         * check difference between fastest and second fastest lap
         * raw delta = later lap - earlier lap
         * correct for fuel effects
         * average between sessions
        */

        /*validation:
         * ignore if lap is greater than 1% of fastest lap
         * set results to default if:
         *   result is negative
         *   result is less than prime tyre degradation
        */

        fastestLaps[] sessionFastestLaps = new fastestLaps[2];
        int[] stintContainingFastestLap = new int[2];
        float stintFastestLap = Data.Settings.DefaultPace;
        int stintIndex;
        int lapNo;
        float optionDegradation = 0;
        int sessionsRead = 0;
        bool canReadFP2 = true;
        bool canReadFP3 = true;

        if (RaceWeekendSessionStints[1].Count == 0) { canReadFP2 = false; }
        if (RaceWeekendSessionStints[2].Count == 0) { canReadFP3 = false; }

        for (int i = 0; i <= 1; i++)
        {
            sessionFastestLaps[i].degradation = 0;
            sessionFastestLaps[i].effectOffFuel = 0;
            sessionFastestLaps[i].fastestLap = Data.Settings.DefaultPace;
            sessionFastestLaps[i].secondFastestLap = Data.Settings.DefaultPace;
        }
    }
}

```

```

        sessionFastestLaps[i].fastestLapIndex = 0;
        sessionFastestLaps[i].secondFastestLapIndex = 0;
    }

    stintContainingFastestLap[0] = 0; //FP2
    stintContainingFastestLap[1] = 1; //FP3

    if (canReadFP2)
    {
        try //get stint containing fastest lap from FP2
        {
            stintIndex = 0;
            foreach (Stint s in RaceWeekendSessionStints[1])
            {
                stintFastestLap = s.FastestLap();
                if (stintFastestLap < sessionFastestLaps[0].fastestLap)
                {
                    sessionFastestLaps[0].fastestLap = stintFastestLap;
                    stintContainingFastestLap[0] = stintIndex;
                }
                stintIndex++;
            }
        }
        catch
        {
            canReadFP2 = false;
        }
    }

    if (canReadFP2)
    {
        try //get degradation from FP2
        {
            lapNo = 0;
            foreach (float lap in
RaceWeekendSessionStints[1][stintContainingFastestLap[0]].lapTimes)
            {
                if ((lap < sessionFastestLaps[0].secondFastestLap) && (lap >
sessionFastestLaps[0].fastestLap))
                {
                    sessionFastestLaps[0].secondFastestLap = lap;
                    sessionFastestLaps[0].secondFastestLapIndex = lapNo;
                }
                if (lap == sessionFastestLaps[0].fastestLap)
                {
                    sessionFastestLaps[0].fastestLapIndex = lapNo;
                }
                lapNo++;
            }

            //Adjust according to the effect of fuel
            sessionFastestLaps[0].effectOfFuel = (FuelEffectPerKilo *
FuelConsumptionPerLap) * Math.Abs((sessionFastestLaps[0].secondFastestLapIndex -
sessionFastestLaps[0].fastestLapIndex));
            sessionFastestLaps[0].degradation = (sessionFastestLaps[0].fastestLap
- (sessionFastestLaps[0].secondFastestLap - sessionFastestLaps[0].effectOfFuel));
        }
        catch
        {
            sessionFastestLaps[0].degradation =
Data.Settings.DefaultOptionDegradation;
        }
    }
    else //set to default
    {

```

```

        sessionFastestLaps[0].degradation =
Data.Settings.DefaultOptionDegradation;
    }

    if (canReadFP3)
    {
        try //get stint containing fastest lap from FP3
        {
            stintIndex = 0;
            foreach (Stint s in RaceWeekendSessionStints[2])
            {
                stintFastestLap = s.FastestLap();
                if (stintFastestLap < sessionFastestLaps[1].fastestLap)
                {
                    sessionFastestLaps[1].fastestLap = stintFastestLap;
                    stintContainingFastestLap[1] = stintIndex;
                }
                stintIndex++;
            }
        }
        catch
        {
            canReadFP3 = false;
        }
    }

    if (canReadFP3)
    {
        try //get degradation from FP3
        {
            lapNo = 0;
            foreach (float lap in
RaceWeekendSessionStints[2][stintContainingFastestLap[1]].lapTimes)
            {
                if ((lap < sessionFastestLaps[1].secondFastestLap) && (lap >
sessionFastestLaps[1].fastestLap))
                {
                    sessionFastestLaps[1].secondFastestLap = lap;
                    sessionFastestLaps[1].secondFastestLapIndex = lapNo;
                }
                if (lap == sessionFastestLaps[1].fastestLap)
                {
                    sessionFastestLaps[1].fastestLapIndex = lapNo;
                }
                lapNo++;
            }

            //adjust for fuel
            sessionFastestLaps[1].effectOfFuel = (FuelEffectPerKilo *
FuelConsumptionPerLap) * (sessionFastestLaps[1].secondFastestLapIndex -
sessionFastestLaps[1].fastestLapIndex);
            sessionFastestLaps[1].degradation =
((sessionFastestLaps[1].secondFastestLap - sessionFastestLaps[1].effectOfFuel) -
sessionFastestLaps[1].fastestLap);
        }
        catch
        {
            sessionFastestLaps[1].degradation =
Data.Settings.DefaultOptionDegradation;
        }
    }
    else
    {
        sessionFastestLaps[1].degradation =
Data.Settings.DefaultOptionDegradation;
    }
}

```

```

        //For each fastestLaps in the sessionFastestLaps array:
        for (int i = 0; i <= 1; i++)
        {
            if (sessionFastestLaps[i].degradation == 0) {
                sessionFastestLaps[i].degradation = Data.Settings.DefaultOptionDegradation; }
            if ((sessionFastestLaps[i].degradation <
                sessionFastestLaps[i].fastestLap * 0.01) && (sessionFastestLaps[i].degradation >
                tyreDegradation[0])) { optionDegradation += sessionFastestLaps[i].degradation;
                sessionsRead++; }
        }

        //Divides by the number of valid sessions found
        if (sessionsRead == 0) { optionDegradation =
            Data.Settings.DefaultOptionDegradation; }
        else { optionDegradation /= sessionsRead; }

        return optionDegradation;
    }
    /// <summary>
    /// Calculates the fastest lap from the race weekend, representing the low
    fuel pace
    /// of the driver
    /// </summary>
    /// <returns>The driver's fastest lap from the race weekend</returns>
    public float GetLowFuelPace()
    {
        float stintFastestLap;
        float fastestLap = Data.Settings.DefaultPace;

        //Finds the fastest lap across all sessions of the weekend
        for (int sessionDataIndex = 0; sessionDataIndex <= 3; sessionDataIndex++)
        {
            foreach (Stint s in RaceWeekendSessionStints[sessionDataIndex])
            {
                //Adjusts according to track evolution
                stintFastestLap = s.FastestLap() -
                    Data.Settings.TrackEvolution[sessionDataIndex];
                if (stintFastestLap < fastestLap)
                {
                    fastestLap = stintFastestLap;
                }
            }
        }

        return fastestLap;
    }

    /// <summary>
    /// Averages all relevant driver parameters across their teams to improve
    /// data accuracy
    /// </summary>
    public static void AverageBetweenTeammates()
    {
        float teamFuelEffect, teamPrimeDeg, teamOptionDeg;
        int nonDefaultFuel, nonDefaultPrime, nonDefaultOption;

        //Cycles through every other driver. driverIndex + 1 denotes the teammate.
        for (int driverIndex = 0; driverIndex < Data.NumberOfDrivers; driverIndex
        += 2)
        {
            teamFuelEffect = 0;
            teamPrimeDeg = 0;
            teamOptionDeg = 0;
            nonDefaultFuel = 0;
            nonDefaultPrime = 0;
            nonDefaultOption = 0;

```

```

        //If the driver's fuel effect is not the default value, the cumulative
total is increased and the number of valid data points is incremented.
        if (Data.Drivers[driverIndex].FuelEffectPerKilo != Data.Settings.DefaultFuelEffect) { teamFuelEffect +=
Data.Drivers[driverIndex].FuelEffectPerKilo; nonDefaultFuel++; }
            if (Data.Drivers[driverIndex+1].FuelEffectPerKilo != Data.Settings.DefaultFuelEffect) { teamFuelEffect += Data.Drivers[driverIndex + 1].FuelEffectPerKilo; nonDefaultFuel++; }

        if (nonDefaultFuel != 0) //Prevents dividing by zero
        {
            teamFuelEffect /= nonDefaultFuel; //calculates the average fuel
effect
            Data.Drivers[driverIndex].FuelEffectPerKilo = teamFuelEffect; //sets
both drivers' effects to this value
            Data.Drivers[driverIndex + 1].FuelEffectPerKilo = teamFuelEffect;
        }

        //As above for tyre type.
        if (Data.Drivers[driverIndex].tyreDegradation[TyreType.Prime] != Data.Settings.DefaultPrimeDegradation) { teamPrimeDeg +=
Data.Drivers[driverIndex].tyreDegradation[TyreType.Prime]; nonDefaultPrime++; }
            if (Data.Drivers[driverIndex + 1].tyreDegradation[TyreType.Prime] != Data.Settings.DefaultPrimeDegradation) { teamPrimeDeg += Data.Drivers[driverIndex + 1].tyreDegradation[TyreType.Prime]; nonDefaultPrime++; }
                if (nonDefaultPrime != 0)
                {
                    teamPrimeDeg /= nonDefaultPrime;
                    Data.Drivers[driverIndex].tyreDegradation[TyreType.Prime] =
teamPrimeDeg;
                    Data.Drivers[driverIndex + 1].tyreDegradation[TyreType.Prime] =
teamPrimeDeg;
                }

        if (Data.Drivers[driverIndex].tyreDegradation[TyreType.Option] != Data.Settings.DefaultOptionDegradation) { teamOptionDeg +=
Data.Drivers[driverIndex].tyreDegradation[TyreType.Option]; nonDefaultOption++; }
            if (Data.Drivers[driverIndex + 1].tyreDegradation[TyreType.Option] != Data.Settings.DefaultOptionDegradation) { teamOptionDeg += Data.Drivers[driverIndex + 1].tyreDegradation[TyreType.Option]; nonDefaultOption++; }
                if (nonDefaultOption != 0)
                {
                    teamOptionDeg /= nonDefaultOption;
                    Data.Drivers[driverIndex].tyreDegradation[TyreType.Option] =
teamOptionDeg;
                    Data.Drivers[driverIndex + 1].tyreDegradation[TyreType.Option] =
teamOptionDeg;
                }
            }

/// <summary>
/// Calculates the optimum strategy for the to use to complete the race
/// </summary>
/// <param name="driverIndex">The index of the driver being calculated</param>
/// <param name="trackIndex">The index of the track that the race will be run
on</param>
/// <returns>The optimised strategy with lap times and pit stops
calculated</returns>
public Strategy OptimiseStrategy(Driver driver, int trackIndex)
{
    float currentStrategyTime = 0;
    float bestStrategyTime = 0;
    int strategyIteration = 0;
    Strategy bestStrategy = null;
    Strategy currentStrategy;
}

```

```

        //find shortest time throughout this loop
        for (int stints = 2; stints <= 4; stints++)
        {
            for (int primeStints = 1; primeStints < stints; primeStints++)
            {
                //get the currently defined strategy
                currentStrategy = new Strategy(stints, primeStints, driver);
                currentStrategyTime = currentStrategy.TotalTime;

                //set the initial value for best time
                if (strategyIteration == 0)
                {
                    bestStrategyTime = currentStrategyTime;
                    bestStrategy = currentStrategy;
                }
                //uses a most wanted holder to hold the best strategy calculated so
                far.
                if (currentStrategyTime < bestStrategyTime)
                {
                    bestStrategyTime = currentStrategyTime;
                    bestStrategy = currentStrategy;
                }

                strategyIteration++;
            }
        }

        return bestStrategy;
    }

    /// <summary>
    ///
    /// </summary>
    /// <param name="upToLap">The lap on which to terminate counting</param>
    /// <returns>The number of pit stops the driver will complete before this
    lap.</returns>
    public int StopsBeforeLap(int upToLap)
    {
        return SelectedStrategy.PitStops.Count<int>(i => i <= upToLap);
    }

    /// <summary>
    /// Clears data from the sessions completed by the driver.
    /// Used when changing the race to be calculated.
    /// </summary>
    public void ClearSessions()
    {
        foreach (var list in RaceWeekendSessionStints)
        {
            list.Clear();
        }
    }

    /// <summary>
    /// Gets or sets the driver's currently stored strategy
    /// </summary>
    public Strategy Strategy
    {
        get { return SelectedStrategy; }
        set { SelectedStrategy = value; }
    }
}

using System;
using System.Collections.Generic;

```

```

namespace StratSim.Model
{
    /// <summary>
    /// Class representing extension methods, predominantly for the list and array
    /// classes.
    /// Includes searching and sorting routines.
    /// </summary>
    public static class ExtensionMethods
    {
        /// <summary>
        /// Sorts a list of type T according to the comparer
        /// </summary>
        /// <typeparam name="T">The type of elements in the list</typeparam>
        /// <param name="comparer">Use a > b for an ascending list.</param>
        public static void Sort<T>(this List<T> List, Func<T, T, bool> comparer)
        {
            T[] Array = List.ToArray();
            Array.Sort<T>(comparer);
        }

        /// <summary>
        /// Sorts an array of type T according to the comparer
        /// </summary>
        /// <typeparam name="T">The type of elements in the array</typeparam>
        /// <param name="comparer">Use a > b for an ascending list.</param>
        public static void Sort<T>(this T[] Array, Func<T, T, bool> comparer)
        {
            int first = 0;
            int last = Array.Length - 1;

            if (last - first >= 4)
            {
                Functions.QuickSort<T>(ref Array, first, last, comparer);
            }
            else
            {
                Functions.InsertionSort<T>(ref Array, first, last, comparer);
            }
        }

        /// <summary>
        /// Checks if the specified element exists in a list.
        /// </summary>
        /// <param name="List"></param>
        /// <param name="itemToFind">The item to find in the list</param>
        /// <returns>True if the value is in the list</returns>
        public static bool Exists(this List<int> List, int itemToFind)
        {
            int Index = 0;

            return List.Exists(itemToFind, false, out Index);
        }

        /// <summary>
        /// Checks if the specified element exists in a list and returns the location
        /// of the element in the list.
        /// </summary>
        /// <param name="List"></param>
        /// <param name="itemToFind">The item to find in the list</param>
        /// <param name="isSorted">True if the list is already sorted into ascending
        /// order</param>
        /// <param name="Index">Returns the index at which the element is found.
        /// <returns>-1 if the value is not in the list.</returns>
        /// <returns>True if the specified element can be found in the list.</returns>
        public static bool Exists(this List<int> List, int itemToFind, bool isSorted,
        out int Index)
        {
    
```

```

        int[] Array = List.ToArray();

        return Array.Exists(itemToFind, isSorted, out Index);
    }

    /// <summary>
    /// Checks if the specified element exists in an array and returns the index
    of the element in the array.
    /// </summary>
    /// <param name="Array"></param>
    /// <param name="itemToSearch">The item to find in the array</param>
    /// <param name="isSorted">True if the array is already sorted in ascending
    order</param>
    /// <param name="Index">Returns the index at which the element is found.
    /// Returns -1 if the value is not in the list.</param>
    /// <returns>True if the specified element can be found in the
    array.</returns>
    public static bool Exists(this int[] Array, int itemToSearch, bool isSorted,
out int Index)
    {
        return Functions.StartBinarySearch(ref Array, itemToSearch, isSorted, out
Index);
    }
}

using System;
using System.Windows.Forms;

namespace StratSim.Model
{
    /// <summary>
    /// Contains methods and functions that are used elsewhere in the program
    /// </summary>
    public class Functions
    {
        public Functions()
        { }

        /// <summary>
        /// Sorts an array of type T using the quicksort algorithm
        /// </summary>
        /// <typeparam name="T">The type of elements in the array to be
        sorted</typeparam>
        /// <param name="Array">An array of elements to be sorted</param>
        /// <param name="first">The location at which to commence the sort</param>
        /// <param name="last">The location at which to end the sort</param>
        /// <param name="Comparer">A comparing function for two types in the array.
        /// a > b produces an ascending sort.
        /// The comparer should not include the 'equal to' statement.</param>
        public static void QuickSort<T>(ref T[] Array, int first, int last, Func<T, T,
bool> Comparer)
        {
            //Validates the list to more than one item
            if (first < last)
            {
                T pivotValue;
                int leftPointer, rightPointer, pivot;
                T temp;

                leftPointer = first + 1;
                rightPointer = last;
                pivot = first;
                pivotValue = Array[pivot];
            }
        }
    }
}

```

```

//Commences sort
while (leftPointer <= rightPointer)
{
    //scan to the left of the pivot
    while ((leftPointer <= rightPointer) && !Comparer(Array[leftPointer],
pivotValue))
    {
        leftPointer++;
    }
    //scan to the right of the pivot
    while ((leftPointer <= rightPointer) && Comparer(Array[rightPointer],
pivotValue))
    {
        rightPointer--;
    }

    //swap if two values need to be swapped.
    if (leftPointer < rightPointer)
    {
        temp = Array[leftPointer];
        Array[leftPointer] = Array[rightPointer];
        Array[rightPointer] = temp;
    }
}

temp = Array[first];
Array[first] = Array[rightPointer];
Array[rightPointer] = temp;

//Quicksort the remaining values
QuickSort<T>(ref Array, first, rightPointer - 1, Comparer);
QuickSort<T>(ref Array, rightPointer + 1, last, Comparer);
}
}

/// <summary>
/// Sorts an array of type T using the insertion sort algorithm
/// </summary>
/// <typeparam name="T">The type of elements in the array to be
sorted</typeparam>
/// <param name="array">An array of elements to be sorted</param>
/// <param name="first">The location at which to commence the sort</param>
/// <param name="last">The location at which to end the sort</param>
/// <param name="Comparer">A comparing function for two types in the array.
/// a > b produces an ascending sort.
/// The comparer should not include the 'equal to' statement.</param>
public static void InsertionSort<T>(ref T[] array, int first, int last,
Func<T,T,bool> comparer)
{
    int pointer;
    T currentValue;

    for (int arrayPosition = first + 1; arrayPosition <= last; ++arrayPosition)
    {
        currentValue = array[arrayPosition];
        pointer = arrayPosition;
        while ((--pointer >= 0) && !comparer(currentValue, array[pointer]))
        {
            array[pointer + 1] = array[pointer];
        }
        array[pointer + 1] = currentValue;
    }
}

/// <summary>
/// Initiates a binary search of an array for the specified value
/// </summary>

```

```

    /// <param name="Array">The array in which to find the value</param>
    /// <param name="itemToFind">The target item to find</param>
    /// <param name="isSorted">Represents whether the array is already sorted or
not</param>
    /// <param name="Index">Outputs the index at which the required item was
found.
    /// Outputs -1 if item is not found.</param>
    /// <returns>True if the item is anywhere in the array</returns>
    public static bool StartBinarySearch(ref int[] Array, int itemToFind, bool
isSorted, out int Index)
{
    Index = 0;
    int first = 0;
    int last = Array.Length - 1;
    bool exitLoop = false;

    if (!isSorted) { Array.Sort<int>((a, b) => a > b); }

    return BinarySearch(ref Array, itemToFind, first, last, ref exitLoop, out
Index);
}

/// <summary>
/// Searches for a specified element in an array using a binary search
algorithm.
/// </summary>
/// <param name="Array">The array to search</param>
/// <param name="itemToFind">The item to find in the array</param>
/// <param name="first">The first index to search</param>
/// <param name="last">The last index to search</param>
/// <param name="exitLoop">Represents whether the whole array has been
searched. This is true if: '/n'
/// The value is confirmed not in the array, or '/n'
/// The value is found</param>
/// <param name="Index">Outputs the index at which the specified item was
found.</param>
/// <returns>True if the item is in the list.</returns>
static bool BinarySearch(ref int[] Array, int itemToFind, int first, int last,
ref bool exitLoop, out int Index)
{
    Index = -1;
    int topIndex = last;
    int bottomIndex = first;
    int middleIndex = 0;
    bool itemFound = false;

    while (topIndex >= bottomIndex && !exitLoop)
    {
        middleIndex = (topIndex - bottomIndex) / 2 + bottomIndex;

        if (Array[middleIndex] == itemToFind)
        {
            itemFound = true;
            Index = middleIndex;
        }
        else
        {
            exitLoop = (itemFound || (topIndex == bottomIndex));
            if (!exitLoop)
            {
                if (Array[middleIndex] > itemToFind)
                {
                    itemFound = BinarySearch(ref Array, itemToFind, bottomIndex,
middleIndex - 1, ref exitLoop, out Index);
                }
                else
                {

```

```

        itemFound = BinarySearch(ref Array, itemToFind, middleIndex +
1, topIndex, ref exitLoop, out Index);
    }
}
return itemFound;
}

/* The following routines are validation routines
 * They will return the value that is sent to them for quick assignment.
 * They require a bound, an error message, and a boolean that is updated if
the value is false
 * These statements can be stacked so that the incorrect value is set to true
if at least one value is incorrect.
 */
public static float ValidateNotEqualTo(float value, float notEqualTo, string
field, ref bool incorrectValue, bool showDialog)
{
    //assign to incorrect value
    if (!incorrectValue)
    {
        incorrectValue = (value == notEqualTo);

        //select on boolean for speed of execution
        if (incorrectValue && showDialog)
        {
            //display a warning message
            string warningMessage = "The field " + field + " cannot be set to " +
Convert.ToString(notEqualTo);
            var warning = MessageBox.Show(warningMessage, "Incorrect Value",
MessageBoxButtons.OK, MessageBoxIcon.Warning);
        }
    }
    return value;
}
public static float ValidateLessThan(float value, float notGreaterThan, string
field, ref bool incorrectValue, bool showDialog)
{
    ValidateNotEqualTo(value, notGreaterThan, field, ref incorrectValue,
showDialog);
    ValidateLessThanOrEqualTo(value, notGreaterThan, field, ref incorrectValue,
showDialog);
    return value;
}
public static float ValidateGreaterThanOrEqual(float value, float notLessThan, string
field, ref bool incorrectValue, bool showDialog)
{
    ValidateNotEqualTo(value, notLessThan, field, ref incorrectValue,
showDialog);
    ValidateGreaterThanOrEqualTo(value, notLessThan, field, ref incorrectValue,
showDialog);
    return value;
}
public static float ValidateGreaterThanOrEqualTo(float value, float notLessThan,
string field, ref bool incorrectValue, bool showDialog)
{
    if (!incorrectValue)
    {
        //assign to incorrect value
        incorrectValue = (value < notLessThan);

        //switch on boolean for speed of execution
        if (incorrectValue && showDialog)
        {
            //display a warning message
        }
    }
}
```

```

        string warningMessage = "The field " + field + " cannot be less than
" + Convert.ToString(notLessThan);
        var warning = MessageBox.Show(warningMessage, "Incorrect Value",
MessageBoxButtons.OK, MessageBoxIcon.Warning);
    }
}
return value;
}
public static float ValidateLessThanOrEqualTo(float value, float notGreaterThanOr-
string field, ref bool incorrectValue, bool showDialog)
{
    if (!incorrectValue)
    {
        //assign to incorrect value
        incorrectValue = (value > notGreaterThanOr);

        //switch on boolean for speed of execution
        if (incorrectValue && showDialog)
        {
            //display a warning message
            string warningMessage = "The field " + field + " cannot be greater
than " + Convert.ToString(notGreaterThanOr);
            var warning = MessageBox.Show(warningMessage, "Incorrect Value",
MessageBoxButtons.OK, MessageBoxIcon.Warning);
        }
    }

    //return the value
    return value;
}
public static float ValidateBetweenExclusive(float value, float lowerBound,
float upperBound, string field, ref bool incorrectValue, bool showDialog)
{
    value = ValidateLessThan(value, upperBound, field, ref incorrectValue,
showDialog);
    value = ValidateGreaterThan(value, lowerBound, field, ref incorrectValue,
showDialog);

    return value;
}
public static float ValidateBetweenInclusive(float value, float lowerBound,
float upperBound, string field, ref bool incorrectValue, bool showDialog)
{
    value = ValidateLessThanOrEqualTo(value, upperBound, field, ref
incorrectValue, showDialog);
    value = ValidateGreaterThanOrEqualTo(value, lowerBound, field, ref
incorrectValue, showDialog);

    return value;
}

/// <summary>
/// Controls the calculation all of the pace parameters for each driver
/// </summary>
/// <param name="raceIndex">The race index on which the race is being
run</param>
public static void CalculatePaceParameters(int raceIndex)
{
    //get parameters
    foreach (Driver d in Data.Drivers)
    {
        if (d.TopSpeed == 0) { d.TopSpeed = Data.Settings.DefaultTopSpeed; }
        d.PitStopLoss = Data.Tracks[raceIndex].pitStopLoss;
        d.FuelConsumptionPerLap = Data.Tracks[raceIndex].fuelPerLap;
        d.TyrePaceDelta = d.GetTyreDelta();
        d.FuelEffectPerKilo = d.GetFuelEffect();
        d.TyreDegradation[TyreType.Prime] = d.GetPrimeDegradation();
    }
}

```

```

        d.TyreDegradation[TyreType.Option] = d.GetOptionDegradation();
        d.LowFuelPace = d.GetLowFuelPace();
    }

    Driver.AverageBetweenTeammates()
    Program.InfoPanel.WriteData("Driver pace successfully analysed");
}

/// <summary>
/// Optimises the strategies of all drivers.
/// Sets the driver's strategy to the optimised strategy.
/// </summary>
public static void OptimiseAllStrategies(int raceIndex)
{
    int driverIndex = 0;
    int laps = Data.Tracks[raceIndex].laps;

    foreach (Driver d in Data.Drivers)
    {
        d.Strategy = d.OptimiseStrategy(Data.Drivers[driverIndex++], raceIndex);
    }
}

/// <summary>
/// Opens a dialog box with the specified message and caption.
/// </summary>
/// <param name="message">The message to be displayed to the user, identifying
the error and
    /// providing information on how to fix it.</param>
    /// <param name="caption">The message to be displayed in the header of the
page</param>
    /// <returns>True if the 'yes' button is clicked, else false</returns>
public static bool StartDialog(string message, string caption)
{
    bool valueToReturn = false;
    var buttons = MessageBoxButtons.YesNo;
    var icon = MessageBoxIcon.Warning;
    var defaultButton = MessageBoxDefaultButton.Button2;

    var dialog = MessageBox.Show(message, caption, buttons, icon,
defaultButton);

    switch (dialog)
    {
        case DialogResult.Yes:
            valueToReturn = true;
            break;
        case DialogResult.No:
            valueToReturn = false;
            break;
        default: valueToReturn = false; break;
    }
}

return valueToReturn;
}

/// <summary>
/// Initialises a 2D array with every element at the specified value
/// </summary>
/// <typeparam name="T">The type of the array</typeparam>
/// <param name="columns">The width of the array</param>
/// <param name="rows">The length of the array</param>
/// <param name="valueToSet">The value to set all elements to</param>
/// <returns>The populated array of values</returns>
public static T[,] InitialiseArrayAtValue<T>(int columns, int rows, T
valueToSet)
{
}

```

```

        T[,] newArray = new T[columns, rows];

        for (int column = 0; column < columns; column++)
        {
            for (int row = 0; row < rows; row++)
            {
                newArray[column, row] = valueToSet;
            }
        }
        return newArray;
    }
}
}

using System.Collections.Generic;

namespace StratSim.Model
{
    public class PitStop
    {
        private int _position, _lapNumber;
        float pitStopLoss;

        public PitStop(int position, int lapNumber, int trackIndex)
        {
            _position = position;
            _lapNumber = lapNumber;
            pitStopLoss = Data.Tracks[trackIndex].pitStopLoss;
        }

        /// <summary>
        /// Updates timings in a race when a pit stop is made
        /// </summary>
        /// <param name="strategies">The array of strategies representing a
race</param>
        public void SimulateStop(ref Strategy[] strategies)
        {
            AddTimeDueToStop(ref strategies, _position, _lapNumber, pitStopLoss);
        }
        public void AddTimeDueToStop(ref Strategy[] strategies, int position, int
lapNumber, float pitStopTimeLoss)
        {
            strategies[position].RaceLapTimes[lapNumber] += pitStopTimeLoss;
        }

        /// <summary>
        /// Updates the positions when a pit stop is made using a simple insert sort.
        /// Caution, O(n) for the size of the race.
        /// </summary>
        /// <param name="strategies">The list of strategies representing a
race</param>
        /// <param name="pitStops">The list of pit stops to process</param>
        /// <param name="trackIndex">The index of the track on which the race is
taking place</param>
        public static void UpdateRacePositionsAfterPitStop(ref Strategy[] strategies,
List<PitStop> pitStops, int trackIndex)
        {
            Strategy temp;

            foreach (PitStop s in pitStops)
            {
                int overtakingDownTo = s.Position;

```

```

        while ((overtakeDownTo + 1 < strategies.Length) &&
(strategies[overtakeDownTo + 1].CumulativeTime <
strategies[overtakeDownTo].CumulativeTime + Data.Tracks[trackIndex].pitStopLoss))
{
    temp = strategies[overtakeDownTo];
    strategies[overtakeDownTo] = strategies[overtakeDownTo + 1];
    strategies[overtakeDownTo + 1] = temp;

    ++overtakeDownTo;

    foreach (PitStop p in pitStops)
    {
        if (p.Position == overtakeDownTo) { --p.Position; }
    }
}

}

/// <summary>
/// Gets or sets the position from which a driver makes the pit stop
/// </summary>
public int Position
{
    get { return _position; }
    set { _position = value; }
}
/// <summary>
/// Gets the lap number on which a driver makes a pit stop
/// </summary>
public int LapNumber
{ get { return _lapNumber; } }
}

}

using StratSim.View.MyFlowLayout;
using StratSim.View.Panels;
using StratSim.View.UserControls;
using StratSim.ViewModel;
using System;
using System.Collections.Generic;
using System.IO;

namespace StratSim.Model
{
    public struct racePosition
    {
        public int position;
        public int driver;
        public float interval;
        public float gap;
        public float cumulativeTime;
    };

    public class Race : IDisposable
    {
        Strategy[] raceStrategies;
        Strategy[] originalStrategies;
        racePosition[,] positions;
        bool[,] forcedOvertake;

        int trackIndex;
        int laps;

        CumulativeTimeGraph raceGraph;
    }
}

```

```

    ///<summary>
    /// Sets up a race and updates the graph displayed on screen
    ///</summary>
    ///<param name="TrackIndex">The index of the track on which the race is
taking place</param>
    ///<param name="Strategies">The list of strategies representing drivers who
are to take part in the race</param>
    ///<param name="AssociatedForm">The form on which the graph is to be
displayed</param>
    public Race(int TrackIndex, Strategy[] Strategies, MainForm AssociatedForm)
    {
        trackIndex = TrackIndex;
        laps = Data.Tracks[trackIndex].laps;

        originalStrategies = (Strategy[])Strategies.Clone();
        raceStrategies = (Strategy[])Strategies.Clone();

        positions = new racePosition[Data.NumberOfDrivers, laps];
        forcedOvertake =
Functions.InitialiseArrayAtValue<bool>(Data.NumberOfDrivers, laps, false);

        raceGraph = new CumulativeTimeGraph(NormalisationType.OnEveryLap,
AssociatedForm);
        raceGraph.GraphRightClick += raceGraph_PitStopAddedOnClick;
        AssociatedForm.IOController.StartGraph(raceGraph);

        MyEvents.SettingsModified += MyEvents_SettingsModified;
    }

    void raceGraph_PitStopAddedOnClick(int driverIndex, int lapNumber)
    {
        forcedOvertake[driverIndex, lapNumber - 1] = true;
        ResetParameters(originalStrategies);
        SetupGrid();
        SimulateRace();
    }

    public Race() { }

    void MyEvents_SettingsModified()
    {
        CheckUpdate();
    }

    ///<summary>
    /// Sorts the strategies by pace to simulate a qualifying setup.
    ///</summary>
    public void SetupGrid()
    {
        Functions.QuickSort<Strategy>(ref raceStrategies, 0, raceStrategies.Length
- 1, (a, b) => a.Driver.LowFuelPace > b.Driver.LowFuelPace);
    }

    ///<summary>
    /// Runs a race simulation and outputs the results to a graph and to file.
    ///</summary>
    public void SimulateRace()
    {
        //overtake variables:
        Driver driverAhead, driverBehind;
        Strategy temp;

        for (int lapNumber = 0; lapNumber < laps; lapNumber++) //for each lap
        {
            for (int position = 0; position < Data.NumberOfDrivers; position++)
            {
                //check for lapped car encounters:

```

```

        bool stayInLoop = true;
        float previousLapDeltaTime = 0;
        float thisLapDeltaTime = 0;
        for (int lappedPosition = Data.NumberOfDrivers - 1; stayInLoop;
lappedPosition--)
        {
            previousLapDeltaTime =
(raceStrategies[lappedPosition].CumulativeTime -
raceStrategies[position].CumulativeTime);
            thisLapDeltaTime =
(raceStrategies[lappedPosition].RaceLapTimes[lapNumber] -
raceStrategies[position].RaceLapTimes[lapNumber]);
            if ((previousLapDeltaTime %
raceStrategies[position].RaceLapTimes[lapNumber]) < thisLapDeltaTime)
            {
                raceStrategies[lappedPosition].RaceLapTimes[lapNumber] += Data.Settings.BackmarkerLoss;
            }

            stayInLoop = (previousLapDeltaTime >
raceStrategies[position].RaceLapTimes[lapNumber]);
        }
    }

    for (int position = 1; position < raceStrategies.Length; position++)
    {
        //check for overtakes:
        driverAhead = raceStrategies[position - 1].Driver;
        driverBehind = raceStrategies[position].Driver;
        if (CalculateOvertake(driverAhead, driverBehind, position, lapNumber)
|| forcedOvertake[driverBehind.DriverIndex,lapNumber])
        {
            raceStrategies[position - 1].RaceLapTimes[lapNumber] += Data.Settings.TimeLoss;
            //swap strategies
            temp = raceStrategies[position - 1];
            raceStrategies[position - 1] = raceStrategies[position];
            raceStrategies[position] = temp;
        }
        else
        {
            //check for following in traffic
            if (FollowingInTraffic(position - 1, position, lapNumber))
            {
                float minimumTime = raceStrategies[position - 1].CumulativeTime
+ raceStrategies[position - 1].RaceLapTimes[lapNumber] + Data.Settings.TimeGap;
                float requiredLap = minimumTime -
raceStrategies[position].CumulativeTime;
                raceStrategies[position].RaceLapTimes[lapNumber] = requiredLap;
            }
        }
    }

    List<PitStop> pitStopList = new List<PitStop>();
    for (int position = 0; position < raceStrategies.Length; position++)
    {
        //update pit stops:
        if (raceStrategies[position].PitStops.Exists(l => l == lapNumber +
1))
        {
            pitStopList.Add(new PitStop(position, lapNumber, trackIndex));
        }
    }

    foreach (PitStop p in pitStopList)
    {
        p.SimulateStop(ref raceStrategies);
    }
}

```

```

        }

        PitStop.UpdateRacePositionsAfterPitStop(ref raceStrategies, pitStopList,
trackIndex);

        for (int position = 0; position < raceStrategies.Length; position++)
        {
            //update the times:
            raceStrategies[position].CumulativeTime +=
raceStrategies[position].RaceLapTimes[lapNumber];

            //update the positions:
            positions[position, lapNumber].driver =
raceStrategies[position].Driver.DriverIndex;
            positions[position, lapNumber].position = position;
            positions[position, lapNumber].cumulativeTime =
raceStrategies[position].CumulativeTime;

            if (position == 0)
            {
                positions[position, lapNumber].gap = 0;
                positions[position, lapNumber].interval = lapNumber;
            }
            else
            {
                positions[position, lapNumber].gap = GetGapBetweenCars(0,
position, lapNumber);
                positions[position, lapNumber].interval =
GetGapBetweenCars(position - 1, position, lapNumber);
            }
            MyEvents.OnUpdateIntervals(positions, lapNumber);
        } //end for each lap

        StartGraph();
    }

    float GetGapBetweenCars(int aheadPosition, int behindPosition, int lap)
    {
        float rawValueGap;

        if (aheadPosition < 0) { rawValueGap = 0F; }
        else { rawValueGap = raceStrategies[behindPosition].CumulativeTime -
raceStrategies[aheadPosition].CumulativeTime; }

        return rawValueGap;
    }

    bool CalculateOvertake(Driver driverAhead, Driver driverBehind, int
positionBehind, int lap)
    {
        Random rand = new Random();
        bool overtake;
        float probability;

        float cumulativeTimeDelta = raceStrategies[positionBehind].CumulativeTime -
raceStrategies[positionBehind - 1].CumulativeTime;
        float paceDelta = raceStrategies[positionBehind].RaceLapTimes[lap] -
raceStrategies[positionBehind - 1].RaceLapTimes[lap];
        float totalDelta = cumulativeTimeDelta + paceDelta;
        float speedDelta = driverBehind.TopSpeed - driverAhead.TopSpeed;

        probability = GetOvertakeProbability(paceDelta, speedDelta, totalDelta,
Data.Settings.RequiredPaceDelta, Data.Settings.RequiredSpeedDelta);

        float random = (float)rand.Next(0, 1001) / 1000F;
        overtake = (probability > random);
    }
}

```

```

        return overtake;
    }

    /// <summary>
    /// Gets the probability that an overtake will occur
    /// </summary>
    /// <param name="paceDelta">The pace of the first car - pace of second
car</param>
    /// <param name="speedDelta">Speed of first car - speed of second car</param>
    /// <param name="totalDelta">Total time gap between the two cars</param>
    /// <param name="requiredPaceDelta">Required pace delta to make an overtake
possible</param>
    /// <param name="requiredSpeedDelta">Required speed delta to make an overtake
possible</param>
    /// <returns>The probability between 0 and 1 that an overtake occurs</returns>
    public float GetOvertakeProbability(float paceDelta, float speedDelta, float
totalDelta, float requiredPaceDelta, float requiredSpeedDelta)
{
    float speedProbability = 0;
    float paceProbability = 0;

    if (requiredPaceDelta > totalDelta && requiredPaceDelta < paceDelta)
    {
        paceProbability = (paceDelta - requiredPaceDelta) /
Math.Abs(requiredPaceDelta);
    }
    else
    {
        paceProbability = 0;
    }

    if (requiredSpeedDelta < speedDelta)
    {
        speedProbability = (speedDelta - requiredSpeedDelta) /
Math.Abs(requiredSpeedDelta);
    }
    else
    {
        speedProbability = 0;
    }

    float probability = speedProbability * paceProbability;

    return probability;
}

bool FollowingInTraffic(int aheadPosition, int behindPosition, int lapNumber)
{
    float behindCumulativeTime = raceStrategies[behindPosition].CumulativeTime +
raceStrategies[behindPosition].RaceLapTimes[lapNumber];
    float aheadCumulativeTime = raceStrategies[aheadPosition].CumulativeTime +
raceStrategies[aheadPosition].RaceLapTimes[lapNumber];

    return ((behindCumulativeTime - aheadCumulativeTime) <
Data.Settings.TimeGap);
}

float FindGap(int lapNumber, int driver)
{
    for (int position = 0; position < Data.NumberOfDrivers; position++)
    {
        if (positions[position, lapNumber].driver == driver)
        {
            return positions[position, lapNumber].gap;
        }
    }
}

```

```

        return 0F;
    }

    float FindTime(int lapNumber, int driver)
    {
        for (int position = 0; position < Data.NumberOfDrivers; position++)
        {
            if (positions[position, lapNumber].driver == driver)
            {
                return positions[position, lapNumber].cumulativeTime;
            }
        }

        return 0F;
    }

    /// <summary>
    /// Creates the traces to be displayed and draws the graph of the race
    /// </summary>
    void StartGraph()
    {
        List<StrategyLine> traces = new List<StrategyLine>(Data.NumberOfDrivers);
        for (int driver = 0; driver < Data.NumberOfDrivers; driver++)
        {
            traces.Add(new StrategyLine(driver));
        }

        lapDataPoint tempPoint = new lapDataPoint();
        for (int lapNumber = 0; lapNumber < laps; lapNumber++)
        {
            for (int driver = 0; driver < Data.NumberOfDrivers; driver++)
            {
                tempPoint.time = FindTime(lapNumber, driver);
                tempPoint.lap = lapNumber;
                tempPoint.driver = driver;
                tempPoint.draw = true;

                traces[driver].AddPoint(tempPoint);
                if (lapNumber == laps - 1) { traces[driver].TotalTime =
tempPoint.time; }
            }
        }

        raceGraph.DrawGraph(traces, true, true);
    }

    void CheckUpdate()
    {
        string message = "Settings have been altered. This may affect the outcome
of the race simulation." +
                    "Restart simulation now?";
        string caption = "Restart Simulation";
        if (Functions.StartDialog(message, caption))
        {
            RestartSimulation(out raceGraph, originalStrategies);
        }
    }

    /// <summary>
    /// Restarts the race simulation when parameters have been changed
    /// </summary>
    /// <param name="graph">The new graph which is output when the race simulation
has been completed</param>
    /// <param name="strategies">The list of strategies to be used in the race
simulation</param>

```

```

    public void RestartSimulation(out CumulativeTimeGraph graph, Strategy[]
strategies)
{
    ResetParameters(strategies);
    raceGraph.SetRaceLaps(laps);
    raceGraph.SetupAxes();
    graph = this.raceGraph;
    forcedOvertake =
Functions.InitialiseArrayAtValue<bool>(Data.NumberOfDrivers, laps, false);
    SetupGrid();
    SimulateRace();
}

void ResetParameters(Strategy[] strategies)
{
    laps = Data.GetRaceLaps();

    originalStrategies = (Strategy[])strategies.Clone();
    raceStrategies = (Strategy[])strategies.Clone();

    foreach (Strategy s in strategies)
    {
        s.CumulativeTime = 0F;
        s.UpdateStrategyParameters();
    }

    positions = new racePosition[Data.NumberOfDrivers, laps];
}

/// <summary>
/// Writes the race data to a CSV file, writing data about each position on a
given lap,
/// then writing the next lap separated by a line
/// </summary>
void WriteRaceData()
{
    StreamWriter w = new StreamWriter("Race.txt");

    string positionString = "";

    for (int lapNumber = 0; lapNumber < laps; lapNumber++)
    {
        for (int position = 0; position < raceStrategies.Length; position++)
        {
            positionString = Convert.ToString(positions[position,
lapNumber].position);
            positionString += ',';
            positionString += Convert.ToString(positions[position,
lapNumber].driver);
            positionString += ',';
            positionString += Convert.ToString(Math.Round(positions[position,
lapNumber].gap, 3));
            positionString += ',';
            positionString += Convert.ToString(Math.Round(positions[position,
lapNumber].interval, 3));

            w.WriteLine(positionString);
        }
        w.WriteLine();
    }

    w.Dispose();
}

/// <summary>
/// Writes interval data from the race to separate files
/// </summary>

```

```

    void writeIntervalData()
    {
        StreamWriter intervals = new StreamWriter("intervals.txt");
        StreamWriter gaps = new StreamWriter("gaps.txt");
        StreamWriter drivers = new StreamWriter("driverOrder.txt");

        string intervalString = "";
        string gapString = "";
        string driverString = "";

        for (int lapNumber = 0; lapNumber < laps; lapNumber++)
        {
            for (int position = 0; position < raceStrategies.Length; position++)
            {
                intervalString = Convert.ToString(Math.Round(positions[position,
                    lapNumber].interval, 3));
                if (position != Strategies.Length - 1)
                    intervalString += ',';

                gapString = Convert.ToString(Math.Round(positions[position,
                    lapNumber].gap, 3));
                if (position != Strategies.Length - 1)
                    gapString += ',';

                driverString = Convert.ToString(positions[position,
                    lapNumber].driver);
                if (position != Strategies.Length - 1)
                    driverString += ',';

                intervals.Write(intervalString);
                gaps.Write(gapString);
                drivers.Write(driverString);
            }
            intervals.WriteLine();
            gaps.WriteLine();
            drivers.WriteLine();
        }

        intervals.Dispose();
        gaps.Dispose();
        drivers.Dispose();
    }

    public void Dispose()
    {
        raceGraph.Dispose();
    }

    /// <summary>
    /// Gets or sets the list of strategies in the race
    /// </summary>
    public Strategy[] Strategies
    {
        get { return raceStrategies; }
        set { raceStrategies = value; }
    }

    /// <summary>
    /// Gets the data from each race position
    /// </summary>
    public racePosition[,] Positions
    {
        get { return positions; }
    }
}

using System.Globalization;

```

```

using System.IO;

namespace StratSim.Model
{
    public class Settings
    {
        public Settings()
        {
            LoadData();
        }

        float requiredPaceDelta, requiredSpeedDelta, timeLoss, backmarkerLoss,
timeGap;
        float defaultCompoundDelta, optionDegradation, primeDegradation, topSpeed,
fuelEffect, pace, P2Fuel;
        string dataFilePath;
        float trackImprovement;
        string PDFReader;
        string timingDataBaseFolder;
        float[] trackEvolution = new float[5];

        public const string SettingsFile = "Settings.txt";

Multiple get-set accessors for settings fields removed to save space

        /// <summary>
        /// Writes the settings data to a file
        /// </summary>
        public void WriteSettingsData()
        {
            using (StreamWriter s = new StreamWriter(SettingsFile))
            {
                s.WriteLine(requiredPaceDelta);
                s.WriteLine(requiredSpeedDelta);
                s.WriteLine(timeLoss);
                s.WriteLine(backmarkerLoss);
                s.WriteLine(timeGap);
                s.WriteLine(defaultCompoundDelta);
                s.WriteLine(optionDegradation);
                s.WriteLine(primeDegradation);
                s.WriteLine(topSpeed);
                s.WriteLine(fuelEffect);
                s.WriteLine(pace);
                s.WriteLine(P2Fuel);
                s.WriteLine(dataFilePath);

                for (int sessionIndex = 0; sessionIndex <= 3; sessionIndex++)
                {
                    s.Write(trackEvolution[sessionIndex]);
                    s.Write(',');
                }
                s.WriteLine(trackEvolution[4]);
                s.WriteLine(trackImprovement);
                s.WriteLine(PDFReader);
                s.WriteLine(timingDataBaseFolder);
            }
        }
        /// <summary>
        /// Reads the settings data from a file
        /// </summary>
        public void LoadData()
        {
            var dp = CultureInfo.InvariantCulture.NumberFormat;
            string[] readEvolution = new string[4];

            using (StreamReader r = new StreamReader(SettingsFile))
            {

```

```

        requiredPaceDelta = float.Parse(r.ReadLine(), dp);
        requiredSpeedDelta = float.Parse(r.ReadLine(), dp);
        timeLoss = float.Parse(r.ReadLine(), dp);
        backmarkerLoss = float.Parse(r.ReadLine(), dp);
        timeGap = float.Parse(r.ReadLine(), dp);
        defaultCompoundDelta = float.Parse(r.ReadLine(), dp);
        optionDegradation = float.Parse(r.ReadLine(), dp);
        primeDegradation = float.Parse(r.ReadLine(), dp);
        topSpeed = float.Parse(r.ReadLine(), dp);
        fuelEffect = float.Parse(r.ReadLine(), dp);
        pace = float.Parse(r.ReadLine(), dp);
        P2Fuel = float.Parse(r.ReadLine(), dp);
        dataFilePath = r.ReadLine();
        readEvolution = r.ReadLine().Split(',');

        for (int sessionIndex = 0; sessionIndex <= 4; sessionIndex++)
        {
            trackEvolution[sessionIndex] =
float.Parse(readEvolution[sessionIndex]);
        }
        trackImprovement = float.Parse(r.ReadLine(), dp);
        PDFReader = r.ReadLine();
        timingDataBaseFolder = r.ReadLine();
    }
}

using System;
using System.Collections.Generic;

namespace StratSim.Model
{
    public class Stint : ICloneable
    {
        //properties
        Driver driver;
        int session;
        public int startLap;
        public int stintLength;
        public TyreType tyreType;
        public bool modified;
        public List<float> lapTimes;

        /// <summary>
        /// Creates a new, empty instance of the stint class.
        /// </summary>
        /// <param name="_driver">The driver to link to the stint</param>
        /// <param name="passSession">The session in which the stint takes
place</param>
        /// <param name="passStartLap">The lap on which the stint starts</param>
        public Stint(Driver _driver, int passSession, int passStartLap)
        {
            driver = _driver;
            session = passSession;
            startLap = passStartLap;
            lapTimes = new List<float>();
            modified = false;
        }

        /// <summary>
        /// Creates a new instance of the stint class specifically for use in
strategies
        /// </summary>
        /// <param name="_driver">The driver to link to the stint</param>
        /// <param name="passStartLap">The lap on which the stint starts</param>
    }
}

```

```

    /// <param name="passTyreType">The tyre type used for the stint</param>
    /// <param name="_stintLength">The number of laps in the stint</param>
    public Stint(Driver _driver, int passStartLap, TyreType passTyreType, int
    _stintLength)
    {
        driver = _driver;
        session = 5;
        startLap = passStartLap;
        lapTimes = new List<float>();
        tyreType = passTyreType;
        stintLength = _stintLength;
        modified = false;
    }

    public Stint()
    { }

    public object Clone()
    {
        return this.MemberwiseClone();
    }

    public static Stint operator +(Stint stint, float lapTimeToAdd)
    {
        stint.AddLap(lapTimeToAdd);
        return stint;
    }

    public static List<Stint> operator +(List<Stint> sessionStints, Stint
    stintToAdd)
    {
        sessionStints.Add(stintToAdd);
        return sessionStints;
    }

    void AddLap(float lapTime)
    {
        lapTimes.Add(lapTime);
    }

    /// <summary>
    /// Writes data about the stint to the specified stream. Writing starts with
    the first
    /// data item and terminates on the same line.
    /// </summary>
    /// <param name="w">The streamwriter to write data to.</param>
    public void WriteStintData(ref System.IO.StreamWriter w)
    {
        w.Write(this.startLap);
        w.Write(',');
        w.Write(this.stintLength);
        w.Write(',');
        w.Write(this.tyreType);
    }

    //methods
    public float TotalTime()
    {
        float totalTime = 0;
        foreach (float l in lapTimes)
        {
            totalTime += l;
        }

        return totalTime;
    }

```

```

public float FastestLap()
{
    float fastestLap = Data.Settings.DefaultPace;
    foreach (float l in lapTimes)
    {
        if (l < fastestLap)
            fastestLap = l;
    }
    return fastestLap;
}

public float AverageLap()
{
    float sum = 0;
    int totalled = 0;
    int count = 0;

    foreach (float l in lapTimes)
    {
        if (count++ != 0)
        {
            sum += l;
            ++totalled;
        }
    }

    return (sum / totalled);
}

struct indexedLapInStint
{
    public float lapTime;
    public int lapIndex;
};

/// <returns>The average time reduction per lap due to any factor except track
evolution for the stints</returns>
public float AverageDegradation()
{
    indexedLapInStint previousLap, currentLap, lap;
    float fastestLap;
    float totalDegradation = 0;
    float degradation = 0;
    int lapsAssessed = 0;
    int lapToAnalyse = 0;

    if (lapTimes.Count >= 2)
    {
        fastestLap = this.FastestLap();

        currentLap.lapTime = Data.Settings.DefaultPace;
        currentLap.lapIndex = 0;

        do
        {
            lap.lapTime = lapTimes[lapToAnalyse];
            lap.lapIndex = lapToAnalyse++;

            if (lap.lapTime < fastestLap * 1.01)
            {
                previousLap = currentLap;
                currentLap = lap;

                ++lapsAssessed;

                if (lapsAssessed >= 2)
                {

```

```

        totalDegradation += (currentLap.lapTime - previousLap.lapTime -
Data.Settings.TrackImprovement);
    }

}
} while (lapToAnalyse < lapTimes.Count);

degradation = (totalDegradation / lapsAssessed);

if (Math.Abs(degradation) > fastestLap * 0.005) { degradation = 0; }
}
else
{
    degradation = 0;
}

return degradation;
}

//methods for adding, removing, and re-ordering stints

/// <summary>
/// Splits a stint at a given lap in the race
/// </summary>
/// <param name="splitLap">The race lap to split the stint at</param>
/// <returns>An array of the two generated stints, each with the same tyre
type</returns>
public Stint[] Split(int splitLap)
{
    Stint stintToSplit = this;
    Stint[] splitStints = new Stint[2];
    int firstStintLength, secondStintLength;

    firstStintLength = splitLap - stintToSplit.startLap;
    secondStintLength = stintToSplit.startLap + stintToSplit.stintLength -
splitLap;

    splitStints[0] = new Stint(stintToSplit.driver, stintToSplit.startLap,
stintToSplit.tyreType, firstStintLength);
    splitStints[1] = new Stint(stintToSplit.driver, splitLap,
stintToSplit.tyreType, secondStintLength);

    splitStints[0].modified = true;
    splitStints[1].modified = true;

    return splitStints;
}
/// <summary>
/// Concatenates two stints into one longer stint, with the tyre type of the
first stint
/// </summary>
/// <returns>The completed stint</returns>
public static Stint Merge(Stint a, Stint b) //returns a stint of the correct
length with no lap times included
{
    Stint mergedStint = a;
    int startLap = a.startLap;
    int endLap = startLap + a.stintLength + b.stintLength;

    mergedStint.lapTimes.Clear();
    mergedStint.stintLength = endLap - startLap;

    mergedStint.modified = true;

    return mergedStint;
}

```

```

    ///<summary>
    /// Swaps two stints orders and updates the start lap accordingly
    ///</summary>
    public static void Swap(ref Stint a, ref Stint b)
    {
        Stint tempStint;
        int stintAstartLap = a.startLap;

        tempStint = a;
        a = b;
        b = tempStint;

        a.startLap = stintAstartLap;
        b.startLap = stintAstartLap + a.stintLength;
    }
}

using StratSim.ViewModel;
using System;
using System.Collections.Generic;
using System.Linq;

namespace StratSim.Model
{
    ///<summary>
    ///<para>Contains data about a race strategy, including list of stints making up the strategy,</para>
    ///<para>the driver completing the stint, and the full list of lap times used for simulating the race.</para>
    ///<para>Contains methods for optimising the strategy and manipulating the stints within the strategy</para>
    ///<para>once it has been created.</para>
    ///</summary>
    public class Strategy : ICloneable
    {
        Driver driver;
        Track t;
        List<Stint> listOfStints = new List<Stint>();
        float[] lapTimes;
        float[] raceLapTimes;
        float totalTime;
        int noOfStints;
        int lapsInRace;
        int optionLaps;
        int primeLaps;
        float cumulativeTime;
        List<int> pitStops = new List<int>();

        ///<summary>
        /// Starts a strategy from driver data and stint information and optimises the strategy
        ///</summary>
        ///<param name="_noOfStints">The number of stints to complete</param>
        ///<param name="_primeStints">The number of prime tyre stints to be completed</param>
        ///<param name="_driverIndex">The index of the driver who is completing the stint</param>
        public Strategy(int _noOfStints, int _primeStints, Driver driver)
        {
            int primeStints = _primeStints;
            int optionStints = _noOfStints - _primeStints;

            int trackIndex = Data.RaceIndex;
        }
    }
}

```

```

        this.driver = driver;
        t = Data.Tracks[trackIndex];

        lapsInRace = Data.GetRaceLaps();
        lapTimes = new float[lapsInRace];

        SetStintLengths(lapsInRace, primeStints, optionStints);

        SetupStrategyStints(_noOfStints, primeStints);
        PopulateAllStints();
        UpdateStrategyParameters();
    }
    /// <summary>
    /// Starts a strategy linked to a driver with a list of pre-determined stints
    to load
    /// </summary>
    public Strategy(Driver driver, List<Stint> _stints)
    {
        lapsInRace = Data.GetRaceLaps();

        listOfStints = new List<Stint>(_stints);

        foreach (Stint s in listOfStints)
        {
            if (s.startLap != 0) { pitStops.Add(s.startLap - 1); }
        }

        t = Data.CurrentTrack;
        this.driver = driver;

        PopulateAllStints();
        UpdateStrategyParameters();
    }

    public Strategy()
    { }

    /// <summary>
    /// Starts a blank strategy linked to a driver
    /// </summary>
    /// <param name="driver">The driver to link to the strategy</param>
    public Strategy(Driver driver)
    {
        this.driver = driver;
    }

    /// <summary>
    /// Copy constructor for the strategy class.
    /// </summary>
    public Strategy(Strategy s)
    {
        driver = s.driver;
        t = s.t;
        lapsInRace = s.lapsInRace;
        optionLaps = s.optionLaps;
        primeLaps = s.primeLaps;

        foreach (Stint stint in s.Stints)
        {
            listOfStints.Add((Stint)stint.Clone());
        }

        PopulateAllStints();
        UpdateStrategyParameters();
    }
}

```

```

public object Clone()
{
    return this.MemberwiseClone();
}

/// <summary>
/// Sets the optimum prime and option stint lengths for the stint
/// </summary>
/// <param name="lapsInRace">The number of laps in the race being
simulated</param>
/// <param name="noOfOptionStints">The number of stints to be completed on the
prime tyre</param>
/// <param name="noOfPrimeStints">The number of stints to be completed on the
option tyre</param>
public void SetStintLengths(int lapsInRace, int noOfPrimeStints, int
noOfOptionStints)
{
    float optimumPrimeLength = OptimisePrime(driver, noOfPrimeStints,
noOfOptionStints, lapsInRace);
    float ratio = (float)noOfPrimeStints / (float)(noOfPrimeStints +
noOfOptionStints);
    primeLaps = RoundValueToIntegerBasedOnRatio(optimumPrimeLength, ratio);
    optionLaps = (lapsInRace - (noOfPrimeStints * primeLaps)) /
(noOfOptionStints);
}

/// <returns>The race time for the strategy</returns>
float GetStrategyTime()
{
    float totalStrategyTime = 0;

    foreach (Stint s in listOfStints)
    {
        totalStrategyTime += s.TotalTime();
    }

    return totalStrategyTime;
}

/// <summary>
/// Sets up the empty stints for the strategy
/// Sets option stints before prime stints by default
/// </summary>
/// <param name="numberOfStints">The number of stints to do in the
race</param>
/// <param name="numberOfPrimeStints">The number of stints to do on the prime
tyre0</param>
void SetupStrategyStints(int numberOfStints, int numberOfPrimeStints)
{
    Stint tempStint;
    int lapsThroughRace = 0;
    int stintLength = 0;

    //Populates option stints before prime stints
    for (int i = numberOfPrimeStints; i < numberOfStints; i++)
    {
        tempStint = new Stint(Driver, lapsThroughRace, TyreType.Option,
optionLaps);
        lapsThroughRace += optionLaps;
        listOfStints += tempStint;
    }
    for (int i = 0; i < numberOfPrimeStints; i++)
    {
        if (i == (numberOfPrimeStints - 1)) { stintLength = t.laps -
lapsThroughRace; }
        else { stintLength = primeLaps; }
    }
}

```

```

        tempStint = new Stint(Driver, lapsThroughRace, TyreType.Prime,
stintLength);
        lapsThroughRace += primeLaps;
        listOfStints += tempStint;
    }
}

/// <summary>
/// Populates the stints of the strategy with lap times
/// Populates the list of pit stops
/// </summary>
void PopulateAllStints()
{
    int lapsThroughRace = 0;
    int stintIndex = 0;
    int totalStints = listOfStints.Count - 1;
    pitStops.Clear();

    for (stintIndex = 0; stintIndex <= totalStints; stintIndex++)
    {
        listOfStints[stintIndex].modified = true;
        if (stintIndex != 0)
            pitStops.Add(lapsThroughRace);
        listOfStints[stintIndex] = PopulateSingleStint(listOfStints[stintIndex],
ref lapsThroughRace);
    }
}

/// <summary>
/// Populates a stint with lap times
/// </summary>
/// <param name="lapsThroughRace">The race lap on which the stint will start.
It is updated within the method</param>
/// <returns>The populated stint</returns>
Stint PopulateSingleStint(Stint stintToPopulate, ref int lapsThroughRace)
{
    float degradation;
    float lapTime;
    float tyreDelta = 0;

    stintToPopulate.lapTimes.Clear();
    degradation = driver.TyreDegradation[stintToPopulate.tyreType];

    if (stintToPopulate.tyreType == TyreType.Prime)
    {
        if (!stintToPopulate.modified)
            lapsThroughRace += primeLaps;
        tyreDelta = -driver.TyrePaceDelta;
    }
    else
    {
        if (!stintToPopulate.modified)
            lapsThroughRace += optionLaps;
    }

    if (stintToPopulate.modified && (stintToPopulate.stintLength != 0))
    { lapsThroughRace += stintToPopulate.stintLength; }

    float fuelAddedTime;
    float fuelTimePerLap = driver.FuelConsumptionPerLap *
driver.FuelEffectPerKilo;

    int lap;
    for (lap = 0; lap < stintToPopulate.stintLength; lap++)
    {
        fuelAddedTime = (lapsInRace - stintToPopulate.startLap - lap) *
(fuelTimePerLap);
    }
}

```

```

        lapTime = driver.LowFuelPace;
        lapTime += tyreDelta;
        lapTime += fuelAddedTime;
        lapTime += (lap * degradation);
        AddLapToStint(stintToPopulate, lapTime);
    }

    if (lapsThroughRace != lapsInRace)
    {
        stintToPopulate.lapTimes[lap - 1] += t.pitStopLoss;
    }

    return stintToPopulate;
}

/// <summary>
/// Sets the total strategy time and the list of lap times used for the
strategy,
/// and calculates the number of stints in the strategy.
/// Sorts the pit stops using a quicksort.
/// </summary>
public void UpdateStrategyParameters()
{
    totalTime = GetStrategyTime();
    lapTimes = SetLapTimes(true);
    noOfStints = listOfStints.Count();
    raceLapTimes = SetLapTimes(false);

    //Sort pit stops
    int[] pitStopList = pitStops.ToArray<int>();
    Functions.QuickSort<int>(ref pitStopList, 0, pitStopList.Length - 1, ((a,
b) => a > b));
    pitStops = new List<int>(pitStopList);
}

/// <summary>
/// Populates an array of lap times for every lap of the race
/// Pulls data from the stints to create a one dimensional array
/// </summary>
/// <param name="pitStopsIncluded">Represents whether pit stop times are
included in the lap times</param>
/// <returns>An array, the length of the race, containing the lap times for
the entire race.</returns>
float[] SetLapTimes(bool pitStopsIncluded)
{
    float[] times = new float[lapsInRace];
    int lapNumber = 0;

    foreach (Stint s in listOfStints)
    {
        foreach (float lap in s.lapTimes)
        {
            times[lapNumber++] = lap;
        }
        if (!pitStopsIncluded)
        {
            if (lapNumber != lapsInRace) { times[lapNumber - 1] -= t.pitStopLoss;
}
        }
    }
}

return times;
}

float OptimisePrime(Driver d, int primeStints, int optionStints, int
lapsInRace)
{

```

```

        int l = lapsInRace;
        float optionD = d.TyreDegradation[TyreType.Option];
        float primeD = d.TyreDegradation[TyreType.Prime];
        float delta = d.TyrePaceDelta;
        float optimumLength = 0;

        optimumLength += (l * optionD / (float)optionStints) + (delta) + (primeD /
2) - (optionD / 2);
        optimumLength /= ((primeD) + (optionD * (float)primeStints /
(float)optionStints));

        return optimumLength;
    }

    void AddLapToStint(Stint stintToAdd, float lapTime)
    {
        stintToAdd += lapTime;
    }

    /// <summary>
    /// Decides if it is optimal to round a stint up or down
    /// </summary>
    /// <param name="value">The value to round</param>
    /// <param name="ratio">The ratio below which the value will be rounded
    down</param>
    int RoundValueToIntegerBasedOnRatio(float value, float ratio)
    {
        float remainder = (value % 1);
        bool roundUp = (remainder > ratio);
        int roundedLength;

        roundedLength = (int)(value - remainder);
        if (roundUp) { ++roundedLength; }

        return roundedLength;
    }

    /// <summary>
    /// Changes the length of a stint in the strategy
    /// </summary>
    /// <returns>The full list of stints in the new strategy</returns>
    public List<Stint> ChangeStintLength(int stintToChange, int newLength)
    {
        List<Stint> newListOfStints = listOfStints;

        int originalLength, lengthChange;
        originalLength = newListOfStints[stintToChange].stintLength;
        lengthChange = newLength - originalLength;

        if (stintToChange == noOfStints - 1)
        {
            newListOfStints[stintToChange].stintLength += lengthChange;
            newListOfStints[stintToChange - 1].stintLength -= lengthChange;

            pitStops.Remove(newListOfStints[stintToChange].startLap);
            newListOfStints[stintToChange].startLap += lengthChange;
            pitStops.Add(newListOfStints[stintToChange].startLap);

            int lapsThroughRace = newListOfStints[stintToChange - 1].startLap;

            newListOfStints[stintToChange - 1] =
PopulateSingleStint(newListOfStints[stintToChange - 1], ref lapsThroughRace);
            newListOfStints[stintToChange] =
PopulateSingleStint(newListOfStints[stintToChange], ref lapsThroughRace);
        }
    }
}

```

```

{
    newListOfStints[stintToChange].stintLength += lengthChange;
    newListOfStints[stintToChange + 1].stintLength -= lengthChange;

    pitStops.Remove(newListOfStints[stintToChange + 1].startLap);
    newListOfStints[stintToChange + 1].startLap += lengthChange;
    pitStops.Add(newListOfStints[stintToChange + 1].startLap);

    int lapsThroughRace = newListOfStints[stintToChange].startLap;

    newListOfStints[stintToChange] =
        PopulateSingleStint(newListOfStints[stintToChange], ref lapsThroughRace);
    newListOfStints[stintToChange + 1] =
        PopulateSingleStint(newListOfStints[stintToChange + 1], ref lapsThroughRace);
}

return newListOfStints;
}

public void ChangeStintTyreType(int stintToChange, TyreType newTyreType)
{
    listOfStints[stintToChange].tyreType = newTyreType;

    int lapsThroughRace = listOfStints[stintToChange].startLap;
    listOfStints[stintToChange] = PopulateSingleStint(Stints[stintToChange],
ref lapsThroughRace);

    UpdateStrategyParameters();
    MyEvents.OnStrategyModified(Driver, this, false);
}

public List<Stint> AddPitStop(int lap)
{
    List<Stint> newListOfStints = new List<Stint>();

    int splitStintIndex = listOfStints.FindLastIndex(s => s.startLap < lap -
1);
    if (splitStintIndex >= 0)
    {
        Stint stintToSplit = listOfStints[splitStintIndex];

        int totalStints = listOfStints.Count;
        int lapsThroughRace = 0;

        Stint[] splitStint = stintToSplit.Split(lap);

        //re-populate strategy:
        int stintIndex = 0;
        while (stintIndex < splitStintIndex)
        {
            newListOfStints += listOfStints[stintIndex];
            lapsThroughRace += listOfStints[stintIndex].stintLength;
            stintIndex++;
        }

        foreach (Stint s in splitStint)
        {
            newListOfStints += PopulateSingleStint(s, ref lapsThroughRace);
        }

        stintIndex++;

        while (stintIndex < totalStints)
        {
            newListOfStints += listOfStints[stintIndex];
            lapsThroughRace += listOfStints[stintIndex].stintLength;
            stintIndex++;
        }
    }
}

```

```

        }

        pitStops.Add(lap);
    }
    else
    {
        newListOfStints = listOfStints;
    }

    return newListOfStints;
}

public List<Stint> RemovePitStop(int lap)
{
    List<Stint> newListOfStints = new List<Stint>();
    Stint mergedStint;

    int removeStintIndex = listOfStints.FindIndex(s => s.startLap == lap);

    if (removeStintIndex >= 0)
    {
        if (removeStintIndex == 0)
        {
            mergedStint = Stint.Merge(listOfStints[removeStintIndex],
listOfStints[removeStintIndex + 1]);
        }
        else
        {
            mergedStint = Stint.Merge(listOfStints[removeStintIndex - 1],
listOfStints[removeStintIndex]);
        }
    }

    int totalStints = listOfStints.Count;
    int lapsThroughRace = 0;

    int stintIndex = 0;
    while (stintIndex < removeStintIndex - 1)
    {
        newListOfStints += listOfStints[stintIndex];
        lapsThroughRace += listOfStints[stintIndex].stintLength;
        stintIndex++;
    }

    newListOfStints += PopulateSingleStint(mergedStint, ref
lapsThroughRace);
    stintIndex += 2;

    while (stintIndex < totalStints)
    {
        newListOfStints += listOfStints[stintIndex];
        lapsThroughRace += listOfStints[stintIndex].stintLength;
        stintIndex++;
    }

    pitStops.Remove(lap);
}
else
{
    newListOfStints = listOfStints;
}

return newListOfStints;
}

public int GetNearestPitStop(int lapNumber, int precision, out bool
withinRange)
{

```

```

        int nearestPitStopLap = -1;
        withinRange = false;

        foreach (int stop in pitStops)
        {
            if (stop + precision >= lapNumber && stop - precision <= lapNumber)
            {
                withinRange = true;
                nearestPitStopLap = stop;
            }
        }

        return nearestPitStopLap;
    }

    public void SwapStints(int stintIndexA, int stintIndexB)
    {
        Stint stintA = listOfStints[stintIndexA];
        Stint stintB = listOfStints[stintIndexB];

        int lapsThroughRace = stintA.startLap;

        Stint.Swap(ref stintA, ref stintB);

        listOfStints[stintIndexA] = PopulateSingleStint(stintA, ref
lapsThroughRace);
        listOfStints[stintIndexB] = PopulateSingleStint(stintB, ref
lapsThroughRace);
    }

    public List<Stint> Stints
    {
        get { return listOfStints; }
        set { listOfStints = value; }
    }

    public Driver Driver
    {
        get { return driver; }
        set { driver = value; }
    }

    public int NoOfStints
    {
        get { return noOfStints; }
        set { noOfStints = value; }
    }

    public int NoOfStops
    {
        get { return noOfStints - 1; }
    }

    public float TotalTime
    {
        get { return totalTime; }
    }

    public float[] LapTimes
    {
        get { return lapTimes; }
        set { lapTimes = value; }
    }

    public float[] RaceLapTimes
    {

```

```

        get { return raceLapTimes; }
        set { raceLapTimes = value; }
    }
    public int PrimeLaps
    { get { return primeLaps; } }
    public int OptionLaps
    { get { return optionLaps; } }

    public float CumulativeTime
    {
        get { return cumulativeTime; }
        set { cumulativeTime = value; }
    }

    public float FuelUsed
    { get { return lapsInRace * driver.FuelConsumptionPerLap; } }

    public List<int> PitStops
    {
        get { return pitStops; }
        set { pitStops = value; }
    }
}

namespace StratSim.Model
{
    public class Track
    {
        //properties
        public int roundNumber;
        public string name;
        public int laps;
        public float fuelPerLap;
        public float pitStopLoss;

        /// <summary>
        /// Creates a new instance of the track class
        /// </summary>
        /// <param name="Name">The name of the track (normally the country)</param>
        /// <param name="Laps">The laps in the race</param>
        /// <param name="fuelConsumption">The fuel use in kg/lap for the track</param>
        /// <param name="_pitStopLoss">The time loss due to a pit stop</param>
        /// <param name="index">The zero-based position in the year of the
race</param>
        public Track(string Name, int Laps, float fuelConsumption, float _pitStopLoss,
int index)
        {
            name = Name;
            laps = Laps;
            fuelPerLap = fuelConsumption;
            pitStopLoss = _pitStopLoss;
            roundNumber = index + 1;
        }

        /// <summary>
        /// Loads all tracks from the database
        /// </summary>
        /// <param name="numberOfTracksInSeason">A variable to which the number of
tracks found is assigned</param>
        /// <returns>The populated array of tracks</returns>
        public static Track[] InitialiseTracks(out int numberOfTracksInSeason)
        {
            Track[] arrayOfTracks;
            int numberOfTracks;
        }
    }
}

```

```
        var stratSimDataSet = new StratSimDataSet();
        var tracksTableAdapter = new
StratSimDataSetTableAdapters.TracksTableAdapter();
        tracksTableAdapter.Fill(stratSimDataSet.Tracks);
        StratSimDataSet.TracksRow row;

        numberOfTracks = stratSimDataSet.Tracks.Count;
        arrayOfTracks = new Track[numberOfTracks];

        for (int raceIndex = 0; raceIndex < numberOfTracks; raceIndex++)
{
    row = stratSimDataSet.Tracks[raceIndex];
    string name = row.Country;
    int laps = row.Laps;
    float fuelConsumption = row.FuelConsumption;
    float pitLoss = row.PitStopLoss;
    arrayOfTracks[raceIndex] = new Track(name, laps, fuelConsumption,
pitLoss, raceIndex);
}

        numberOfTracksInSeason = numberOfTracks;
        return arrayOfTracks;
    }
}
}
```

StratSim.View.Forms

```

using System;
using System.ComponentModel;
using System.Drawing;
using System.Reflection;
using System.Windows.Forms;

namespace StratSim.View.Forms
{
    /// <summary>
    /// A form displaying information about the current system version and
    /// copyrights.
    /// </summary>
    public partial class VersionWindow : Form
    {
        Designer generated code for initialising component and displaying objects removed

        public VersionWindow()
        {
            InitializeComponent();
            this.Text = String.Format("About {0}", AssemblyTitle);
            this.labelProductName.Text = AssemblyProduct;
            this.labelVersion.Text = String.Format("Version {0}", AssemblyVersion);
            this.labelCopyright.Text = AssemblyCopyright;
            this.labelCompanyName.Text = AssemblyCompany;
            this.textBoxDescription.Text = AssemblyDescription;
            this.ShowIcon = true;

            ComponentResourceManager resources = new
            ComponentResourceManager(typeof(VersionWindow));
            this.Icon = (Icon)resources.GetObject("$this.Icon");
        }

        public string AssemblyTitle
        {
            get
            {
                object[] attributes =
                Assembly.GetExecutingAssembly().GetCustomAttributes(typeof(AssemblyTitleAttribute),
                false);
                if (attributes.Length > 0)
                {
                    AssemblyTitleAttribute titleAttribute =
                    ( AssemblyTitleAttribute ) attributes[0];
                    if (titleAttribute.Title != "")
                    {
                        return titleAttribute.Title;
                    }
                }
                return
                System.IO.Path.GetFileNameWithoutExtension(Assembly.GetExecutingAssembly().CodeBase);
            }
        }

        public string AssemblyVersion
        {
            get
            {
                return Assembly.GetExecutingAssembly().GetName().Version.ToString();
            }
        }

        public string AssemblyDescription
        {
            get

```

```

    {
        object[] attributes =
Assembly.GetExecutingAssembly().GetCustomAttributes(typeof(AssemblyDescriptionAttribute), false);
        if (attributes.Length == 0)
        {
            return "";
        }
        return ((AssemblyDescriptionAttribute)attributes[0]).Description;
    }
}

public string AssemblyProduct
{
    get
    {
        object[] attributes =
Assembly.GetExecutingAssembly().GetCustomAttributes(typeof(AssemblyProductAttribute), false);
        if (attributes.Length == 0)
        {
            return "";
        }
        return ((AssemblyProductAttribute)attributes[0]).Product;
    }
}

public string AssemblyCopyright
{
    get
    {
        object[] attributes =
Assembly.GetExecutingAssembly().GetCustomAttributes(typeof(AssemblyCopyrightAttribute), false);
        if (attributes.Length == 0)
        {
            return "";
        }
        return ((AssemblyCopyrightAttribute)attributes[0]).Copyright;
    }
}

public string AssemblyCompany
{
    get
    {
        object[] attributes =
Assembly.GetExecutingAssembly().GetCustomAttributes(typeof(AssemblyCompanyAttribute), false);
        if (attributes.Length == 0)
        {
            return "";
        }
        return ((AssemblyCompanyAttribute)attributes[0]).Company;
    }
}

private void okButton_Click(object sender, EventArgs e)
{
    this.Hide();
}
}
}

```

StratSim.View.MyFlowLayout

```

using StratSim.ViewModel;
using System;
using System.Collections.Generic;
using System.Drawing;
using System.Windows.Forms;

namespace StratSim.View.MyFlowLayout
{
    public class ContentTabControl : MyPanel
    {
        TabControl MainControls;

        List<MyPanel> PanelsToDisplay = new List<MyPanel>();
        List<TabPage> TabPages = new List<TabPage>();

        ContextMenuStrip thisContextMenu;
        ToolStripDropDownButton closePanels;

        public ContentTabControl(MainForm FormToAdd)
            : base(400, 300, "Main", FormToAdd, Properties.Resources.Main)
        {
            FlowLayoutEvents.MainPanelResizeEnd += ResizeMainControl;

            MainControls = new TabControl();
            this.Controls.Add(MainControls);

            MainControls.ControlAdded += MainControls_ControlAdded;
            MainControls.ControlRemoved += MainControls_ControlRemoved;
            OpenedInNewForm += ContentTabControl_OpenedInNewForm;
            MyEvents.RemoveContentPanel += MyEvents_RemoveContentPanel;

            SetupContextMenu();

            SetPanelProperties(DockTypes.Top, AutosizeTypes.Free, FillStyles.None,
this.Size);
        }

        void MyEvents_RemoveContentPanel(int PanelIndex)
        {
            closePanels.DropDownItems.RemoveAt(PanelIndex);
        }

        void ContentTabControl_OpenedInNewForm(MainForm NewForm)
        {
            foreach (MyPanel p in PanelsToDisplay)
            {
                p.Form = NewForm;
            }
            NewForm.IOController.SetContentTabControl(this);
        }

        void MainControls_ControlRemoved(object sender, ControlEventArgs e)
        {
            MainControls.SelectedIndex = MainControls.TabCount - 1;
        }

        void MainControls_ControlAdded(object sender, ControlEventArgs e)
        {
            MainControls.SelectedIndex = MainControls.TabCount - 1;
        }

        void SetupContextMenu()
        {
            closePanels = new ToolStripDropDownButton("Close");
        }
    }
}

```

```

thisContextMenuStrip = new ContextMenuStrip();
thisContextMenuStrip.Items.Add(closePanels);

MyToolStripButton tempButton;
int index = 0;
foreach (MyPanel p in PanelsToDisplay)
{
    tempButton = new MyToolStripButton(index++);
    tempButton.Text = p.Name;
    tempButton.ButtonClicked += tempButton_CustomCheckedChanged;
    closePanels.DropDownItems.Add(tempButton);
}
this.ContextMenuStrip = thisContextMenuStrip;
ContextMenuStrip.Width = 100;
}

void tempButton_CustomCheckedChanged(int buttonIndex, Type buttonType)
{
    this.RemovePanelFromControl(PanelsToDisplay[buttonIndex]);
    thisContextMenuStrip.Items.Remove(closePanels.DropDownItems[buttonIndex]);
    closePanels.DropDownItems.RemoveAt(buttonIndex);
    foreach (MyToolStripButton t in closePanels.DropDownItems)
    {
        if (t.ButtonIndex > buttonIndex)
        {
            t.ButtonIndex--;
            PanelsToDisplay[t.ButtonIndex].PanelIndex--;
        }
    }
}

void ResizeMainControl()
{
    MainControls.Size = new Size(this.Width - MyPadding.Horizontal, this.Height
- MyPadding.Vertical);
    MainControls.Location = new Point(MyPadding.Left, MyPadding.Top);

    foreach (TabPage p in TabPages)
    {
        p.Size = MainControls.ClientSize;
    }

    foreach (MyPanel p in PanelsToDisplay)
    {
        if (p.Parent == null) { p.Size = MainControls.ClientSize; }
        else { p.Size = p.Parent.ClientSize; }
        p.PositionComponents();
    }
}

void DisplayControls()
{
    int visiblePanels = 0;
    TabPages.Clear();
    MainControls.TabPages.Clear();

    foreach (MyPanel p in PanelsToDisplay)
    {
        if (p.MyVisible == true) { ++visiblePanels; }
    }

    if (visiblePanels == 0)
    {
        this.Visible = false;
    }
    if (visiblePanels == 1)
    {
}
}

```

```

        MainControls.Visible = false;
        foreach (MyPanel p in PanelsToDisplay)
        {
            if (p.MyVisible == true)
            {
                p.Location = new Point(MyPadding.Left, MyPadding.Top);
                p.Size = new Size(this.Width - MyPadding.Horizontal, this.Height -
MyPadding.Vertical);
            }
            this.Controls.Add((Control)p);
        }
    }
    if (visiblePanels > 1)
    {
        MainControls.Visible = true;
        int panelsAdded = 0;
        foreach (MyPanel p in PanelsToDisplay)
        {
            if (p.MyVisible == true)
            {
                TabPage tempTabPage = new TabPage(p.Name);
                tempTabPage.Controls.Add(p);

                p.Location = new Point(0, 0);
                p.Size = new Size(MainControls.DisplayRectangle.Width,
MainControls.DisplayRectangle.Height);

                TabPages.Add(tempTabPage);

                MainControls.TabPages.Add(TabPages[panelsAdded++]);
            }
        }
    }
}

/// <summary>
/// Adds a panel to the tab control and re-draws the control.
/// </summary>
/// <param name="PanelToAdd">The panel to be added to the control.</param>
public void AddPanelToTabControl(MyPanel PanelToAdd)
{
    if (!PanelToAdd.InContentPanel)
    {
        PanelsToDisplay.Add(PanelToAdd);
        PanelToAdd.RemoveToolbarButtons();
        PanelToAdd.PanelIndex = PanelsToDisplay.Count - 1;
        PanelToAdd.InContentPanel = true;

        //Add tool strip button to close the panel.
        MyToolStripButton ButtonToAdd;
        ButtonToAdd = new MyToolStripButton(PanelsToDisplay.Count - 1);
        ButtonToAdd.Text = PanelToAdd.Name;
        ButtonToAdd.ButtonClicked += tempButton_CustomCheckedChanged;
        closePanels.DropDownItems.Add(ButtonToAdd);

        DisplayControls();
    }
}

/// <summary>
/// Removes a panel from the content tab control, triggers the PanelClosed
event and re-draws the control.
/// </summary>
/// <param name="PanelToRemove">The panel to be removed from the tab
control</param>
public void RemovePanelFromControl(MyPanel PanelToRemove)
{

```

```

        if (PanelToRemove.InContentPanel)
        {
            PanelsToDisplay.Remove(PanelToRemove);
            PanelToRemove.InContentPanel = false;
            DisplayControls();
        }
    }
}

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Drawing;
using System.Windows.Forms;

using StratSim.ViewModel;

namespace StratSim.View.MyFlowLayout
{
    enum DragState { Drag, Drop }

    /// <summary>
    /// Provides a panel and methods for displaying and controlling
    /// dynamic control re-ordering.
    /// </summary>
    class DragDropController : Panel
    {
        List<MyPanel> visiblePanels;
        MyPanel panelBeingRelocated;
        Point startLocation;

        Dictionary<DockTypes, Point> dockPoints;

        DragState dragState;
        DockTypes dockPointToHighlight;

        List<Rectangle> rectanglesToDelete = new List<Rectangle>();
        Rectangle panelRectangle;

        MainForm parentForm;

        public DragDropController(MainForm Form)
        {
            parentForm = Form;
            this.DoubleBuffered = true;
        }

        /// <summary>
        /// Starts the dynamic drag-drop process on the form by displaying the drag-
        drop interface.
        /// Events are subscribed to so that when the panel is clicked again the panel
        is dropped.
        /// </summary>
        /// <param name="panelBeingDragged">The panel that has been selected</param>
        /// <param name="visibleControls">A list of all visible panels on the
        form</param>
        /// <param name="startLocation">The initial click location</param>
        /// <param name="dockPoints">A dictionary of the available points the panel
        can be docked to on the form</param>
        public void StartDragDropLayout(MyPanel panelBeingDragged, List<MyPanel>
visibleControls, Point startLocation, Dictionary<DockTypes, Point> dockPoints)
        {
            this.visiblePanels = visibleControls;

```

```

        this.panelBeingRelocated = panelBeingDragged;
        this.startLocation = startLocation;
        this.dockPoints = dockPoints;

        MouseMove += DragDropController_MouseMove;
        MouseUp += DragDropController_MouseUp;
        FlowLayoutEvents.PanelDrag += MyEvents_PanelDrag;

        Program.AddEventToForms(DropPanelOnCurrentForm);

        rectanglesToDelete.Clear();

        dragState = DragState.Drag;

        StartLayout();

        parentForm.IOController.ShowDynamicLayoutPanel(this);
    }

    void DragDropController_MouseUp(object sender, MouseEventArgs e)
    {
        dragState = DragState.Drop;
        DropPanel(e.Location);
    }

    void DragDropController_MouseMove(object sender, MouseEventArgs e)
    {
        if (dragState == DragState.Drag)
            FlowLayoutEvents.OnPanelDrag(e.Location);
    }

    void MyEvents_PanelDrag(Point newPoint)
    {
        panelRectangle.Location = new Point(newPoint.X, newPoint.Y);
        dockPointToHighlight = FindNearestDockPoint(newPoint, false);
        Invalidate();
    }

    void StartLayout()
    {
        foreach (MyPanel p in visiblePanels)
        {
            if (p == panelBeingRelocated)
            {
                panelRectangle = new Rectangle(p.Location, p.OriginalSize);
            }
            else
            {
                rectanglesToDelete.Add(new Rectangle(p.Location, p.Size));
            }
        }

        Invalidate();
    }

    /// <summary>
    /// Unsubscribes from events and hides the layout panel, returning to the
    original window state.
    /// </summary>
    void FinishLayout()
    {
        parentForm.IOController.HideDynamicLayoutPanel(this);

        MouseMove -= DragDropController_MouseMove;
        MouseUp -= DragDropController_MouseUp;
        FlowLayoutEvents.PanelDrag -= MyEvents_PanelDrag;
        Program.RemoveEventFromForms(DropPanelOnCurrentForm);
    }
}

```

```

}

/// <summary>
/// Drops the panel at the nearest dock point to the point where it has been
dropped.
/// </summary>
/// <param name="LocationToDrop">A point representing the coordinates of the
location that the panel was dropped at</param>
void DropPanel(Point LocationToDrop)
{
    //Setting the dock type fires the re-layout event
    panelBeingRelocated.DockType = FindNearestDockPoint(LocationToDrop, true);
    FinishLayout();
}

void DropPanelOnCurrentForm(MainForm formPanelIsDroppedOn)
{
    dragState = DragState.Drop;
    parentForm.IOController.OpenPanelInCurrentForm(panelBeingRelocated,
formPanelIsDroppedOn);

    FinishLayout();
}

DockTypes FindNearestDockPoint(Point p, bool positionPanels)
{
    DockTypes nearestDockType = panelBeingRelocated.DockType;

    int lowestDifference = 0;
    int currentDifference = 0;

    foreach (var dockType in (DockTypes[])Enum.GetValues(typeof(DockTypes)))
    {
        currentDifference = (int)Math.Pow((dockPoints[dockType].X - p.X), 2) +
(int)Math.Pow((dockPoints[dockType].Y - p.Y), 2);

        if ((int)dockType == 0)
            lowestDifference = currentDifference;

        if (currentDifference <= lowestDifference)
        {
            lowestDifference = currentDifference;
            nearestDockType = dockType;
        }
    }

    //If not near any dock points, the panel will be opened in a new window.
    //The dock point is set to the top left so the panel appears full screen.
    if (lowestDifference >= 65000)
    {
        if (positionPanels)
        {
            Program.OpenInNewWindow(panelBeingRelocated, parentForm);
        }
        nearestDockType = DockTypes.TopLeft;
    }
}

return nearestDockType;
}

protected override void OnPaint(PaintEventArgs e)
{
    var g = e.Graphics;
    var unusedPanelsPen = new Pen(Color.Black, 1F);
    var locatePanelsPen = new Pen(Color.Red, 1F);
    var dockPointsPen = new Pen(Color.Blue, 2F);
    var highlightPointsPen = new Pen(Color.LightBlue, 2F);
}

```

```

        base.OnPaint(e);

        foreach (var dockType in (DockTypes[])Enum.GetValues(typeof(DockTypes)))
        {
            g.DrawEllipse((dockType == dockPointToHighlight ? highlightPointsPen : dockPointsPen), (int)dockPoints[dockType].X - 10, (int)dockPoints[dockType].Y - 10, 20, 20);
        }

        foreach (Rectangle r in rectanglesToDelete)
        {
            g.DrawRectangle(unusedPanelsPen, r);
        }
        g.DrawRectangle(locatePanelsPen, panelRectangle);

        unusedPanelsPen.Dispose();
        locatePanelsPen.Dispose();
        dockPointsPen.Dispose();
        highlightPointsPen.Dispose();
    }

}

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Drawing;

namespace StratSim.View.MyFlowLayout
{
    /// <summary>
    /// Class controlling events when the window requires layout
    /// </summary>
    class FlowLayoutEvents
    {
        public FlowLayoutEvents()
        { }

        public static void OnMainPanelLayoutChanged()
        {
            if (MainPanelResizeStart != null)
                MainPanelResizeStart();
            MainPanelLayoutChanged();
        }

        public static void OnMainPanelResize()
        {
            if (MainPanelResizeEnd != null)
                MainPanelResizeEnd();
        }

        public delegate void PanelLayoutEventHandler();
        public static event PanelLayoutEventHandler MainPanelLayoutChanged;
        public static event PanelLayoutEventHandler MainPanelResizeStart;
        public static event PanelLayoutEventHandler MainPanelResizeEnd;

        public static void OnPanelDrag(Point newPoint)
        {
            PanelDrag(newPoint);
        }
    }
}

```

```

        public delegate void PanelDragEventhandler(Point newPoint);
        public static event PanelDragEventhandler PanelDrag;
    }

}

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Drawing;

namespace StratSim.View.MyFlowLayout
{
    public interface IDockableControl
    {
        FillStyles FillStyle { get; set; }
        AutosizeTypes AutoSizeType { get; set; }
        DockTypes DockType { get; set; }
        Size OriginalSize { get; set; }
        Point DockPointLocation { get; set; }
        Type Type { get; }
        bool MyVisible { get; set; }
        bool Removed { get; set; }
    }
}

using System;
using System.Collections.Generic;
using System.Drawing;
using System.Windows.Forms;

namespace StratSim.View.MyFlowLayout
{
    public enum LineDirection { Vertical, Horizontal };
    public enum Locations { Top, Left, Bottom, Right };

    /// <summary>
    /// Contains data about a straight horizontal or vertical line
    /// Contains methods for adjusting the length or position of the line.
    /// </summary>
    public class Line
    {
        Point startPoint;
        Point endPoint;

        /// <summary>
        /// Creates a new instance of the line class that is either horizontal or
        vertical
        /// </summary>
        /// <param name="start">The start location of the line, on the axis parallel
        to the line direction</param>
        /// <param name="end">The end location of the line, on the axis parallel to
        the line direction</param>
        /// <param name="staticPosition">The location of the line on the axis
        perpendicular to the line direction</param>
        /// <param name="direction">The direction in which the line is
        oriented</param>
        public Line(int start, int end, int staticPosition, LineDirection direction)
    }
}

```

```

{
    if (direction == LineDirection.Horizontal)
    {
        if (start < end)
        {
            startPoint = new Point(start, staticPosition);
            endPoint = new Point(end, staticPosition);
        }
        else
        {
            startPoint = new Point(end, staticPosition);
            endPoint = new Point(start, staticPosition);
        }
    }
    else
    {
        if (start < end)
        {
            startPoint = new Point(staticPosition, start);
            endPoint = new Point(staticPosition, end);
        }
        else
        {
            startPoint = new Point(staticPosition, end);
            endPoint = new Point(staticPosition, start);
        }
    }
}
public Line() { }
/// <summary>
/// Copy constructor for the line class
/// </summary>
public Line(Line l)
{
    startPoint = l.startPoint;
    endPoint = l.endPoint;
}

public void Draw(Graphics g, Color colour)
{
    var Pen = new Pen(colour, 5);
    g.DrawLine(Pen, startPoint, endPoint);
    Pen.Dispose();
}

public void setStartPointY(int value)
{ startPoint.Y = value; }
public void setStartPointX(int value)
{ startPoint.X = value; }
public void setEndPointY(int value)
{ endPoint.Y = value; }
public void setEndPointX(int value)
{ endPoint.X = value; }

public Point StartPoint
{
    get { return startPoint; }
    set { startPoint = value; }
}
public Point EndPoint
{
    get { return endPoint; }
    set { endPoint = value; }
}
public int Length
{
    get

```

```

        {
            if (endPoint.X == startPoint.X)
            { return (endPoint.Y - startPoint.Y); }
            else
            { return (endPoint.X - startPoint.X); }
        }
    }
    public int StaticLocation
    {
        get
        {
            if (endPoint.X == startPoint.X)
            { return endPoint.X; }
            else
            { return startPoint.Y; }
        }
        set
        {
            if (endPoint.X == startPoint.X)
            { startPoint.X = endPoint.X = value; }
            else
            { startPoint.Y = endPoint.Y = value; }
        }
    }
    public LineDirection Direction
    {
        get
        {
            if (endPoint.X == startPoint.X)
            { return LineDirection.Vertical; }
            else
            { return LineDirection.Horizontal; }
        }
    }
}

/// <summary>
/// Contains a set of lines bounding the panels on the form, used for re-sizing
the panels on a window flow panel.
/// Contains methods for adjusting the lines when panels are added, and
/// methods for finding the closest and furthest lines, allowing panels to be
resized.
/// </summary>
class LayoutLines
{
    Dictionary<Locations, List<Line>> lines;

    public LayoutLines(Rectangle container)
    {
        lines = new Dictionary<Locations, List<Line>>();
        foreach (var Location in (Locations[])Enum.GetValues(typeof(Locations)))
        {
            lines[Location] = new List<Line>();
        }
        SetupLinesAroundContainer(container);
    }

    void SetupLinesAroundContainer(Rectangle container)
    {
        foreach (var Location in (Locations[])Enum.GetValues(typeof(Locations)))
        {
            lines[Location].Clear();
        }
    }

    lines[Locations.Left].Add(new Line(container.Top, container.Bottom,
    container.Left, LineDirection.Vertical));
}

```

```

        lines[Locations.Top].Add(new Line(container.Left, container.Right,
container.Top, LineDirection.Horizontal));
        lines[Locations.Right].Add(new Line(container.Top, container.Bottom,
container.Right, LineDirection.Vertical));
        lines[Locations.Bottom].Add(new Line(container.Left, container.Right,
container.Bottom, LineDirection.Horizontal));
    }

    /// <summary>
    /// <para>Finds the indices of lines that are intersected by edges of a
panel.</para>
    /// <para>These lines are the lines that must be re-sized to allow the panel
to re-size in the given direction</para>
    /// </summary>
    /// <param name="panelToBeSized">The control that is to be re-sized on the
form</param>
    /// <param name="directionToLookIn">The direction the panel is to be resized
in</param>
    /// <returns>An array of two indices, in ascending order, of the indices of
the lines that restrict panel sizing</returns>
    int[] GetIndicesOfLinesRestrictingPanel(Control panelToBeSized, Locations
directionToLookIn)
    {
        int[] indicesToReturn = new int[2];
        int currentIndex = 0;
        int lastIndex = 0;
        bool firstIndexAllocated = false;

        //Find the set of lines to check for the line that restricts the panel
        switch (directionToLookIn)
        {
            case Locations.Top:
            {
                //Check all the lines
                foreach (Line l in lines[directionToLookIn])
                {
                    //if the line is the first one found with an end point after
                    the left of the panel to be sized:
                    if (!firstIndexAllocated && (l.EndPoint.X >
panelToBeSized.Left))
                    {
                        indicesToReturn[0] = currentIndex;
                        firstIndexAllocated = true;
                    }
                    //if the line start point is before the right side of the panel
                    if (l.StartPoint.X < panelToBeSized.Right)
                    {
                        lastIndex = currentIndex;
                    }

                    currentIndex++;
                }
                //the last index found before the right side of the panel is set
                as the second bounding index
                indicesToReturn[1] = lastIndex;
                break;
            }
            case Locations.Left:
            {
                foreach (Line l in lines[directionToLookIn])
                {
                    if (!firstIndexAllocated && (l.EndPoint.Y >
panelToBeSized.Top))
                    {
                        indicesToReturn[0] = currentIndex;
                        firstIndexAllocated = true;
                    }
                }
            }
        }
    }
}

```

```

        if (l.StartPoint.Y < panelToBeSized.Bottom)
        {
            lastIndex = currentIndex;
        }

        currentIndex++;
    }
    indicesToReturn[1] = lastIndex;
    break;
}
case Locations.Bottom:
{
    foreach (Line l in lines[directionToLookIn])
    {
        if (!firstIndexAllocated && (l.EndPoint.X >
panelToBeSized.Left))
        {
            indicesToReturn[0] = currentIndex;
            firstIndexAllocated = true;
        }
        if (l.StartPoint.X < panelToBeSized.Right)
        {
            lastIndex = currentIndex;
        }

        currentIndex++;
    }
    indicesToReturn[1] = lastIndex;
    break;
}
case Locations.Right:
{
    foreach (Line l in lines[directionToLookIn])
    {
        if (!firstIndexAllocated && (l.EndPoint.Y >
panelToBeSized.Top))
        {
            indicesToReturn[0] = currentIndex;
            firstIndexAllocated = true;
        }
        if (l.StartPoint.Y < panelToBeSized.Bottom)
        {
            lastIndex = currentIndex;
        }

        currentIndex++;
    }
    indicesToReturn[1] = lastIndex;
    break;
}
}

return indicesToReturn;
}

/// <summary>
/// Inserts a panel into the layout lines system, and updates the lines around
it.
/// </summary>
/// <param name="p">The control to add to the layout lines system</param>
/// <param name="DockType">The dock type of the control that is added to the
system</param>
public void AddPanel(Control p, DockTypes DockType)
{
    int leftIndex = 0;
    int rightIndex = 0;
    int topIndex = 0;
}

```

```

        int bottomIndex = 0;
        int indexToInsertPanelLineAt;
        Line panelLineToInsert;
        int[] lineIndicesRestrictingPanel;
        List<int> lineIndicesToRemove = new List<int>();
        Locations horizontalLocationToAddLine, verticalLocationToAddLine;

        horizontalLocationToAddLine = (MyPanel.IsDockedAtLeft(DockType) ?
    Locations.Left : Locations.Right);
        verticalLocationToAddLine = (MyPanel.IsDockedAtTop(DockType) ?
    Locations.Top : Locations.Bottom);

        //VERTICAL POSITION
        lineIndicesRestrictingPanel = GetIndicesOfLinesRestrictingPanel(p,
    verticalLocationToAddLine);

        leftIndex = lineIndicesRestrictingPanel[0];
        rightIndex = lineIndicesRestrictingPanel[1];

        if (leftIndex == rightIndex)
        {
            lines[verticalLocationToAddLine].Insert(leftIndex + 1, new
    Line(lines[verticalLocationToAddLine][leftIndex]));
            rightIndex = leftIndex + 1;
        }

        lines[verticalLocationToAddLine][leftIndex].setEndPointX(p.Left);
        lines[verticalLocationToAddLine][rightIndex].setStartPointX(p.Right);

        if (lines[verticalLocationToAddLine][rightIndex].Length == 0)
        {
            lineIndicesToRemove.Add(rightIndex);
        }
        for (int i = rightIndex - 1; i > leftIndex; i--)
        {
            lineIndicesToRemove.Add(i);
        }
        if (lines[verticalLocationToAddLine][leftIndex].Length == 0)
        {
            lineIndicesToRemove.Add(leftIndex);
            indexToInsertPanelLineAt = leftIndex;
        }
        else
        {
            indexToInsertPanelLineAt = leftIndex + 1;
        }

        foreach (int index in lineIndicesToRemove)
        {
            lines[verticalLocationToAddLine].RemoveAt(index);
        }
        lineIndicesToRemove.Clear();

        panelLineToInsert = AddLineFromSideOfPanel(p, (verticalLocationToAddLine ==
    Locations.Top ? Locations.Bottom : Locations.Top));
        lines[verticalLocationToAddLine].Insert(indexToInsertPanelLineAt,
    panelLineToInsert);

        //HORIZONTAL POSITION
        lineIndicesRestrictingPanel = GetIndicesOfLinesRestrictingPanel(p,
    horizontalLocationToAddLine);

        topIndex = lineIndicesRestrictingPanel[0];
        bottomIndex = lineIndicesRestrictingPanel[1];
    
```

```

        //If only one line index, the same layout line covers the whole of the top
        //of the panel
        if (topIndex == bottomIndex)
        {
            //Add a copy of this line
            lines[horizontalLocationToAddLine].Insert(topIndex + 1, new
Line(lines[horizontalLocationToAddLine][topIndex]));
            bottomIndex = topIndex + 1;
        }

        //Set the end of the left line and start of the right line to the corners
        //of the panel.
        lines[horizontalLocationToAddLine][topIndex].setEndPointY(p.Top);
        lines[horizontalLocationToAddLine][bottomIndex].setStartPointY(p.Bottom);

        //Remove any zero length lines at the bottom of the panel
        if (lines[horizontalLocationToAddLine][bottomIndex].Length == 0)
        {
            lineIndicesToRemove.Add(bottomIndex);
        }
        //Set a flag to remove any lines contained within the bounds of the panel
        for (int lineIndex = bottomIndex - 1; lineIndex > topIndex; lineIndex--)
        {
            lineIndicesToRemove.Add(lineIndex);
        }
        //Remove zero length lines at the top of the panel
        if (lines[horizontalLocationToAddLine][topIndex].Length == 0)
        {
            lineIndicesToRemove.Add(topIndex);
            indexToInsertPanelLineAt = topIndex;
            //If lines to the left are removed, the insertion index is the top index
        }
        else
        {
            indexToInsertPanelLineAt = topIndex + 1;
            //Else the insertion index is one above the top index
        }

        //Remove the lines marked for removal
        foreach (int index in lineIndicesToRemove)
        {
            lines[horizontalLocationToAddLine].RemoveAt(index);
        }
        lineIndicesToRemove.Clear();

        //Insert the line based on the edge of the panel
        panelLineToInsert = AddLineFromSideOfPanel(p, (horizontalLocationToAddLine
== Locations.Left ? Locations.Right : Locations.Left));
        lines[horizontalLocationToAddLine].Insert(indexToInsertPanelLineAt,
panelLineToInsert);
    }

    public void ClearLinesInContainer(Rectangle container)
    {
        SetupLinesAroundContainer(container);
    }

    /// <summary>Gets the line index of a line that matches the specified
    line</summary>
    /// <returns>The index of the line that matches the specified line
    /// Returns -1 if the line does not exist</returns>
    public int GetLineIndex(Line lineToMatch, Locations lineLocationToFind)
    {
        int lineIndex, currentLineIndex;

        currentLineIndex = 0;
        lineIndex = -1;
    }
}

```

```

        foreach (Line l in lines[lineLocationToFind])
        {
            if (l.StartPoint == lineToMatch.StartPoint && l.EndPoint ==
lineToMatch.EndPoint)
            {
                lineIndex = currentLineIndex;
            }
            currentLineIndex++;
        }

        return lineIndex;
    }
/// <summary>
/// Gets the maximum amount that the panel's size can be increased in the
selected location
/// </summary>
/// <returns>The number of pixels that the panel can be increased in size
by</returns>
public int GetSizeChangeAllowed(Control p, Locations location)
{
    int sizeChange;

    List<Line> LinesAtEndOfEnlarge = FindLinesSurroundingPanel(p, location);
    sizeChange = GetGreatestDistanceFromEdge(LinesAtEndOfEnlarge, location);

    return sizeChange;
}

/// <returns>A collection of the lines within the range of the panel, in the
selected direction</returns>
List<Line> FindLinesSurroundingPanel(Control p, Locations direction)
{
    int leftLimit = p.Left;
    int rightLimit = p.Right;
    int topLimit = p.Top;
    int bottomLimit = p.Bottom;

    List<Line> linesFound = new List<Line>();

    foreach (Line l in lines[direction])
    {
        if (direction == Locations.Bottom || direction == Locations.Top)
        {
            if (IsInRangeHorizontal(leftLimit, rightLimit, l))
            {
                linesFound.Add(l);
            }
        }
        else
        {
            if (IsInRangeVertical(topLimit, bottomLimit, l))
            {
                linesFound.Add(l);
            }
        }
    }

    return linesFound;
}
public int FindLineIndexToTruncate(Control p, Locations location)
{
    int lineIndexFound = 0;

    switch (location)
    {
        case Locations.Bottom:
    
```

```

        lineIndexFound = lines[Locations.Bottom].FindIndex(l =>
l.EndPoint.X >= p.Right);
        break;
    }
    case Locations.Left:
    {
        lineIndexFound = lines[Locations.Left].FindIndex(l => l.EndPoint.Y
>= p.Bottom);
        break;
    }
    case Locations.Top:
    {
        lineIndexFound = lines[Locations.Top].FindIndex(l => l.EndPoint.X
>= p.Right);
        break;
    }
    case Locations.Right:
    {
        lineIndexFound = lines[Locations.Right].FindIndex(l =>
l.EndPoint.Y >= p.Bottom);
        break;
    }
}
return lineIndexFound;
}

/// <returns>The distance away from the edge of the panel of the line that is
furthest away from the edge</returns>
int GetGreatestDistanceFromEdge(List<Line> Lines, Locations location)
{
    int furthestDistance = 0;
    int linesChecked = 0;

    if (location == Locations.Left || location == Locations.Top)
    {
        foreach (Line l in Lines)
        {
            if (linesChecked++ == 0) { furthestDistance = l.StaticLocation; }
            if (l.StaticLocation > furthestDistance) { furthestDistance =
l.StaticLocation; }
        }
    }
    else
    {
        foreach (Line l in Lines)
        {
            if (linesChecked++ == 0) { furthestDistance = l.StaticLocation; }
            if (l.StaticLocation < furthestDistance) { furthestDistance =
l.StaticLocation; }
        }
    }

    return furthestDistance;
}

/// <returns>A line matching the location of the side of the panel</returns>
public Line AddLineFromSideOfPanel(Control p, Locations sideToAddOn)
{
    Line lineToAdd = new Line();

    switch (sideToAddOn)
    {
        case Locations.Bottom:
        {
            lineToAdd = new Line(p.Left, p.Right, p.Bottom,
LineDirection.Horizontal);
            break;
        }
    }
}

```

```

        }
        case Locations.Left:
        {
            lineToAdd = new Line(p.Top, p.Bottom, p.Left,
LineDirection.Vertical);
            break;
        }
        case Locations.Top:
        {
            lineToAdd = new Line(p.Left, p.Right, p.Top,
LineDirection.Horizontal);
            break;
        }
        case Locations.Right:
        {
            lineToAdd = new Line(p.Top, p.Bottom, p.Right,
LineDirection.Vertical);
            break;
        }
    }
    return lineToAdd;
}

bool IsWithinRangeHorizontal(int leftLimit, int rightLimit, Line l)
{
    bool shouldBeAdded = false;

    if (l.StartPoint.X <= leftLimit && l.EndPoint.X >= rightLimit)
    {
        shouldBeAdded = true;
    }
    else
    {
        if (l.StartPoint.X >= leftLimit && l.StartPoint.X < rightLimit)
        {
            shouldBeAdded = true;
        }
        else
        {
            if (l.EndPoint.X > leftLimit && l.EndPoint.X <= rightLimit)
            {
                shouldBeAdded = true;
            }
        }
    }
    return shouldBeAdded;
}
bool IsWithinRangeVertical(int topLimit, int bottomLimit, Line l)
{
    bool shouldBeAdded = false;

    if (l.StartPoint.Y <= topLimit && l.EndPoint.Y >= bottomLimit)
    {
        shouldBeAdded = true;
    }
    else
    {
        if (l.StartPoint.Y >= topLimit && l.StartPoint.Y < bottomLimit)
        {
            shouldBeAdded = true;
        }
        else
        {
            if (l.EndPoint.Y > topLimit && l.EndPoint.Y <= bottomLimit)
            {
                shouldBeAdded = true;
            }
        }
    }
}

```

```

        }
    }

    return shouldBeAdded;
}

public List<Line> Top
{
    get { return lines[Locations.Top]; }
    set { lines[Locations.Top] = value; }
}
public List<Line> Left
{
    get { return lines[Locations.Left]; }
    set { lines[Locations.Left] = value; }
}
public List<Line> Bottom
{
    get { return lines[Locations.Bottom]; }
    set { lines[Locations.Bottom] = value; }
}
public List<Line> Right
{
    get { return lines[Locations.Right]; }
    set { lines[Locations.Right] = value; }
}
public Dictionary<Locations, List<Line>> LinesForLayout
{
    get { return lines; }
    set { lines = value; }
}
}

}

using System;
using System.Drawing;
using System.Windows.Forms;

namespace StratSim.View.MyFlowLayout
{
    /// <summary>
    /// Represents the form on which the majority of the system is displayed
    /// Allows window flow panel to be displayed and panels to be added and removed
    /// Contains a custom header and title.
    /// </summary>
    public class MainForm : Form
    {
        MainFormIOController IO;
        FormWindowState LastWindowState;
        public Panel header;
        Button minimise, maximise, close;

        int _formIndex;

        public delegate void MyKeyPressEventHandler(Keys key);
        public event MyKeyPressEventHandler MyKeyPress;

        public MainForm(int FormIndex)
        {
            InitializeComponent();
            SetupHeader();
            IO = new MainFormIOController(this);
        }
    }
}

```

```

        if (FormIndex == 0)
        {
            IO.SetupControls();
        }

        _formIndex = FormIndex;

        Resize += CustomOnResize;
        ResizeEnd += CustomEndResize;
        FormClosed += MainForm_FormClosed;
        IO.MainPanel.Click += MainPanel_Click;
    }

    protected override bool ProcessCmdKey(ref Message msg, Keys keyData)
    {
        MyKeyPress(keyData);
        return base.ProcessCmdKey(ref msg, keyData);
    }

    void MainPanel_Click(object sender, EventArgs e)
    {
        if (PanelDroppedOnForm != null)
            PanelDroppedOnForm(this);
    }

    public void MainForm_FormClosed(object sender, FormClosedEventArgs e)
    {
        Program.FormClosed(_formIndex);
    }

    void CustomOnResize(object sender, EventArgs e)
    {
        if (WindowState != LastWindowState)
        {
            LastWindowState = WindowState;
            FlowLayoutEvents.OnMainPanelLayoutChanged();
        }

        LocateComponents();
    }

    void CustomEndResize(object sender, EventArgs e)
    {
        FlowLayoutEvents.OnMainPanelLayoutChanged();
    }

    void LocateComponents()
    {
        header.Size = new Size(this.ClientRectangle.Right, 17);

        int rightBorder = header.ClientRectangle.Right - 3;
        minimise.Location = new Point(rightBorder - 66, 0);
        maximise.Location = new Point(rightBorder - 49, 0);
        close.Location = new Point(rightBorder - 32, 0);
    }

    public delegate void PanelDroppedEventHandler(MainForm DroppedForm);
    public event PanelDroppedEventHandler PanelDroppedOnForm;

    public void OnPanelAdded(int panelIndex)
    {
        PanelAdded(panelIndex);
    }

    public delegate void PanelAddedEventHandler(int panelIndex);
    public event PanelAddedEventHandler PanelAdded;

```

```

/// <summary>
/// Sets up the header of the form with minimise, maximise, and close buttons.
/// </summary>
void SetupHeader()
{
    header = new Panel();
    header.BackgroundImage = StratSim.Properties.Resources.Form_Header;
    header.Location = new Point(0, 0);

    Label HeaderTitle = new Label();
    HeaderTitle.Text = "Strat Sim";
    HeaderTitle.Location = new Point(0, 0);
    HeaderTitle.Size = new Size(100, 17);
    HeaderTitle.BackColor = Color.FromArgb(52, 43, 236);
    HeaderTitle.ForeColor = Color.White;
    header.Controls.Add(HeaderTitle);

    minimise = new Button();
    minimise.Size = new Size(17, 17);
    minimise.FlatStyle = FlatStyle.Flat;
    minimise.BackgroundImage = StratSim.Properties.Resources.Hide_Display;
    minimise.MouseLeave += (s, e) => minimise.BackgroundImage =
        StratSim.Properties.Resources.Hide_Display;
    minimise.MouseHover += (s, e) => minimise.BackgroundImage =
        StratSim.Properties.Resources.Hide_Hover;
    minimise.Click += (s, e) => this.WindowState = FormWindowState.Minimized;
    header.Controls.Add(minimise);

    maximise = new Button();
    maximise.Size = new Size(17, 17);
    maximise.BackgroundImage =
        StratSim.Properties.Resources.FullScreen_Display;
    maximise.MouseLeave += (s, e) => maximise.BackgroundImage =
        StratSim.Properties.Resources.FullScreen_Display;
    maximise.MouseHover += (s, e) => maximise.BackgroundImage =
        StratSim.Properties.Resources.FullScreen_Hover;
    maximise.FlatStyle = FlatStyle.Flat;
    maximise.Click += SetButtonImages;
    header.Controls.Add(maximise);

    close = new Button();
    close.Size = new Size(32, 17);
    close.FlatStyle = FlatStyle.Flat;
    close.BackgroundImage = StratSim.Properties.Resources.Close_Display;
    close.MouseHover += (s, e) => close.BackgroundImage =
        StratSim.Properties.Resources.Close_Hover;
    close.MouseLeave += (s, e) => close.BackgroundImage =
        StratSim.Properties.Resources.Close_Display;
    close.Click += (s, e) => this.Close();
    header.Controls.Add(close);

    LocateComponents();

    this.Controls.Add(header);
}

/// <summary>
/// <para>Alters the form window state after the 'restore' button is
/// clicked.</para>
/// <para>Changes the button image to reflect the current window state.</para>
/// </summary>
void SetButtonImages(object sender, EventArgs e)
{
    if (this.WindowState == FormWindowState.Normal)
    {
        this.WindowState = FormWindowState.Maximized;
    }
}

```

```

        maximise.BackgroundImage =
StratSim.Properties.Resources.RestoreDown_Display;
        maximise.MouseHover += (a, b) => maximise.BackgroundImage =
StratSim.Properties.Resources.RestoreDown_Hover;
        maximise.MouseLeave += (a, b) => maximise.BackgroundImage =
StratSim.Properties.Resources.RestoreDown_Display;
        maximise.MouseHover -= (a, b) => maximise.BackgroundImage =
StratSim.Properties.Resources.FullScreen_Hover;
        maximise.MouseLeave -= (a, b) => maximise.BackgroundImage =
StratSim.Properties.Resources.FullScreen_Display;

    }
else
{
    if (this.WindowState == FormWindowState.Maximized)
    {
        this.WindowState = FormWindowState.Normal;
        maximise.BackgroundImage =
StratSim.Properties.Resources.FullScreen_Display;
        maximise.MouseHover += (a, b) => maximise.BackgroundImage =
StratSim.Properties.Resources.FullScreen_Hover;
        maximise.MouseLeave += (a, b) => maximise.BackgroundImage =
StratSim.Properties.Resources.FullScreen_Display;
        maximise.MouseHover -= (a, b) => maximise.BackgroundImage =
StratSim.Properties.Resources.RestoreDown_Hover;
        maximise.MouseLeave -= (a, b) => maximise.BackgroundImage =
StratSim.Properties.Resources.RestoreDown_Display;
    }
}
}

void InitializeComponent()
{
    System.ComponentModel.ComponentResourceManager resources = new
System.ComponentModel.ComponentResourceManager(typeof(MainForm));
    this.SuspendLayout();

    this.Name = " MainForm";
    this.ControlBox = false;
    this.Text = String.Empty;
    this.Icon = (Icon)resources.GetObject("$this.Icon");
    this.Height = Screen.GetWorkingArea(this).Height;
    this.Width = Screen.GetWorkingArea(this).Width;
    this.DoubleBuffered = true;
    this.DesktopLocation = new Point(0, 0);
    this.BackColor = Color.White;

    this.ResumeLayout(true);

}

/// <summary>
/// The index of the form in the list of forms currently active in the system
/// </summary>
public int FormIndex
{
    get { return _formIndex; }
}

public MainFormIOController IOController
{
    get{ return IO;}
}
}
}

```

```

using StratSim.Model;
using StratSim.View.Panels;
using StratSim.ViewModel;
using System;
using System.Windows.Forms;

namespace StratSim.View.MyFlowLayout
{
    /// <summary>
    /// Controller providing methods to hide and show panels on the main form
    /// Contains methods for adding and removing controls,
    /// and for starting the functions used on the form.
    /// </summary>
    public class MainFormIOController : IDisposable
    {
        MainForm associatedForm;
        MyToolbar toolbar;
        WindowFlowPanel mainPanel;
        ContentTabControl ContentTabControl;
        DragDropController dragDropController;

        public MainFormIOController(MainForm Form)
        {
            associatedForm = Form;
            mainPanel = new WindowFlowPanel(associatedForm);
            toolbar = new MyToolbar(mainPanel);
            dragDropController = new DragDropController(associatedForm);

            associatedForm.KeyPreview = true;
            associatedForm.KeyPress += mainPanel.AlexMcCormickEasterEgg;
            associatedForm.FormClosed += form_FormClosed;
            associatedForm.Controls.Add(mainPanel);
            AddPanel(toolbar);

            SubscribeToEvents();
        }

        void form_FormClosed(object sender, FormClosedEventArgs e)
        {
            UnsubscribeFromEvents();
        }

        /// <summary>
        /// Sets up the initial set of controls on the form when it is initialised.
        /// </summary>
        public void SetupControls()
        {
            InitialiseControls();
            AddControls();
            FinishedAdding();
        }
        void InitialiseControls()
        {
            Program.InfoPanel = new InfoPanel(associatedForm);
            Program.SettingsPanel = new SettingsPanel(associatedForm);
            Program.ContentTabControl = new ContentTabControl(associatedForm);
            Program.NewStartPanel = new NewStartPanel(associatedForm);
        }
        void AddControls()
        {
            AddContentTabControl(Program.ContentTabControl);
            AddContentPanel(Program.NewStartPanel);
        }

        void UnsubscribeFromEvents()
        {
    
```

```

    PanelControlEvents.LoadPaceParametersFromFile -=
PanelControlEvents_LoadPaceParametersFromFile;
    PanelControlEvents.LoadStrategiesFromFile -=
PanelControlEvents_LoadStrategiesFromFile;
    PanelControlEvents.LoadPaceParametersFromRace -=
PanelControlEvents_LoadPaceParametersFromRace;
    PanelControlEvents.LoadStrategiesFromData -=
PanelControlEvents_LoadStrategiesFromData;
    PanelControlEvents.StartRaceFromStrategies -=
PanelControlEvents_StartRaceFromStrategies;
    PanelControlEvents.ShowInfoPanel -= PanelControlEvents_ShowInfoPanel;
    PanelControlEvents.ShowSettingsPanel -=
PanelControlEvents_ShowSettingsPanel;
    PanelControlEvents.ShowContentTabControl -=
PanelControlEvents_ShowContentTabControl;
    PanelControlEvents.ShowDriverSelectPanel -=
PanelControlEvents_ShowDriverSelectPanel;
    PanelControlEvents.ShowGraph -= PanelControlEvents_ShowGraph;
    PanelControlEvents.ShowAxes -= PanelControlEvents_ShowAxes;
    PanelControlEvents.ShowPaceParameters -=
PanelControlEvents_ShowPaceParameters;
    PanelControlEvents.ShowStrategies -= PanelControlEvents_ShowStrategies;
    PanelControlEvents.ShowArchives -= PanelControlEvents_ShowArchives;
    PanelControlEvents.ShowDataInput -= PanelControlEvents_ShowDataInput;
    PanelControlEvents.RemoveGraphPanels -=
PanelControlEvents_RemoveGraphPanels;
}
void SubscribeToEvents()
{
    PanelControlEvents.LoadPaceParametersFromFile +=
PanelControlEvents_LoadPaceParametersFromFile;
    PanelControlEvents.StrategiesLoaded +=
PanelControlEvents_LoadStrategiesFromFile;
    PanelControlEvents.LoadPaceParametersFromRace +=
PanelControlEvents_LoadPaceParametersFromRace;
    PanelControlEvents.LoadStrategiesFromData +=
PanelControlEvents_LoadStrategiesFromData;
    PanelControlEvents.StartRaceFromStrategies +=
PanelControlEvents_StartRaceFromStrategies;
    PanelControlEvents.ShowInfoPanel += PanelControlEvents_ShowInfoPanel;
    PanelControlEvents.ShowSettingsPanel +=
PanelControlEvents_ShowSettingsPanel;
    PanelControlEvents.ShowContentTabControl +=
PanelControlEvents_ShowContentTabControl;
    PanelControlEvents.ShowDriverSelectPanel +=
PanelControlEvents_ShowDriverSelectPanel;
    PanelControlEvents.ShowGraph += PanelControlEvents_ShowGraph;
    PanelControlEvents.ShowAxes += PanelControlEvents_ShowAxes;
    PanelControlEvents.ShowPaceParameters +=
PanelControlEvents_ShowPaceParameters;
    PanelControlEvents.ShowStrategies += PanelControlEvents_ShowStrategies;
    PanelControlEvents.ShowArchives += PanelControlEvents_ShowArchives;
    PanelControlEvents.ShowDataInput += PanelControlEvents_ShowDataInput;
    PanelControlEvents.RemoveGraphPanels +=
PanelControlEvents_RemoveGraphPanels;
}

void PanelControlEvents_RemoveGraphPanels(MainForm callingForm)
{
    if (callingForm == associatedForm)
    {
        RemoveGraphPanels();
    }
}
void PanelControlEvents_ShowDataInput(MainForm callingForm)
{
    if (callingForm == associatedForm)

```

```

    {
        StartDataInput();
    }
}
void PanelControlEvents_ShowArchives(MainForm callingForm)
{
    if (callingForm == associatedForm)
    {
        StartTimingArchives();
    }
}
void PanelControlEvents_ShowStrategies(MainForm callingForm)
{
    //Shows all controls required for the strategies to be displayed
    if (callingForm == associatedForm)
    {
        RemoveGraphPanels();
        ShowStrategyViewer();
        ShowDriverList(false);
        ShowGraph();
        FinishedAdding();
    }
}
void PanelControlEvents_ShowPaceParameters(MainForm callingForm)
{
    if (callingForm == associatedForm)
    {
        ShowTimingDataPanel();
        FinishedAdding();
    }
}
void PanelControlEvents_ShowAxes(MainForm callingForm)
{
    if (callingForm == associatedForm)
    {
        ShowAxes();
        FinishedAdding();
    }
}
void PanelControlEvents_ShowGraph(MainForm callingForm)
{
    if (callingForm == associatedForm)
    {
        ShowGraph();
        FinishedAdding();
    }
}
void PanelControlEvents_ShowDriverSelectPanel(MainForm callingForm)
{
    if (callingForm == associatedForm)
    {
        ShowDriverList();
        FinishedAdding();
    }
}
void PanelControlEvents_ShowContentTabControl(MainForm callingForm)
{
    if (callingForm == associatedForm)
    {
        ShowContentPanel();
        FinishedAdding();
    }
}
void PanelControlEvents_ShowSettingsPanel(MainForm callingForm)
{
    if (callingForm == associatedForm)
    {

```

```

        ShowSettingsPanel();
        FinishedAdding();
    }
}
void PanelControlEvents_ShowInfoPanel(MainForm callingForm)
{
    if (callingForm == associatedForm)
    {
        ShowInfoPanel();
        FinishedAdding();
    }
}
void PanelControlEvents_StartRaceFromStrategies()
{
    StartRaceSimulation();
    ShowDriverList(true);
    ShowGraph();
    FinishedAdding();
}
void PanelControlEvents_LoadStrategiesFromData()
{
    StartStrategyProcessing(false);
}
void PanelControlEvents_LoadPaceParametersFromRace()
{
    StartTimingDataPanel(false);
}
void PanelControlEvents_LoadStrategiesFromFile()
{
    StartStrategyProcessing(true);
}
void PanelControlEvents_LoadPaceParametersFromFile()
{
    StartTimingDataPanel(true);
}

/// <summary>
/// Adds a control to the form.
/// </summary>
/// <param name="ControlToAdd">The IDockableControl to add to the form</param>
public void AddPanel(IDockableControl ControlToAdd)
{
    if (ControlToAdd != null && ControlToAdd.Removed)
    {
        mainPanel.AddControl(ControlToAdd, true);
        if (ControlToAdd.Type == typeof(MyPanel))
            ((MyPanel)ControlToAdd).OnPanelOpened();
        if (ControlToAdd.Type == typeof(MyToolbar))
            ControlToAdd.Removed = false;
    }
}
/// <summary>
/// Removes a control from the form.
/// </summary>
/// <param name="ControlToRemove">The IDockableControl to remove from the
form</param>
public void RemovePanel(IDockableControl ControlToRemove)
{
    if (ControlToRemove.Type == typeof(MyPanel))
    {
        int panelIndexToRemove = ((MyPanel)ControlToRemove).PanelIndex;

        if (!ControlToRemove.Removed)
        {
            toolbar.RemovePanel(panelIndexToRemove);
            mainPanel.RemoveControl(ControlToRemove);
            if (mainPanel.PanelsOnForm == 0)

```

```

        {
            if (AssociatedForm.FormIndex != 0)
                CloseForm();
        }
        else
        {
            foreach (MyPanel p in mainPanel.VisiblePanels)
            {
                if (p.PanelIndex > panelIndexToRemove)
                {
                    p.PanelIndex--;
                }
            }
        }
        ((MyPanel)ControlToRemove).OnPanelClosed();
        FlowLayoutEvents.OnMainPanelLayoutChanged();
    }
}
/// <summary>
/// Adds the content tab control to the form.
/// </summary>
/// <param name="MainTabControl">The instance of a ContentTabControl to add to
the form</param>
public void AddContentTabControl(ContentTabControl MainTabControl)
{
    if (MainTabControl.Removed)
    {
        MainTabControl.Removed = false;
        SetContentTabControl(MainTabControl);
        mainPanel.AddControl(Program.ContentTabControl, true);
    }
}
/// <summary>
/// Sets the forms tab control to an instance of a ContentTabControl
/// </summary>
public void SetContentTabControl(ContentTabControl TabControl)
{
    ContentTabControl = TabControl;
}
/// <summary>
/// Adds a panel to the content tab control
/// </summary>
/// <param name="ControlToAdd">The panel to add within the main tab
control</param>
void AddContentPanel(MyPanel ControlToAdd)
{
    ContentTabControl.AddPanelToTabControl(ControlToAdd);
    if (ControlToAdd.Removed)
        ControlToAdd.OnPanelOpened();
}
/// <summary>
/// Removes a panel from the content tab control
/// </summary>
/// <param name="ControlToRemove">The panel to remove from the main tab
control</param>
void RemoveContentPanel(MyPanel ControlToRemove)
{
    if (!ControlToRemove.Removed)
        ControlToRemove.OnPanelClosed();
    ContentTabControl.RemovePanelFromControl(ControlToRemove);
}

public void FinishedAdding()
{
    mainPanel.FinishedAddingPanels();
}

```

```

void StartDataInput()
{
    Program.DataInput = new DataInput(associatedForm);
    Program.InfoPanel.WriteData("Starting Data Input Window");
    AddContentPanel(Program.DataInput);
}
void StartTimingDataPanel(bool loadFromFile)
{
    Program.PopulateDriverDataFromFiles(Data.RaceIndex);
    Functions.CalculatePaceParameters(Data.RaceIndex);
    if (Program.TimeAnalysis != null)
    {
        RemoveContentPanel(Program.TimeAnalysis);

        //If the panel has been started and not initialised (i.e. data not
loaded):
        if (Program.TimeAnalysis.Removed && Program.TimeAnalysis.Initialised)
            Program.TimeAnalysis.OnPanelClosed();
    }

    Program.TimeAnalysis = new PaceParameters(associatedForm, loadFromFile);
    MyEvents.OnFinishedLoadingParameters();
}
void StartTimingArchives()
{
    Program.TimingArchives = new TimingArchives(associatedForm);
    Program.InfoPanel.WriteData("Starting timing archives");
    AddContentPanel(Program.TimingArchives);
}
void StartStrategyProcessing(bool loadFromFile)
{
    Functions.OptimiseAllStrategies(Data.RaceIndex);
    StartStrategyPanel(loadFromFile);
}
void StartStrategyPanel(bool loadFromFile)
{
    if (Program.DriverSelectPanel == null)
        StartDriverList();

    if (Program.StrategyViewer != null)
    {
        RemoveContentPanel(Program.StrategyViewer);
        RemoveGraphPanels();

        //If the panel has been started and not initialised (i.e. data not
loaded):
        if (Program.StrategyViewer.Removed &&
Program.StrategyViewer.Initialised)
            Program.StrategyViewer.OnPanelClosed();
    }

    Program.StrategyViewer = new StrategyViewer(associatedForm, loadFromFile);
    Program.StrategyViewer.SetDriverPanel(Program.DriverSelectPanel);
}
public void StartGraph(CumulativeTimeGraph graph)
{
    Program.AxesWindow = new AxesWindow(associatedForm);
    Program.Graph = graph;
    LinkGraphToDrivers();
}
void StartDriverList()
{
    Program.DriverSelectPanel = new DriverSelectPanel(associatedForm);
    LinkGraphToDrivers();
}
void ShowInfoPanel()

```

```

{
    AddPanel(Program.InfoPanel);
}
void ShowSettingsPanel()
{
    AddPanel(Program.SettingsPanel);
}
void ShowContentPanel()
{
    if (ContentTabControl != null)
    {
        AddContentTabControl(ContentTabControl);
    }
}
void ShowTimingDataPanel()
{
    if (Program.TimeAnalysis != null)
    {
        RemoveContentPanel(Program.TimeAnalysis);
        AddContentPanel(Program.TimeAnalysis);
        Program.TimeAnalysis.AddToolStrip();
        Program.InfoPanel.WriteData("Starting time analysis panel");
    }
}
void ShowStrategyViewer()
{
    if (Program.StrategyViewer != null)
    {
        RemoveContentPanel(Program.StrategyViewer);
        AddContentPanel(Program.StrategyViewer);
        Program.StrategyViewer.AddToolStrip();
        RemoveGraphPanels();
        StartGraph(Program.StrategyViewer.GetGraph());
        ShowGraph();
    }
}
void ShowDriverList(bool showTimeGaps)
{
    if (Program.DriverSelectPanel != null)
    {
        RemovePanel(Program.DriverSelectPanel);
        AddPanel(Program.DriverSelectPanel);
        Program.DriverSelectPanel.ShowTimeGaps = showTimeGaps;
    }
}
void ShowDriverList()
{
    if (Program.DriverSelectPanel != null)
    {
        RemovePanel(Program.DriverSelectPanel);
        AddPanel(Program.DriverSelectPanel);
    }
}
void ShowGraph()
{
    if (Program.Graph != null)
    {
        AddPanel(Program.Graph);
    }
}
void ShowAxes()
{
    if (Program.AxesWindow != null)
    {
        AddPanel(Program.AxesWindow);
    }
}

```

```

void StartRaceSimulation()
{
    //must remove in reverse order of panel index
    if (Program.StrategyViewer.PanelIndex > Program.TimeAnalysis.PanelIndex)
    {
        if (Program.StrategyViewer != null)
            RemoveContentPanel(Program.StrategyViewer);
        if (Program.TimeAnalysis != null)
            RemoveContentPanel(Program.TimeAnalysis);
    }
    else
    {
        if (Program.TimeAnalysis != null)
            RemoveContentPanel(Program.TimeAnalysis);
        if (Program.StrategyViewer != null)
            RemoveContentPanel(Program.StrategyViewer);
    }

    RemoveGraphPanels();

    //Setup the strategies for the race
    Strategy[] Strategies = new Strategy[Data.NumberOfDrivers];
    int driverIndex = 0;
    foreach (Driver d in Data.Drivers)
    {
        Strategies[driverIndex++] = d.Strategy;
    }

    StartDriverList();
    Program.DriverSelectPanel.ShowTimeGaps = true;

    //Start the race
    if (Data.Race == null)
    {
        Data.Race = new Race(Data.RaceIndex, Strategies, associatedForm);
        Data.Race.SetupGrid();
        Data.Race.SimulateRace();
    }
    else
    {
        CumulativeTimeGraph newGraph;
        Data.Race.RestartSimulation(out newGraph, Strategies);
        StartGraph(newGraph);
    }
}
void RemoveGraphPanels()
{
    if (Program.Graph != null)
        RemovePanel(Program.Graph);
    if (Program.DriverSelectPanel != null)
        RemovePanel(Program.DriverSelectPanel);
    if (Program.AxesWindow != null)
        RemovePanel(Program.AxesWindow);
}
void LinkGraphToDrivers()
{
    if (Program.Graph != null && Program.DriverSelectPanel != null)
    {
        Program.DriverSelectPanel.SetGraph(Program.Graph);
        Program.Graph.SetDriverPanel(Program.DriverSelectPanel);
    }
    if (Program.StrategyViewer != null && Program.DriverSelectPanel != null)
    {
        Program.StrategyViewer.SetDriverPanel(Program.DriverSelectPanel);
    }
    if (Program.StrategyViewer != null && Program.Graph != null)
    {
}

```

```

        Program.StrategyViewer.SetGraph(Program.Graph);
    }

    public void StartReLayout(MyPanel SizingPanel, System.Drawing.Point
StartLocation)
    {
        associatedForm.Controls.Remove(MainPanel);
        dragDropController.StartDragDropLayout(SizingPanel,
MainPanel.VisiblePanels, StartLocation, MainPanel.DockPoints);
    }
    public void EndReLayout()
    {
        associatedForm.Controls.Add(MainPanel);
    }

    public void ShowDynamicLayoutPanel(Pane LayoutPanel)
    {
        LayoutPanel.Size = associatedForm.ClientSize;
        LayoutPanel.Location = new System.Drawing.Point(0, 0);
        associatedForm.Controls.Add(LayoutPanel);
        LayoutPanel.Show();
    }
    public void HideDynamicLayoutPanel(Pane LayoutPanel)
    {
        associatedForm.Controls.Remove(LayoutPanel);
        EndReLayout();
    }

    public void OpenPanelInCurrentForm(MyPanel PanelToAddToWindow, MainForm
FormToAddOn)
    {
        RemovePanel(PanelToAddToWindow);
        FormToAddOn.IOController.AddPanel(PanelToAddToWindow);
        PanelToAddToWindow.OnOpenedInDifferentForm(FormToAddOn);
        FlowLayoutEvents.OnMainPanelLayoutChanged();
    }

    public void CloseForm()
    {
        associatedForm.Close();
    }

    public void Dispose()
    {
        mainPanel.Dispose();
        toolbar.Dispose();
        dragDropController.Dispose();
    }

    public MainForm AssociatedForm
    {
        get { return associatedForm; }
        set { associatedForm = value; }
    }

    public MyToolbar Toolbar
    { get { return toolbar; } }
    public WindowFlowPanel MainPanel
    { get { return mainPanel; } }
}

using System;
using System.Windows.Forms;

```

```

namespace StratSim.View.MyFlowLayout
{
    /// <summary>
    /// Represents a button used in the context menu, used for changing the
    properties of the panel
    /// </summary>
    public class MyToolStripButton : ToolStripButton
    {
        int thisButtonIndex;
        Type type;

        //Constructors for buttons based on different properties passed to them
        public MyToolStripButton(int buttonIndex)
        {
            thisButtonIndex = buttonIndex;
            type = typeof(int);
            CheckOnClick = true;
            Click += GenerateClickEvent;
            this.Width = 100;
            this.DisplayStyle = ToolStripItemDisplayStyle.Text;
        }

        public MyToolStripButton(DockTypes dockType)
        {
            thisButtonIndex = (int)dockType;
            type = typeof(DockTypes);
            CheckOnClick = true;
            Click += GenerateClickEvent;
            this.Width = 100;
            this.DisplayStyle = ToolStripItemDisplayStyle.Text;
        }

        public MyToolStripButton(AutosizeTypes autosizeType)
        {
            thisButtonIndex = (int)autosizeType;
            type = typeof(AutosizeTypes);
            CheckOnClick = true;
            Click += GenerateClickEvent;
            this.Width = 100;
            this.DisplayStyle = ToolStripItemDisplayStyle.Text;
        }

        public MyToolStripButton(FillStyles fillStyle)
        {
            thisButtonIndex = (int)fillStyle;
            type = typeof(FillStyles);
            CheckOnClick = true;
            Click += GenerateClickEvent;
            this.Width = 100;
            this.DisplayStyle = ToolStripItemDisplayStyle.Text;
        }

        public virtual void GenerateClickEvent(object sender, EventArgs e)
        {
            ButtonClicked(this.ButtonIndex, this.Type);
        }

        public delegate void OnCheckedEventHandler(int ButtonIndex, Type ButtonType);
        public event OnCheckedEventHandler ButtonClicked;

        public int ButtonIndex
        {
            get { return thisButtonIndex; }
            set { thisButtonIndex = value; }
        }
        public Type Type
        { get { return type; } }
    }
}

```

```

}

/// <summary>
/// Represents a custom context menu, displayed when the header of a panel is
right-clicked
/// Provides options for changing the layout of the panel
/// </summary>
class MyContextMenu : ContextMenuStrip
{
    ToolStripDropDownButton DockTypeDropDown, AutoSizeDropDown, FillStyleDropDown;

    MyPanel _parent;

    public MyContextMenu(MyPanel parent)
    {
        _parent = parent;
        SetupComponents();
    }

    void SetupComponents()
    {
        MyToolStripButton TempButton;

        //Dock Type
        DockTypeDropDown = new ToolStripDropDownButton();
        DockTypeDropDown.Text = "Item Dock Location";
        DockTypeDropDown.ToolTipText = "Select where this item is docked on your
screen";

        foreach (var DockType in (DockTypes[])Enum.GetValues(typeof(DockTypes)))
        {
            TempButton = new MyToolStripButton(DockType);
            TempButton.Text = Convert.ToString(DockType);
            TempButton.CheckOnClick = true;
            if (_parent.DockType == DockType) { TempButton.Checked = true; }
            TempButton.ButtonClicked += ContextMenuClick;
            DockTypeDropDown.DropDownItems.Add(TempButton);
        }

        //AutoSize
        AutoSizeDropDown = new ToolStripDropDownButton();
        AutoSizeDropDown.Text = "Item Autosize Properties";
        AutoSizeDropDown.ToolTipText = "Select how this item can be stretched to
fill the screen";

        foreach (var AutoSize in
(AutosizeTypes[])Enum.GetValues(typeof(AutosizeTypes)))
        {
            TempButton = new MyToolStripButton(AutoSize);
            TempButton.Text = Convert.ToString(AutoSize);
            TempButton.CheckOnClick = true;
            if (_parent.AutoScaleType == AutoSize) { TempButton.Checked = true; }
            TempButton.ButtonClicked += ContextMenuClick;
            AutoSizeDropDown.DropDownItems.Add(TempButton);
        }

        AutoSizeDropDown.Width = 100;

        //Fill Style
        FillStyleDropDown = new ToolStripDropDownButton();
        FillStyleDropDown.Text = "Item Fill Properties";
        FillStyleDropDown.ToolTipText = "Select whether this item fills the screen
or is finite-size";

        foreach (var FillStyle in (FillStyles[])Enum.GetValues(typeof(FillStyles)))
        {
            TempButton = new MyToolStripButton(FillStyle);

```

```

        TempButton.Text = Convert.ToString(FillStyle);
        TempButton.CheckOnClick = true;
        if (_parent.FillStyle == FillStyle) { TempButton.Checked = true; }
        TempButton.ButtonClicked += ContextMenuClick;
        FillStyleDropDown.DropDownItems.Add(TempButton);
    }
    FillStyleDropDown.Width = 100;

    this.Items.Add(DockTypeDropDown);
    this.Items.Add(AutoSizeDropDown);
    this.Items.Add(FillStyleDropDown);
}

/// <summary>
/// Selects the correct buttons in the context menu based on panel properties
/// </summary>
public void SetCheckButtons()
{
    foreach (var DockType in (DockTypes[])Enum.GetValues(typeof(DockTypes)))
    {

        ((MyToolStripButton)DockTypeDropDown.DropDownItems[(int)DockType]).Checked =
(_parent.DockType == DockType);
    }
    foreach (var AutoSize in
(AutosizeTypes[])Enum.GetValues(typeof(AutosizeTypes)))
    {

        ((MyToolStripButton)AutoSizeDropDown.DropDownItems[(int)AutoSize]).Checked =
(_parent.AutoSizeType == AutoSize);
    }
    foreach (var FillStyle in (FillStyles[])Enum.GetValues(typeof(FillStyles)))
    {

        ((MyToolStripButton)FillStyleDropDown.DropDownItems[(int)FillStyle]).Checked =
(_parent.FillStyle == FillStyle);
    }
}

/// <summary>
/// Handles a click event on a context menu button
/// </summary>
void ContextMenuClick(int buttonIndex, Type buttonType)
{
    int buttonIndexIncrement = 0;
    if (buttonType == typeof(DockTypes))
    {
        foreach (MyToolStripButton ts in DockTypeDropDown.DropDownItems)
        {
            ts.Checked = (buttonIndexIncrement++ == buttonIndex);
        }
        (_parent).DockType = (DockTypes)buttonIndex;
    }
    if (buttonType == typeof(AutosizeTypes))
    {
        foreach (MyToolStripButton ts in AutoSizeDropDown.DropDownItems)
        {
            ts.Checked = (buttonIndexIncrement++ == buttonIndex);
        }
        (_parent).AutoSizeType = (AutosizeTypes)buttonIndex;
    }
    if (buttonType == typeof(FillStyles))
    {
        foreach (MyToolStripButton ts in FillStyleDropDown.DropDownItems)
        {
            ts.Checked = (buttonIndexIncrement++ == buttonIndex);
        }
    }
}

```

```

        (_parent).FillStyle = (FillStyles)buttonIndex;
    }
}
}

using StratSim.ViewModel;
using System;
using System.Drawing;
using System.Windows.Forms;

namespace StratSim.View.MyFlowLayout
{
    #region styles
    public enum FillStyles
    {
        None,
        FullWidth,
        FullHeight,
        FullScreen
    };

    public enum AutosizeTypes
    {
        Free,
        Constant,
        AutoWidth,
        AutoHeight
    };

    public enum DockTypes
    {
        TopLeft,
        Left,
        BottomLeft,
        Top,
        None,
        Bottom,
        TopRight,
        Right,
        BottomRight,
        Top2,
        Bottom2
    };
    #endregion

    /// <summary>
    /// The base class for a panel that is displayed on the window flow panel.
    /// Contains event handlers for events on the panel,
    /// and static methods for using the panel.
    /// Contains the base properties of the panel including those used for layout
    logic.
    /// </summary>
    public class MyPanel : Panel, IDockableControl
    {
        Panel header;
        Label titleLabel;
        PictureBox panelIconBox;
        Button hide, fullScreenToggle, close;
        MyContextMenu contextMenu;

        FillStyles windowFillStyle;
        AutosizeTypes autoSize;
        DockTypes windowDockType;
        Size originalSize;
        Point dockPointLocation;
    }
}

```

```

Type type;
Padding padding;

MainForm parentForm;
Image icon;

bool visible;
bool removed;
int panelIndex;

bool inContentPanel;

public const int controlSpacing = 4;

/// <summary>
/// <para>Starts a new instance of a panel to be displayed and resized
dynamically
/// <para>on a window flow panel.</para>
/// <para>Contains methods for setting up and manipulating the panels.</para>
/// <para>After the panel is constructed, 'set panel properties' must be
called.</para>
/// </summary>
/// <param name="width">The minimum width of the panel to be displayed</param>
/// <param name="height">The minimum height of the panel to be
displayed</param>
/// <param name="title">The text to display in the header of the panel</param>
/// <param name="ParentForm">The form on which the panel is to be
displayed</param>
/// <param name="Icon">A png file path representing the icon to be displayed
in the header of the panel</param>
public MyPanel(int width, int height, string title, MainForm ParentForm, Image
Icon)
{
    this.Width = width;
    this.Height = height;
    this.Name = title;

    icon = Icon;
    parentForm = ParentForm;

    visible = true;
    padding = new Padding(10, 25, 10, 10);
    DoubleBuffered = true;
    removed = false;
    BackColor = Color.White;

    FlowLayoutEvents.MainPanelResizeEnd += MyEvents_MainPanelResizeEnd;
    FlowLayoutEvents.MainPanelLayoutChanged += MyEvents_MainPanelLayoutChanged;
    PanelClosed += MyPanel_PanelClosed;
    PanelOpened += MyPanel_PanelOpened;

    SetupComponents(title);
    PositionComponents();
    this.type = typeof(MyPanel);
}

void MyPanel_PanelOpened()
{
    removed = false;
}

void MyPanel_PanelClosed()
{
    if (inContentPanel)
    {
        MyEvents.OnRemoveContentPanel(this.PanelIndex);
    }
}

```

```

        removed = true;
    }

    void MyEvents_MainPanelLayoutChanged()
    {
        contextMenu.SetCheckButtons();
    }

    void MyEvents_MainPanelResizeEnd()
    {
        PositionComponents();
    }

    /// <summary>
    /// Sets the panel's layout properties to the specified values.
    /// </summary>
    public void SetPanelProperties(DockTypes dockType, AutosizeTypes autosizeType,
FillStyles fillStyle, Size size)
    {
        windowDockType = dockType;
        autoSize = autosizeType;
        windowFillStyle = fillStyle;
        originalSize = size;

        SetupContextMenu();
    }

    void SetupComponents(string title)
    {
        this.BorderStyle = BorderStyle.FixedSingle;

        header = new Panel();
        header.Location = new Point(0, 0);
        header.Size = new Size(this.Width, 25);
        header.BackgroundImage = Properties.Resources.Panel_Header;
        header.ImageLayout = ImageLayout.Stretch;
        header.MouseDown += header_MouseDown;
        header.MouseUp += header_MouseUp;

        panelIconBox = new PictureBox();
        panelIconBox.Image = icon;
        panelIconBox.Location = new Point(0, 0);
        panelIconBox.Size = new Size(16, 16);

        titleLabel = new Label();
        titleLabel.AutoSize = true;
        titleLabel.Text = title;
        titleLabel.Location = new Point(20, 0);
        titleLabel.ForeColor = Color.White;
        titleLabel.BackColor = Color.FromArgb(52, 43, 236);

        hide = new Button();
        hide.Size = new Size(17, 17);
        hide.FlatStyle = FlatStyle.Flat;
        hide.BackgroundImage = Properties.Resources.Hide_Display;
        hide.MouseHover += (s, e) => hide.BackgroundImage =
Properties.Resources.Hide_Hover;
        hide.MouseLeave += (s, e) => hide.BackgroundImage =
Properties.Resources.Hide_Display;
        hide.Click += ((s, e) => MyVisible = false);

        fullScreenToggle = new Button();
        fullScreenToggle.Size = new Size(17, 17);
        fullScreenToggle.FlatStyle = FlatStyle.Flat;
        fullScreenToggle.BackgroundImage = Properties.Resources.FullScreen_Display;
        fullScreenToggle.Click += fullScreenToggle_Click;
    }
}

```

```

        close = new Button();
        close.Size = new Size(32, 17);
        close.FlatStyle = FlatStyle.Flat;
        close.BackgroundImage = Properties.Resources.Close_Display;
        close.MouseHover += (s, e) => close.BackgroundImage =
Properties.Resources.Close_Hover;
        close.MouseLeave += (s, e) => close.BackgroundImage =
Properties.Resources.Close_Display;
        close.Click += ((s, e) => Form.IOController.RemovePanel(this));

        header.Controls.Add(panelIconBox);
        header.Controls.Add(titleLabel);
        header.Controls.Add(hide);
        header.Controls.Add(fullScreenToggle);
        header.Controls.Add(close);

        this.Controls.Add(header);

        SetupContextMenu();
    }

    void fullScreenToggle_Click(object sender, EventArgs e)
    {
        if (windowFillStyle != FillStyles.FullScreen)
        {
            FillStyle = FillStyles.FullScreen;
        }
        else
        {
            FillStyle = FillStyles.None;
        }
    }

    void SetButtonImagesNotFullScreen()
    {
        fullScreenToggle.BackgroundImage =
StratSim.Properties.Resources.FullScreen_Display;
        fullScreenToggle.MouseHover += (a, b) => fullScreenToggle.BackgroundImage =
StratSim.Properties.Resources.FullScreen_Hover;
        fullScreenToggle.MouseLeave += (a, b) => fullScreenToggle.BackgroundImage =
StratSim.Properties.Resources.FullScreen_Display;
        fullScreenToggle.MouseHover -= (a, b) => fullScreenToggle.BackgroundImage =
StratSim.Properties.Resources.RestoreDown_Hover;
        fullScreenToggle.MouseLeave -= (a, b) => fullScreenToggle.BackgroundImage =
StratSim.Properties.Resources.RestoreDown_Display;
    }

    void SetButtonImagesIsFullScreen()
    {
        fullScreenToggle.BackgroundImage =
StratSim.Properties.Resources.RestoreDown_Display;
        fullScreenToggle.MouseHover += (a, b) => fullScreenToggle.BackgroundImage =
StratSim.Properties.Resources.RestoreDown_Hover;
        fullScreenToggle.MouseLeave += (a, b) => fullScreenToggle.BackgroundImage =
StratSim.Properties.Resources.RestoreDown_Display;
        fullScreenToggle.MouseHover -= (a, b) => fullScreenToggle.BackgroundImage =
StratSim.Properties.Resources.FullScreen_Hover;
        fullScreenToggle.MouseLeave -= (a, b) => fullScreenToggle.BackgroundImage =
StratSim.Properties.Resources.FullScreen_Display;
    }

    void header_MouseUp(object sender, MouseEventArgs e)
    {
        if (e.Button == MouseButtons.Left && PanelDropped != null)
            PanelDropped(e.Location);
    }

    void header_MouseDown(object sender, MouseEventArgs e)

```

```

{
    if (e.Button == System.Windows.Forms.MouseButtons.Left)
    {
        if (PanelSelected != null)
            PanelSelected();

        Form.IOController.StartReLayout(this, e.Location);
    }
}

public void OnPanelOpened()
{
    if (PanelOpened != null)
        PanelOpened();
}

public void OnPanelClosed()
{
    if (PanelClosed != null)
        PanelClosed();
}

/// <summary>
/// Removes the buttons displayed in the header, for use when the panel is
displayed in a content panel
/// </summary>
public void RemoveToolbarButtons()
{
    header.Controls.Remove(hide);
    header.Controls.Remove(fullScreenToggle);
    header.Controls.Remove(close);
    header.MouseDown -= header_MouseDown;
    header.MouseUp -= header_MouseUp;
}

/// <summary>
/// Positions the components in the header after a re-size
/// </summary>
public void PositionComponents()
{
    header.Size = new Size(this.ClientRectangle.Width, 25);

    int rightBorder = header.ClientRectangle.Right - 3;
    hide.Location = new Point(rightBorder - 66, 0);
    fullScreenToggle.Location = new Point(rightBorder - 49, 0);
    close.Location = new Point(rightBorder - 32, 0);
}

void SetupContextMenu()
{
    contextMenu = new MyContextMenu(this);
    header.ContextMenuStrip = contextMenu;
}

/// <returns>True if the dock type is associated with the top of the
panel</returns>
public static bool IsDockedAtTop(DockTypes dockType)
{
    bool dockedAtTop;

    if (dockType == DockTypes.TopLeft
        || dockType == DockTypes.Top
        || dockType == DockTypes.Top2
        || dockType == DockTypes.TopRight
        || dockType == DockTypes.Left
        || dockType == DockTypes.None)
    {
}

```

```

        dockedAtTop = true;
    }
    else
    {
        dockedAtTop = false;
    }

    return dockedAtTop;
}

/// <returns>True if the dock type is associated with the left of the
panel</returns>
public static bool IsDockedAtLeft(DockTypes dockType)
{
    bool dockedAtLeft;

    if (dockType == DockTypes.TopLeft
        || dockType == DockTypes.Left
        || dockType == DockTypes.BottomLeft
        || dockType == DockTypes.Top
        || dockType == DockTypes.None
        || dockType == DockTypes.Bottom)
    {
        dockedAtLeft = true;
    }
    else
    {
        dockedAtLeft = false;
    }

    return dockedAtLeft;
}

public delegate void PanelClosedEventHandler();
public event PanelClosedEventHandler PanelClosed;
public delegate void PanelOpenedEventHandler();
public event PanelOpenedEventHandler PanelOpened;

public delegate void PanelDroppedEventHandler(Point Location);
public event PanelDroppedEventHandler PanelDropped;

public delegate void PanelSelectedEventHandler();
public event PanelSelectedEventHandler PanelSelected;

public void OnOpenedInDifferentForm(MainForm Form)
{
    parentForm = Form;
    parentForm.IOController.AssociatedForm = Form;
    if (OpenedInNewForm != null)
        OpenedInNewForm(Form);
}

public delegate void NewFormEventHandler(MainForm NewForm);
public event NewFormEventHandler OpenedInNewForm;

public FillStyles FillStyle
{
    get { return windowFillStyle; }
    set
    {
        if (windowFillStyle != FillStyles.FullScreen && value ==
FillStyles.FullScreen)
        {
            SetButtonImagesIsFullScreen();
        }
        if (windowFillStyle == FillStyles.FullScreen && value !=
FillStyles.FullScreen)
    }
}

```

```

        {
            SetButtonImagesNotFullScreen();
        }
        windowFillStyle = value;
        FlowLayoutEvents.OnMainPanelLayoutChanged();
    }
}
public AutosizeTypes AutoSizeType
{
    get { return autoSize; }
    set
    {
        autoSize = value;
        FlowLayoutEvents.OnMainPanelLayoutChanged();
    }
}
public DockTypes DockType
{
    get { return windowDockType; }
    set
    {
        windowDockType = value;
        FlowLayoutEvents.OnMainPanelLayoutChanged();
    }
}
public Type Type
{
    get { return type; }
}
public Size OriginalSize
{
    get { return originalSize; }
    set { originalSize = value; }
}
/// <summary>
/// Gets or sets the location of the point that the panel was docked to.
/// </summary>
public Point DockPointLocation
{
    get { return dockPointLocation; }
    set { dockPointLocation = value; }
}
public Padding MyPadding
{
    get { return padding; }
    set { padding = value; }
}
public bool MyVisible
{
    get { return visible; }
    set
    {
        visible = value;
        FlowLayoutEvents.OnMainPanelLayoutChanged();
    }
}
public bool Removed
{
    get { return removed; }
    set { removed = value; }
}
public int PanelIndex
{
    get { return panelIndex; }
    set { panelIndex = value; }
}

```

```

public MainForm Form
{
    get { return parentForm; }
    set { parentForm = value; }
}

public bool InContentPanel
{
    get { return inContentPanel; }
    set { inContentPanel = value; }
}

public Image PanelIcon
{ get { return icon; } }

}

using StratSim.Model;
using StratSim.ViewModel;
using System;
using System.Collections.Generic;
using System.Drawing;
using System.Windows.Forms;

namespace StratSim.View.MyFlowLayout
{
    /// <summary>
    /// Provides functionality for controlling the contents of the window flow panel.
    /// </summary>
    public class MyToolbar : ToolStripContainer, IDockableControl
    {
        ToolStrip menuBar, windowsBar;
        ToolStripButton VersionInfo;

        ToolStripDropDownButton ShowPanels;
        List<ToolStripButton> PanelList = new List<ToolStripButton>();

        ToolStripDropDownButton QuickStart;
        ToolStripDropDownButton Parameters, Strategies;
        ToolStripButton Race, Archives;

        ToolStripDropDownButton OpenWindows;
        ToolStripButton Info, Settings, Main, ParameterViewer, StrategyViewer,
ArchiveViewer, Graph, Axes, DriverViewer;

        FillStyles windowFillStyle;
        AutosizeTypes autoSize;
        DockTypes windowDockType;
        Size originalSize;
        Point dockPointLocation;

        Type type;

        WindowFlowPanel _parent;
        MainForm form;

        bool removed;

        public MyToolbar(WindowFlowPanel parent)
        {
            SetupToolbar();
            windowFillStyle = FillStyles.FullWidth;
            autoSize = AutosizeTypes.Constant;
            windowDockType = DockTypes.TopLeft;
        }
    }
}

```

```

    _parent = parent;
    form = _parent._parent;
    removed = true;

    type = typeof(MyToolbar);

    originalSize = this.Size;

    form.PanelAdded += OnPanelAdded;
    FlowLayoutEvents.MainPanelLayoutChanged += GetShownHiddenPanels;
    MyEvents.ShowToolStrip += OnToolStripAdded;
    MyEvents.RemoveToolStrip += OnToolStripRemoved;
    MyEvents.FinishedLoadingParameters += OnPaceParametersFinishedLoading;
    MyEvents.FinishedLoadingStrategies += OnStrategiesFinishedLoading;
}

void OnToolStripAdded(ToolStripDropDownItem cm)
{
    AddToolStripToToolbar(cm);
}
void OnToolStripRemoved(ToolStripDropDownItem cm)
{
    RemoveToolStripFromToolbar(cm);
}

void OnPanelAdded(int panelIndex)
{
    AddPanel(panelIndex);
}

void SetupToolbar()
{
    PrepareContainerForToolBar();
    AddMenuStripToContainer();
    AddButtonsToToolBar();
}
void PrepareContainerForToolBar()
{
    this.Dock = DockStyle.Top;
    this.LeftToolStripPanelVisible = false;
    this.BottomToolStripPanelVisible = false;
    this.RightToolStripPanelVisible = false;
    this.Size = new Size(this.Width, 25);
}
void AddMenuStripToContainer()
{
    menuBar = new ToolStrip();
    menuBar.BackColor = Color.White;
    windowsBar = new ToolStrip();
    windowsBar.BackColor = Color.White;
    this.TopToolStripPanel.Controls.Add(windowsBar);
    this.TopToolStripPanel.Controls.Add(menuBar);
    this.TopToolStripPanel.BackColor = Color.White;
}

void AddButtonsToToolBar()
{
    //Show panels drop down
    ShowPanels = new ToolStripDropDownButton();
    menuBar.Items.Add(ShowPanels);
    ShowPanels.Text = "View";
    //Finish panels drop down

    //QuickStart
    QuickStart = new ToolStripDropDownButton();
    menuBar.Items.Add(QuickStart);
}

```

```

        QuickStart.Text = "Startup";

        Parameters = new ToolStripDropDownButton("Parameters",
Properties.Resources.Pace_Parameters);
        QuickStart.DropDownItems.Add(Parameters);
        ToolStripDropDownButton FromRaceData = new ToolStripDropDownButton("From
Race Data");
        Parameters.DropDownItems.Add(FromRaceData);
        FromRaceData.AutoSize = true;
        for (int trackIndex = 0; trackIndex < Data.NumberOfTracks; trackIndex++)
{
    RaceDropDown button = new RaceDropDown(trackIndex);
    FromRaceData.DropDownItems.Add(button);
    button.ButtonClicked += RaceIndexSelected;
}
ToolStripButtonFromFile = new ToolStripButton("From File");
Parameters.DropDownItems.AddFromFile);
FromFile.Click +=FromFile_Click;

Strategies = new ToolStripDropDownButton("Strategies",
Properties.Resources.Strategies);
Strategies.Visible = false;
QuickStart.DropDownItems.Add(Strategies);
ToolStripButton StrategiesFromFile = new ToolStripButton("From File");
Strategies.DropDownItems.Add(StrategiesFromFile);
StrategiesFromFile.Click += StrategiesFromFile_Click;
ToolStripButton StrategiesFromData = new ToolStripButton("From Pace Data");
Strategies.DropDownItems.Add(StrategiesFromData);
StrategiesFromData.Click += StrategiesFromData_Click;

Race = new ToolStripButton("Race", Properties.Resources.Race);
Race.Visible = false;
Race.Click += Race_Click;
QuickStart.DropDownItems.Add(Race);

Archives = new ToolStripButton("Archives", Properties.Resources.Archives);
Archives.Click += ArchivesReady;
QuickStart.DropDownItems.Add(Archives);

ToolStripButton DataInput = new ToolStripButton("Input Data",
Properties.Resources.Input);
DataInput.Click += DataInput_Click;
QuickStart.DropDownItems.Add(DataInput);
//Finish Quickstart

//Open Windows
ToolStripButton tempButton;

OpenWindows = new ToolStripDropDownButton();
OpenWindows.Text = "Open Windows";
menuBar.Items.Add(OpenWindows);

//Adds the buttons and event handlers for all panels that could be opened
by the user
tempButton = new ToolStripButton("Info", Properties.Resources.Info);
tempButton.AutoSize = true;
tempButton.Click += (s, e) => PanelControlEvents.OnShowInfoPanel(form);
Info = tempButton;
OpenWindows.DropDownItems.Add(Info);
tempButton = new ToolStripButton("Settings",
Properties.Resources.Settings);
tempButton.AutoSize = true;
tempButton.Click += (s, e) => PanelControlEvents.OnShowSettingsPanel(form);
Settings = tempButton;
OpenWindows.DropDownItems.Add(Settings);
tempButton = new ToolStripButton("Main", Properties.Resources.Main);
tempButton.AutoSize = true;

```

```

        tempButton.Click += (s, e) =>
PanelControlEvents.OnShowContentTabControl(form);
Main = tempButton;
OpenWindows.DropDownItems.Add(Main);
tempButton = new ToolStripButton("Data Input", Properties.Resources.Input);
tempButton.AutoSize = true;
tempButton.Click += (s, e) => PanelControlEvents.OnShowDataInput(form);
DataInput = tempButton;
OpenWindows.DropDownItems.Add(DataInput);
tempButton = new ToolStripButton("Parameter Viewer",
Properties.Resources.Pace_Parameters);
tempButton.AutoSize = true;
tempButton.Click += (s, e) =>
PanelControlEvents.OnShowPaceParameters(form);
ParameterViewer = tempButton;
OpenWindows.DropDownItems.Add(ParameterViewer);
tempButton = new ToolStripButton("Strategy Viewer",
Properties.Resources.Strategies);
tempButton.AutoSize = true;
tempButton.Click += (s, e) => PanelControlEvents.OnShowStrategies(form);
StrategyViewer = tempButton;
OpenWindows.DropDownItems.Add(StrategyViewer);
tempButton = new ToolStripButton("Archives",
Properties.Resources.Archives);
tempButton.AutoSize = true;
tempButton.Click += (s, e) => PanelControlEvents.OnShowArchives(form);
ArchiveViewer = tempButton;
OpenWindows.DropDownItems.Add(ArchiveViewer);
tempButton = new ToolStripButton("Graph", Properties.Resources.Graph);
tempButton.AutoSize = true;
tempButton.Click += (s, e) => PanelControlEvents.OnShowGraph(form);
Graph = tempButton;
OpenWindows.DropDownItems.Add(Graph);
tempButton = new ToolStripButton("Axes", Properties.Resources.Axes);
tempButton.AutoSize = true;
tempButton.Click += (s, e) => PanelControlEvents.OnShowAxes(form);
Axes = tempButton;
OpenWindows.DropDownItems.Add(Axes);
tempButton = new ToolStripButton("Drivers",
Properties.Resources.Driver_Select);
tempButton.AutoSize = true;
tempButton.Click += (s, e) =>
PanelControlEvents.OnShowDriverSelectPanel(form);
DriverViewer = tempButton;
OpenWindows.DropDownItems.Add(DriverViewer);
//Finish Windows

//Version info
VersionInfo = new ToolStripButton();
VersionInfo.Text = "Version";
VersionInfo.Click += VersionInfo_Click;
menuBar.Items.Add(VersionInfo);
//Finish version info
}

void OnPaceParametersFinishedLoading()
{
    Strategies.Visible = true;
}
void OnStrategiesFinishedLoading()
{
    Race.Visible = true;
}

void DataInput_Click(object sender, EventArgs e)
{
    PanelControlEvents.OnShowDataInput(form);
}

```

```

        }

        void VersionInfo_Click(object sender, EventArgs e)
        {
            PanelControlEvents.OnShowVersionInfo();
        }

        void ArchivesReady(object sender, EventArgs e)
        {
            PanelControlEvents.OnShowArchives(form);
        }

        void FromFile_Click(object sender, EventArgs e)
        {
            PanelControlEvents.OnLoadPaceParametersFromFile();
            PanelControlEvents.OnShowPaceParameters(form);
        }

        void RaceIndexSelected(int buttonIndex, Type buttonType)
        {
            Data.RaceIndex = buttonIndex;
            PanelControlEvents.OnLoadPaceParametersFromRace();
            PanelControlEvents.OnShowPaceParameters(form);
        }

        void StrategiesFromFile_Click(object sender, EventArgs e)
        {
            PanelControlEvents.OnLoadStrategiesFromFile();
            PanelControlEvents.OnShowStrategies(form);
        }

        void StrategiesFromData_Click(object sender, EventArgs e)
        {
            PanelControlEvents.OnLoadStrategiesFromData();
            PanelControlEvents.OnShowStrategies(form);
        }

        void Race_Click(object sender, EventArgs e)
        {
            PanelControlEvents.OnStartRaceFromStrategies();
        }

        void AddToolStripToToolbar(ToolStripDropDownItem toolStripItemToAdd)
        {
            windowsBar.Items.Add(toolStripItemToAdd);
        }

        void RemoveToolStripFromToolbar(ToolStripDropDownItem toolStripItemToRemove)
        {
            windowsBar.Items.Remove(toolStripItemToRemove);
        }

        /// <summary>
        /// Adds the functionality to control a panel from the toolbar
        /// </summary>
        /// <param name="panelIndex">The index of the panel to add with respect to the
        form</param>
        public void AddPanel(int panelIndex)
        {
            MyToolStripButton PanelViewButton = new MyToolStripButton(panelIndex);
            PanelViewButton.ButtonClicked += ShowHidePanel;
            PanelViewButton.Text = _parent.VisiblePanels[panelIndex].Name;
            PanelViewButton.Checked = _parent.VisiblePanels[panelIndex].MyVisible;
            PanelViewButton.DisplayStyle = ToolStripItemDisplayStyle.ImageAndText;
            PanelViewButton.Image = _parent.VisiblePanels[panelIndex].PanelIcon;
            PanelList.Add(PanelViewButton);
            ShowPanels.DropDownItems.Add(PanelViewButton);
        }

        /// <summary>
        /// Removes the show panel button for a panel from the toolbar
        /// </summary>
        /// <param name="panelIndex">The index of the panel to remove</param>
        public void RemovePanel(int panelIndex)
    }
}

```

```

    {
        ShowPanels.DropDownItems.Remove(PanelList[panelIndex]);
        PanelList.RemoveAt(panelIndex);
        foreach (MyToolStripButton t in PanelList)
        {
            if (t.ButtonIndex > panelIndex)
            {
                t.ButtonIndex--;
            }
        }
    }

    void ShowHidePanel(int buttonIndex, Type buttonType)
    {
        _parent.VisiblePanels[buttonIndex].MyVisible =
        PanelList[buttonIndex].Checked;
    }
    void GetShownHiddenPanels()
    {
        int panelIndex = 0;
        foreach (MyToolStripButton t in ShowPanels.DropDownItems)
        {
            t.Checked = _parent.VisiblePanels[panelIndex++].MyVisible;
        }
    }
}

public FillStyles FillStyle
{
    get { return windowFillStyle; }
    set { windowFillStyle = value; }
}
public AutosizeTypes AutoSizeType
{
    get { return autoSize; }
    set { autoSize = value; }
}
public DockTypes DockType
{
    get { return windowDockType; }
    set { windowDockType = value; }
}
public Type Type
{
    get { return type; } }
public Size OriginalSize
{
    get { return originalSize; }
    set { originalSize = value; }
}
public Point DockPointLocation
{
    get { return dockPointLocation; }
    set { dockPointLocation = value; }
}
public bool MyVisible
{
    get { return Visible; }
    set { Visible = value; }
}
public bool Removed
{
    get { return removed; }
    set { removed = value; }
}
}
}

```

```

using StratSim.Model;
using System;

namespace StratSim.View.MyFlowLayout
{
    /// <summary>
    /// Represents a tool strip button linked to a track.
    /// </summary>
    class RaceDropDown : MyToolStripButton
    {
        public RaceDropDown(int Index)
            : base(Index)
        {
            this.Text = Data.Tracks[Index].name;
        }

        public override void GenerateClickEvent(object sender, EventArgs e)
        {
            base.GenerateClickEvent(sender, e);
            this.Checked = false;
        }
    }
}

using System;
using System.Collections.Generic;
using System.Drawing;
using System.Windows.Forms;

namespace StratSim.View.MyFlowLayout
{
    /// <summary>
    /// Panel to be displayed on the main form.
    /// Contains methods for dynamically arranging 'MyPanel' constituents.
    /// Contains data about the panels on the form
    /// </summary>
    public class WindowFlowPanel : Panel
    {
        List<IDockableControl> dockableControlsInPanel = new List<IDockableControl>();
        //list of controls on form
        List<MyPanel> myPanelsOnForm = new List<MyPanel>();

        LayoutLines myLayoutLines;
        Rectangle remainingSize;

        public MainForm _parent;

        Dictionary<DockTypes, Point> dockPointLocations = new Dictionary<DockTypes, Point>();

        int panelsOnForm;
        int alexMcCormickBackgroundState;
        int easterEggBackgroundImage;

        public WindowFlowPanel(MainForm parentForm)
        {
            //Define the parent
            _parent = parentForm;

            //Add a background
            this.BackgroundImage = Properties.Resources.RedBull;
            this.BackgroundImageLayout = ImageLayout.Stretch;
            this.DoubleBuffered = true;

            //Locate this panel on the form
            Location = new Point(0, _parent.header.Height);
        }
    }
}

```

```

        Size = new Size(_parent.ClientRectangle.Width, _parent.ClientSize.Height -
_pparent.header.Height);

        //Define the initial dock points
SetInitialLocationsForDockPoints();

        //Setup local variables
panelsOnForm = 0;
remainingSize.Size = this.ClientSize;
remainingSize.Location = new Point(0, 0);
alexMcCormickBackgroundState = 0;
easterEggBackgroundImage = 0;
myLayoutLines = new LayoutLines(this.ClientRectangle);

        //Subscribe to events fired when layout changes
FlowLayoutEvents.MainPanelLayoutChanged += MyEvents_MainPanelLayoutChanged;
FlowLayoutEvents.MainPanelResizeEnd += FlowLayoutEvents_MainPanelResizeEnd;
}

void FlowLayoutEvents_MainPanelResizeEnd()
{
    this.Focus();
}

/// <summary>
/// Easter egg for changing the background of the panel
/// </summary>
internal void AlexMcCormickEasterEgg(Keys k)
{
    switch (alexMcCormickBackgroundState)
    {
        case 0:
        case 1:
        {
            if (k == Keys.Up) { alexMcCormickBackgroundState++; }
            else { alexMcCormickBackgroundState = 0; }
            break;
        }
        case 2:
        case 3:
        {
            if (k == Keys.Down) { alexMcCormickBackgroundState++; }
            else { alexMcCormickBackgroundState = 0; }
            break;
        }
        case 4:
        case 6:
        {
            if (k == Keys.Left) { alexMcCormickBackgroundState++; }
            else { alexMcCormickBackgroundState = 0; }
            break;
        }
        case 5:
        case 7:
        {
            if (k == Keys.Right) { alexMcCormickBackgroundState++; }
            else { alexMcCormickBackgroundState = 0; }
            break;
        }
        case 8:
        {
            if (k == Keys.B) { alexMcCormickBackgroundState++; }
            else { alexMcCormickBackgroundState = 0; }
            break;
        }
        case 9:
        {
    
```

```

        if (k == Keys.A)
        {
            switch (easterEggBackgroundImage)
            {
                case 0: this.BackgroundImage =
Properties.Resources.MercedesAMG; break;
                case 1: this.BackgroundImage = Properties.Resources.Ferrari;
break;
                case 2: this.BackgroundImage = Properties.Resources.Lotus;
break;
                case 3: this.BackgroundImage = Properties.Resources.Mclaren;
break;
                case 4: this.BackgroundImage =
Properties.Resources.ForceIndia; break;
                case 5: this.BackgroundImage = Properties.Resources.Sauber;
break;
                case 6: this.BackgroundImage =
Properties.Resources.TorroRosso; break;
                case 7: this.BackgroundImage = Properties.Resources.Williams;
break;
                case 8: this.BackgroundImage = Properties.Resources.Marussia;
break;
                case 9: this.BackgroundImage = Properties.Resources.Caterham;
break;
                case 10: this.BackgroundImage = Properties.Resources.RedBull;
break;
            }
            easterEggBackgroundImage = (easterEggBackgroundImage + 1) % 11;
alexMcCormickBackgroundState = 0;
        }
        else { alexMcCormickBackgroundState = 0; }
break;
    }
}

/// <summary>
/// Lays out the panel after a user change to the layout,
/// which requires a change to panel sizes or locations
/// </summary>
void MyEvents_MainPanelLayoutChanged()
{
    this.Size = new Size(_parent.ClientSize.Width, _parent.ClientSize.Height -
_parent.header.Height);

    remainingSize.Size = this.Size;
    remainingSize.Location = new Point(0, 0);

    //Re-layout the panel and resize components on it
    RepeatLayout();
    ResizeUsingLayoutLines();
}

/// <summary>
/// Should be called after panels have been added to the window requiring it
to be laid out again
/// </summary>
public void FinishedAddingPanels()
{
    //Set the size to the minimum size allowed
    foreach (IDockableControl c in dockableControlsInPanel)
    {
        ((Control)c).Size = c.OriginalSize;
    }

    //Set the remaining size on the panel to the size of the container
}

```

```

        remainingSize.Size = this.Size;
        remainingSize.Location = new Point(0, 0);

        //Re-layout the panel and resize components on it
        RepeatLayout();
        ResizeUsingLayoutLines();
    }

    /// <summary>
    /// Physically adds a control to the panel
    /// </summary>
    /// <param name="controlToAdd"></param>
    /// <param name="notAlreadyOnForm"></param>
    public void AddControl(IDockableControl controlToAdd, bool notAlreadyOnForm)
    {
        Control controlCopy = (Control)controlToAdd;

        if (notAlreadyOnForm) //if the panel is not already on the form, add it to
the form
        {
            dockableControlsInPanel.Add(controlToAdd);
            if (controlToAdd.Type == typeof(MyPanel))
            {
                myPanelsOnForm.Add((MyPanel)controlCopy);
                ((MyPanel)controlCopy).PanelIndex = panelsOnForm;
                _parent.OnPanelAdded(panelsOnForm++);
            }
        }

        if (controlToAdd.MyVisible) //if the panel is to be displayed
        {
            //add the control to the form
            this.Controls.Add(controlCopy);

            controlCopy.Size = controlToAdd.OriginalSize;
            controlCopy.Location = GetLocation(controlToAdd);

            //If the control takes the full size of the window, set the size now
            if (controlToAdd.FillStyle == FillStyles.FullWidth)
            {
                controlCopy.Width = remainingSize.Width;
                controlCopy.Left = remainingSize.Left;
            }
            if (controlToAdd.FillStyle == FillStyles.FullHeight)
            {
                controlCopy.Height = remainingSize.Height;
                controlCopy.Top = remainingSize.Top;
            }
            if (controlToAdd.FillStyle == FillStyles.FullScreen)
            {
                controlCopy.Location = remainingSize.Location;
                controlCopy.Size = remainingSize.Size;
            }

            //Updates the remaining size for other controls which take the full size
of the window
            UpdateRemainingSize(controlToAdd);
            //Update the dock point locations
            UpdatePoints(controlToAdd);

            //for the purposes of the layout lines:
            myLayoutLines.AddPanel(controlCopy, controlToAdd.DockType);
        }
    }

    /// <summary>
    /// Removes a control from the panel and therefore the form.

```

```

/// </summary>
/// <param name="panelToRemove">The control to remove from the form</param>
public void RemoveControl(IDockableControl panelToRemove)
{
    //Remove the control from the lists of controls to draw
    dockableControlsInPanel.Remove(panelToRemove);
    myPanelsOnForm.Remove((MyPanel)panelToRemove);
    this.Controls.Remove((MyPanel)panelToRemove);
    panelsOnForm--;
}

void SetInitialLocationsForDockPoints()
{
    Point topLeft = new Point(ClientRectangle.Left, ClientRectangle.Top);
    Point bottomLeft = new Point(ClientRectangle.Left, ClientRectangle.Bottom);
    Point bottomRight = new Point(ClientRectangle.Right,
ClientRectangle.Bottom);
    Point topRight = new Point(ClientRectangle.Right, ClientRectangle.Top);

    dockPointLocations[DockTypes.TopLeft] = topLeft; //Top Left
    dockPointLocations[DockTypes.Left] = topLeft; //Left
    dockPointLocations[DockTypes.BottomLeft] = bottomLeft; //BottomLeft
    dockPointLocations[DockTypes.Top] = topLeft; //Top
    dockPointLocations[DockTypes.None] = topLeft; //None
    dockPointLocations[DockTypes.Bottom] = bottomLeft; //Bottom
    dockPointLocations[DockTypes.TopRight] = topRight; //TopRight
    dockPointLocations[DockTypes.Right] = topRight; //Right
    dockPointLocations[DockTypes.BottomRight] = bottomRight; //BottomRight
    dockPointLocations[DockTypes.Top2] = topRight; //TopRightCentre
    dockPointLocations[DockTypes.Bottom2] = bottomRight; //BottomRightCentre
}

/// <summary>
/// Lays out the controls dynamically
/// Adds the controls to their respective lists for FillStyle
/// Adds the controls in order so that their maximum size is observed
/// Then adds all other controls using the layout lines to size the panels
/// </summary>
void RepeatLayout()
{
    List<IDockableControl> FullScreens = new List<IDockableControl>();
    List<IDockableControl> FullWidths = new List<IDockableControl>();
    List<IDockableControl> FullHeights = new List<IDockableControl>();
    List<IDockableControl> Others = new List<IDockableControl>();
    IDockableControl toolbar = null;

    myLayoutLines.ClearLinesInContainer(this.ClientRectangle);
    SetInitialLocationsForDockPoints();

    //Remove all controls and assign them to relevant lists
    foreach (IDockableControl d in dockableControlsInPanel)
    {
        this.Controls.Remove((Control)d);
        if (d.Type == typeof(MyToolbar)) { toolbar = d; }
        if (d.FillStyle == FillStyles.FullScreen) { FullScreens.Add(d); }
        if (d.FillStyle == FillStyles.FullWidth) { FullWidths.Add(d); }
        if (d.FillStyle == FillStyles.FullHeight) { FullHeights.Add(d); }
        if (d.FillStyle == FillStyles.None) { Others.Add(d); }
    }

    //Work through the lists adding controls again

    //If any control is fullscreen, only it and the toolbar should be added
    if (FullScreens.Count != 0)
    {
        AddControl(toolbar, false);
        foreach (IDockableControl d in FullScreens)

```

```

        {
            AddControl(d, false);
        }
    }
    else
    {
        foreach (IDockableControl d in FullWidths)
        {
            AddControl(d, false);
        }
        foreach (IDockableControl d in FullHeights)
        {
            AddControl(d, false);
        }
        foreach (IDockableControl d in Others)
        {
            AddControl(d, false);
        }
    }
}

/// <returns>The location at which to dock the panel at</returns>
Point GetLocation(IDockableControl c)
{
    Control controlCopy = (Control)c;

    Point drawPoint = new Point(); //point at which to draw control
    Point dockPoint = new Point(); //point at which control is docked.

    DockTypes DockType = c.DockType;

    dockPoint = dockPointLocations[DockType];

    //set the property of the control to the location at which it is docked.
    c.DockPointLocation = dockPoint;

    //Finding controlCopy.Location:
    //if docked at bottom, drawpoint is higher than dockpoint
    if ((DockType == DockTypes.Bottom) || (DockType == DockTypes.BottomLeft) ||
(DockType == DockTypes.BottomRight) || (DockType == DockTypes.Bottom2))
    {
        drawPoint.Y = dockPoint.Y - controlCopy.Height;
    }
    else //drawpoint.Y is the same as dockpoint.Y
    {
        drawPoint.Y = dockPoint.Y;
    }

    //if docked at right, drawpoint is to the left of dockpoint
    if ((DockType == DockTypes.TopRight) || (DockType == DockTypes.Right) ||
(DockType == DockTypes.BottomRight) || (DockType == DockTypes.Bottom2) || (DockType ==
== DockTypes.Top2))
    {
        drawPoint.X = dockPoint.X - controlCopy.Width;
    }
    else
    {
        drawPoint.X = dockPoint.X;
    }

    return drawPoint;
}

void UpdatePoints(IDockableControl controlBeingAdded)
{
    Control controlCopy = (Control)controlBeingAdded;
}

```

```

    //Depending on how the control fills the screen,
    //update the location of the dock points
    switch (controlBeingAdded.FillStyle)
    {
        case FillStyles.FullWidth:
        {
            //Change the location of the dock points that are affected by the
            control
            if (MyPanel.IsDockedAtTop(controlBeingAdded.DockType))
            {
                dockPointLocations[DockTypes.TopLeft] =
                AddHeightToDockPoint(DockTypes.TopLeft, controlCopy.Height);
                dockPointLocations[DockTypes.Left] =
                AddHeightToDockPoint(DockTypes.Left, controlCopy.Height);
                dockPointLocations[DockTypes.Top] =
                AddHeightToDockPoint(DockTypes.Top, controlCopy.Height);
                dockPointLocations[DockTypes.None] =
                AddHeightToDockPoint(DockTypes.None, controlCopy.Height);
                dockPointLocations[DockTypes.TopRight] =
                AddHeightToDockPoint(DockTypes.TopRight, controlCopy.Height);
                dockPointLocations[DockTypes.Right] =
                AddHeightToDockPoint(DockTypes.Right, controlCopy.Height);
                dockPointLocations[DockTypes.Top2] =
                AddHeightToDockPoint(DockTypes.Top2, controlCopy.Height);
            }
            else
            {
                dockPointLocations[DockTypes.BottomLeft] =
                AddHeightToDockPoint(DockTypes.BottomLeft, -controlCopy.Height);
                dockPointLocations[DockTypes.Bottom] =
                AddHeightToDockPoint(DockTypes.Bottom, -controlCopy.Height);
                dockPointLocations[DockTypes.BottomRight] =
                AddHeightToDockPoint(DockTypes.BottomRight, -controlCopy.Height);
                dockPointLocations[DockTypes.Bottom2] =
                AddHeightToDockPoint(DockTypes.Bottom2, -controlCopy.Height);
            }
            break;
        }

        case FillStyles.FullHeight:
        {
            if (MyPanel.IsDockedAtLeft(controlBeingAdded.DockType))
            {
                dockPointLocations[DockTypes.TopLeft] =
                AddWidthToDockPoint(DockTypes.TopLeft, controlCopy.Width);
                dockPointLocations[DockTypes.Left] =
                AddWidthToDockPoint(DockTypes.Left, controlCopy.Width);
                dockPointLocations[DockTypes.BottomLeft] =
                AddWidthToDockPoint(DockTypes.BottomLeft, controlCopy.Width);
                dockPointLocations[DockTypes.Top] =
                AddWidthToDockPoint(DockTypes.Top, controlCopy.Width);
                dockPointLocations[DockTypes.None] =
                AddWidthToDockPoint(DockTypes.None, controlCopy.Width);
                dockPointLocations[DockTypes.Bottom] =
                AddWidthToDockPoint(DockTypes.Bottom, controlCopy.Width);
            }
            else
            {
                dockPointLocations[DockTypes.TopRight] =
                AddWidthToDockPoint(DockTypes.TopRight, -controlCopy.Width);
                dockPointLocations[DockTypes.Right] =
                AddWidthToDockPoint(DockTypes.Right, -controlCopy.Width);
                dockPointLocations[DockTypes.BottomRight] =
                AddWidthToDockPoint(DockTypes.BottomRight, -controlCopy.Width);
                dockPointLocations[DockTypes.Top2] =
                AddWidthToDockPoint(DockTypes.Top2, -controlCopy.Width);
            }
        }
    }
}

```

```

        dockPointLocations[DockTypes.Bottom2] =
AddWidthToDockPoint(DockTypes.Bottom2, -controlCopy.Width);
    }
    break;
}
case FillStyles.None:
{
    //If the panel being added will affect the dock point:
    if ((controlBeingAdded.DockPointLocation ==
dockPointLocations[DockTypes.TopLeft]) || (controlBeingAdded.DockPointLocation ==
dockPointLocations[DockTypes.Left]) || (controlBeingAdded.DockPointLocation ==
dockPointLocations[DockTypes.None]))
    {
        dockPointLocations[DockTypes.TopLeft] =
AddHeightToDockPoint(DockTypes.TopLeft, controlCopy.Height);
        dockPointLocations[DockTypes.Left] =
AddHeightToDockPoint(DockTypes.Left, controlCopy.Height);
        dockPointLocations[DockTypes.None] =
AddHeightToDockPoint(DockTypes.None, controlCopy.Height);
    }
    if (controlBeingAdded.DockPointLocation ==
dockPointLocations[DockTypes.BottomLeft])
    {
        dockPointLocations[DockTypes.BottomLeft] =
AddHeightToDockPoint(DockTypes.BottomLeft, -controlCopy.Height);
    }
    if (controlBeingAdded.DockPointLocation ==
dockPointLocations[DockTypes.Top])
    {
        dockPointLocations[DockTypes.Top] =
AddWidthToDockPoint(DockTypes.Top, controlCopy.Width);
    }
    if (controlBeingAdded.DockPointLocation ==
dockPointLocations[DockTypes.Bottom])
    {
        dockPointLocations[DockTypes.Bottom] =
AddWidthToDockPoint(DockTypes.Bottom, controlCopy.Width);
    }
    if (controlBeingAdded.DockPointLocation ==
dockPointLocations[DockTypes.TopRight])
    {
        dockPointLocations[DockTypes.TopRight] =
AddHeightToDockPoint(DockTypes.TopRight, controlCopy.Height);
    }
    if (controlBeingAdded.DockPointLocation ==
dockPointLocations[DockTypes.Right])
    {
        dockPointLocations[DockTypes.Right] =
AddHeightToDockPoint(DockTypes.Right, controlCopy.Height);
    }
    if (controlBeingAdded.DockPointLocation ==
dockPointLocations[DockTypes.BottomRight])
    {
        dockPointLocations[DockTypes.BottomRight] =
AddHeightToDockPoint(DockTypes.BottomRight, -controlCopy.Height);
    }
    if (controlBeingAdded.DockPointLocation ==
dockPointLocations[DockTypes.Top2])
    {
        dockPointLocations[DockTypes.Top2] =
AddWidthToDockPoint(DockTypes.Top2, -controlCopy.Width);
    }
    if (controlBeingAdded.DockPointLocation ==
dockPointLocations[DockTypes.Bottom2])
    {
        dockPointLocations[DockTypes.Bottom2] =
AddWidthToDockPoint(DockTypes.Bottom2, -controlCopy.Width);
    }
}

```

```

        }
        break;
    }
}
/// <summary>
/// Updates the rectangle of remaining area on the panel after a control is
added
/// </summary>
/// <param name="addedControl">The control that was added</param>
void UpdateRemainingSize(IDockableControl addedControl)
{
    Control controlCopy = (Control)addedControl;

    if (addedControl.FillStyle == FillStyles.FullHeight)
    {
        remainingSize.Width -= controlCopy.Width;
        if (MyPanel.IsDockedAtLeft(addedControl.DockType))
        {
            remainingSize.Location = new Point(remainingSize.Left +
controlCopy.Width, remainingSize.Top);
        }
    }
    if (addedControl.FillStyle == FillStyles.FullWidth)
    {
        remainingSize.Height -= controlCopy.Height;
        if (MyPanel.IsDockedAtLeft(addedControl.DockType))
        {
            remainingSize.Location = new Point(remainingSize.Left,
remainingSize.Top + controlCopy.Height);
        }
    }
}

/// <summary>
/// Resizes all panels on the form using the layout lines
/// All panels are displayed at the maximum size possible
/// </summary>
void ResizeUsingLayoutLines()
{
    Dictionary<Locations, int> indicesOfLinesAroundPanel;
    Dictionary<Locations, bool> canSizeInDirection;

    foreach (IDockableControl controlToSize in dockableControlsInPanel)
    {
        indicesOfLinesAroundPanel = new Dictionary<Locations, int>();
        canSizeInDirection = new Dictionary<Locations, bool>();
        int newPositionOfEdge;

        Control copyOfControl = (Control)controlToSize;
        if (controlToSize.AutoSizeType != AutosizeTypes.Constant)
        {
            //Check around the panel
            foreach (var direction in
(Locations[])Enum.GetValues(typeof(Locations)))
            {
                indicesOfLinesAroundPanel[direction] =
myLayoutLines.GetLineIndex(myLayoutLines.AddLineFromSideOfPanel(copyOfControl,
direction), InvertLocation(direction));

                //The panel can size if a layout line exists that matches the side
of the panel,
                //and the autosize type allows the panel to be resized
                canSizeInDirection[direction] =
((indicesOfLinesAroundPanel[direction] != -1)
                    && AutosizeTypeAllowsResize(direction, controlToSize));
            }
        }
    }
}

```

```

        //Size the panel in each direction
        foreach (var direction in
(Locations[])Enum.GetValues(typeof(Locations)))
{
    if (canSizeInDirection[direction])
    {
        //Gets the furthest position to which a panel can be resized.
        newPositionOfEdge =
myLayoutLines.GetSizeChangeAllowed(copyOfControl, direction);

        switch (direction)
        {
            case Locations.Top:
            {
                //Resize the panel by the required amount
                //The height is increased by the difference between the
possible position
                newPositionOfEdge;
                the same amount
                newPositionOfEdge;

                //Move the relevant layout line to the new position
                if (indicesOfLinesAroundPanel[Locations.Top] != -1)
                {

                    myLayoutLines.LinesForLayout[Locations.Bottom][(indicesOfLinesAroundPanel[Locations.Top])].StaticLocation = newPositionOfEdge;
                }
                //Update the length of the lines to the sides of the
panel.
                if (indicesOfLinesAroundPanel[Locations.Right] != -1)
                {
                    //Set the start point of the line to the side to the
new position
                    myLayoutLines.LinesForLayout[Locations.Left][(indicesOfLinesAroundPanel[Locations.Right])].setStartPointY(newPositionOfEdge);
                    //Set the end point of the line to the side to the
new position,
                    //maintaining the link between the start point of one
line and the end point of the previous line.
                    if (indicesOfLinesAroundPanel[Locations.Right] > 0)

                        myLayoutLines.LinesForLayout[Locations.Left][(indicesOfLinesAroundPanel[Locations.Right]) - 1].setEndPointY(newPositionOfEdge);
                    }
                    if (indicesOfLinesAroundPanel[Locations.Left] != -1)
                    {

                        myLayoutLines.LinesForLayout[Locations.Right][(indicesOfLinesAroundPanel[Locations.Left])].setStartPointY(newPositionOfEdge);
                        if (indicesOfLinesAroundPanel[Locations.Left] > 0)

                            myLayoutLines.LinesForLayout[Locations.Right][(indicesOfLinesAroundPanel[Locations.Left]) - 1].setEndPointY(newPositionOfEdge);
                        }

                        break;
                    }
                    case Locations.Left: //See comments in above sections
                    {
                        copyOfControl.Width += copyOfControl.Left -
newPositionOfEdge;
                    }
                }
            }
        }
    }
}

```

```

        copyOfControl.Left -= copyOfControl.Left -
newPositionOfEdge;

        if (indicesOfLinesAroundPanel[Locations.Left] != -1)
        {

            myLayoutLines.LinesForLayout[Locations.Right][(indicesOfLinesAroundPanel[Location
s.Left])].StaticLocation = newPositionOfEdge;
            }
            if (indicesOfLinesAroundPanel[Locations.Bottom] != -1)
            {

                myLayoutLines.LinesForLayout[Locations.Top][(indicesOfLinesAroundPanel[Locations.
Bottom])].setStartPointX(newPositionOfEdge);
                if (indicesOfLinesAroundPanel[Locations.Bottom] > 0)

                    myLayoutLines.LinesForLayout[Locations.Top][(indicesOfLinesAroundPanel[Locations.
Bottom]) - 1].setEndPointX(newPositionOfEdge);
                    }
                    if (indicesOfLinesAroundPanel[Locations.Top] != -1)
                    {

                        myLayoutLines.LinesForLayout[Locations.Bottom][(indicesOfLinesAroundPanel[Locatio
ns.Top])].setStartPointX(newPositionOfEdge);
                        if (indicesOfLinesAroundPanel[Locations.Top] > 0)

                            myLayoutLines.LinesForLayout[Locations.Bottom][(indicesOfLinesAroundPanel[Locatio
ns.Top]) - 1].setEndPointX(newPositionOfEdge);
                            }

                            break;
                        }
                        case Locations.Bottom: //See comments in above sections
                        {
                            copyOfControl.Height += newPositionOfEdge -
copyOfControl.Bottom;

                            if (indicesOfLinesAroundPanel[Locations.Bottom] != -1)

                                myLayoutLines.LinesForLayout[Locations.Top][(indicesOfLinesAroundPanel[Locations.
Bottom])].StaticLocation = newPositionOfEdge;
                                }
                                if (indicesOfLinesAroundPanel[Locations.Left] != -1)
                                {

                                    myLayoutLines.LinesForLayout[Locations.Right][(indicesOfLinesAroundPanel[Location
s.Left])].setEndPointY(newPositionOfEdge);
                                    if (indicesOfLinesAroundPanel[Locations.Left] <
myLayoutLines.LinesForLayout[Locations.Right].Count - 1)

                                        myLayoutLines.LinesForLayout[Locations.Right][(indicesOfLinesAroundPanel[Location
s.Left]) + 1].setStartPointY(newPositionOfEdge);
                                        }
                                        if (indicesOfLinesAroundPanel[Locations.Right] != -1)
                                        {

                                            myLayoutLines.LinesForLayout[Locations.Left][(indicesOfLinesAroundPanel[Locations.
Right])].setEndPointY(newPositionOfEdge);
                                            if (indicesOfLinesAroundPanel[Locations.Right] <
myLayoutLines.LinesForLayout[Locations.Left].Count - 1)

                                                myLayoutLines.LinesForLayout[Locations.Left][(indicesOfLinesAroundPanel[Locations
.Right]) + 1].setStartPointY(newPositionOfEdge);
                                                }

                                                break;
}

```

```

        }
        case Locations.Right: //See comments in above sections
        {
            copyOfControl.Width += newPositionOfEdge -
copyOfControl.Right;

            if (indicesOfLinesAroundPanel[Locations.Right] != -1)
            {

                myLayoutLines.LinesForLayout[Locations.Left][(indicesOfLinesAroundPanel[Locations.Right])].StaticLocation = newPositionOfEdge;
            }
            if (indicesOfLinesAroundPanel[Locations.Top] != -1)
            {

                myLayoutLines.LinesForLayout[Locations.Bottom][(indicesOfLinesAroundPanel[Locations.Top])].setEndPointX(newPositionOfEdge);
                if (indicesOfLinesAroundPanel[Locations.Top] <
myLayoutLines.LinesForLayout[Locations.Bottom].Count - 1)

                    myLayoutLines.LinesForLayout[Locations.Bottom][(indicesOfLinesAroundPanel[Locations.Top]) + 1].setStartPointX(newPositionOfEdge);
                }
                if (indicesOfLinesAroundPanel[Locations.Bottom] != -1)
                {

                    myLayoutLines.LinesForLayout[Locations.Top][(indicesOfLinesAroundPanel[Locations.Bottom])].setEndPointX(newPositionOfEdge);
                    if (indicesOfLinesAroundPanel[Locations.Bottom] <
myLayoutLines.LinesForLayout[Locations.Top].Count - 1)

                        myLayoutLines.LinesForLayout[Locations.Top][(indicesOfLinesAroundPanel[Locations.Bottom]) + 1].setStartPointX(newPositionOfEdge);
                    }
                }

                break;
            }
        }
    }

    FlowLayoutEvents.OnMainPanelResize();
}

/// <summary>
/// Checks if the AutoSizeType for a panel allows it to resize in the
specified direction
/// </summary>
/// <param name="direction">The direction the panel is to be sized in</param>
/// <param name="controlToResize">The control to be resized</param>
/// <returns>True if the panel can be resized in the specified
direction</returns>
static bool AutosizeTypeAllowsResize(Locations direction, IDockableControl
controlToResize)
{
    bool typeCorrect = false;

    if (controlToResize.Auto.SizeType == AutosizeTypes.AutoHeight)
    {
        if (direction == Locations.Top || direction == Locations.Bottom)
        {
            typeCorrect = true;
        }
    }
}

```

```

        }
        if (controlToResize.AutoSizeType == AutosizeTypes.AutoWidth)
        {
            if (direction == Locations.Left || direction == Locations.Right)
            {
                typeCorrect = true;
            }
        }
        if (controlToResize.AutoSizeType == AutosizeTypes.Free)
        {
            typeCorrect = true;
        }

        return typeCorrect;
    }
    /// <returns>The direction that is opposite to the specified
    direction</returns>
    static Locations InvertLocation(Locations location)
    {
        Locations returnLocation;

        switch (location)
        {
            case Locations.Top: returnLocation = Locations.Bottom; break;
            case Locations.Left: returnLocation = Locations.Right; break;
            case Locations.Bottom: returnLocation = Locations.Top; break;
            case Locations.Right: returnLocation = Locations.Left; break;
            default: returnLocation = Locations.Top; break;
        }

        return returnLocation;
    }

    /// <summary>
    /// Adds a height, in pixels, to the y-coordinate of the dock point at the
    specified dock type
    /// </summary>
    /// <returns>The new point with the height added</returns>
    Point AddHeightToDockPoint(DockTypes dockType, int heightToAdd)
    {
        return new Point(dockPointLocations[dockType].X,
dockPointLocations[dockType].Y + heightToAdd);
    }
    /// <summary>
    /// Adds a width, in pixels, to the x-coordinate of the dock point at the
    specified dock type
    /// </summary>
    /// <returns>The new point with the width added</returns>

    Point AddWidthToDockPoint(DockTypes dockType, int widthToAdd)
    {
        return new Point(dockPointLocations[dockType].X + widthToAdd,
dockPointLocations[dockType].Y);
    }

    public List<MyPanel> VisiblePanels
    {
        get { return myPanelsOnForm; }
        set { myPanelsOnForm = value; }
    }
    public int PanelsOnForm
    {
        get { return panelsOnForm; }
        set { panelsOnForm = value; }
    }
    public Dictionary<DockTypes, Point> DockPoints

```

```
{ get { return dockPointLocations; } }
```

StratSim.View.Panels

```

using StratSim.Model;
using StratSim.View.MyFlowLayout;
using StratSim.View.UserControls;
using StratSim.ViewModel;
using System;
using System.Drawing;
using System.Windows.Forms;

namespace StratSim.View.Panels
{
    /// <summary>
    /// Displays data about graph axes on a panel,
    /// and provides methods for modifying the axes
    /// </summary>
    public class AxesWindow : MyPanel
    {
        TableLayoutPanel tableOfParameters;
        Label horizontalAxisTitle, verticalAxisTitle;
        Label Offset, ScaleFactor, StartPoint, LabelSpacing;
        TextBox[,] axisParameterDataBoxes;
        Button OK, Cancel;
        ComboBox NormalisationTypeSelected;

        axisParameters localHorizontalAxis, localVerticalAxis;
        NormalisationType localNormalisationType;

        public AxesWindow(MainForm FormToAdd)
            : base(250, 250, "Axes", FormToAdd, Properties.Resources.Axes)
        {
            InitialiseControls();
            AddControls();

            FlowLayoutEvents.MainPanelResizeEnd += MyEvents_MainPanelResize;
            MyEvents.AxesModified += MyEvents_AxesModified;

            SetPanelProperties(DockTypes.Bottom, AutosizeTypes.Constant,
FillStyles.None, this.Size);
        }

        void MyEvents_AxesModified(axisParameters XAxis, axisParameters YAxis,
NormalisationType normalisation, bool UserModified)
        {
            if (!UserModified) //If the axes have been modified outside of the panel
            {
                localHorizontalAxis = XAxis;
                localVerticalAxis = YAxis;
                PopulateAxesBoxes(XAxis, YAxis, normalisation);
            }
        }

        void MyEvents_MainPanelResize()
        {
            SizeComponent();
        }

        void SizeComponent()
        {
            tableOfParameters.Width = this.ClientSize.Width - 20;
            tableOfParameters.Height = this.ClientSize.Height - 40;
            OK.Location = new Point(this.Width - 160, this.Height - 40);
            Cancel.Location = new Point(this.Width - 80, this.Height - 40);
        }

        /// <summary>
        /// Sets up the controls to be displayed on the panel
    }
}

```

```

/// </summary>
void InitialiseControls()
{
    Size ControlDefault = new Size(70, 25);
    MyToolTip toolTip;

    tableOfParameters = new TableLayoutPanel();
    tableOfParameters.ColumnCount = 3;
    tableOfParameters.RowCount = 5;
    tableOfParameters.Location = new Point(0, 25);

    OK = new Button();
    OK.Size = new Size(60, 25);
    OK.Text = "OK";
    OK.FlatStyle = FlatStyle.Flat;
    OK.Click += OK_Click;
    Cancel = new Button();
    Cancel.Size = new Size(60, 25);
    Cancel.Text = "Cancel";
    Cancel.FlatStyle = FlatStyle.Flat;
    Cancel.Click += Cancel_Click;
    SizeComponent();

    horizontalAxisTitle = new Label();
    horizontalAxisTitle.Text = "X axis";
    horizontalAxisTitle.Size = ControlDefault;

    verticalAxisTitle = new Label();
    verticalAxisTitle.Text = "Y axis";
    verticalAxisTitle.Size = ControlDefault;

    Offset = new Label();
    Offset.Text = "Offset";
    Offset.Size = ControlDefault;
    toolTip = new MyToolTip(Offset, "Start location of the axis in units");

    ScaleFactor = new Label();
    ScaleFactor.Text = "Scale";
    ScaleFactor.Size = ControlDefault;
    toolTip = new MyToolTip(ScaleFactor, "Pixels per unit scale of the axis");

    StartPoint = new Label();
    StartPoint.Text = "Start Location";
    StartPoint.Size = ControlDefault;
    toolTip = new MyToolTip(StartPoint, "Start location of the axis in pixels,
with \r\nrespect to the top left corner of the panel");

    LabelSpacing = new Label();
    LabelSpacing.Text = "Label Spacing";
    LabelSpacing.Size = ControlDefault;
    toolTip = new MyToolTip(LabelSpacing, "Unit spacing of labels on the
axis");

    NormalisationTypeSelected = new ComboBox();
    foreach (var n in
(NormalisationType[])Enum.GetValues(typeof(NormalisationType)))
    {
        NormalisationTypeSelected.Items.Add(Convert.ToString(n));
    }
    NormalisationTypeSelected.Location = new Point(50, 180);
    NormalisationTypeSelected.DropDownStyle = ComboBoxStyle.DropDownList;
    NormalisationTypeSelected.FlatStyle = FlatStyle.Flat;
    toolTip = new MyToolTip(NormalisationTypeSelected, "Normalisation type of
the graph, either on an average or on every lap");

    axisParameterDataBoxes = new TextBox[2, 4];
    for (int column = 0; column <= 1; column++)

```

```

{
    for (int j = 0; j <= 3; j++)
    {
        axisParameterDataBoxes[column, j] = new TextBox();
        axisParameterDataBoxes[column, j].Size = ControlDefault;
    }
}

/// <summary>
/// Displays the controls on the panel
/// </summary>
void AddControls()
{
    tableOfParameters.Controls.Add(horizontalAxisTitle, 1, 0);
    tableOfParameters.Controls.Add(verticalAxisTitle, 2, 0);
    tableOfParameters.Controls.Add(Offset, 0, 1);
    tableOfParameters.Controls.Add(ScaleFactor, 0, 2);
    tableOfParameters.Controls.Add(StartPoint, 0, 3);
    tableOfParameters.Controls.Add(LabelSpacing, 0, 4);

    for (int columnIndex = 0; columnIndex <= 1; columnIndex++)
    {
        for (int rowIndex = 0; rowIndex <= 3; rowIndex++)
        {
            tableOfParameters.Controls.Add(axisParameterDataBoxes[columnIndex,
rowIndex], columnIndex + 1, rowIndex + 1);
        }
    }

    this.Controls.Add(OK);
    this.Controls.Add(Cancel);

    this.Controls.Add(NormalisationTypeSelected);

    this.Controls.Add(tableOfParameters);
}

void Cancel_Click(object sender, EventArgs e)
{
    //Populates the text boxes from the file, resetting the values
    PopulateAxesBoxes(localHorizontalAxis, localVerticalAxis,
localNormalisationType);
}

void OK_Click(object sender, EventArgs e)
{
    //Validate the input
    bool incorrectValue = false;
    PopulateAxisValues(ref incorrectValue);

    if (!incorrectValue) //If all values are within reasonable bounds
    {
        //Modify the axes
        MyEvents.OnAxesModified(localHorizontalAxis, localVerticalAxis,
localNormalisationType, true);
    }
}

/// <summary>
/// Populates the text boxes on the panel with data from the given axes
/// </summary>
void PopulateAxesBoxes(axisParameters horizontalAxis, axisParameters
verticalAxis, NormalisationType normalisationType)
{
}

```

```

        axisParameterDataBoxes[0, 0].Text =
Convert.ToString(horizontalAxis.baseOffset);
        axisParameterDataBoxes[0, 1].Text =
Convert.ToString(horizontalAxis.scaleFactor);
        axisParameterDataBoxes[0, 2].Text =
Convert.ToString(horizontalAxis.startLocation);
        axisParameterDataBoxes[0, 3].Text =
Convert.ToString(horizontalAxis.axisLabelSpacing);

        axisParameterDataBoxes[1, 0].Text =
Convert.ToString(verticalAxis.baseOffset);
        axisParameterDataBoxes[1, 1].Text =
Convert.ToString(verticalAxis.scaleFactor);
        axisParameterDataBoxes[1, 2].Text =
Convert.ToString(verticalAxis.startLocation);
        axisParameterDataBoxes[1, 3].Text =
Convert.ToString(verticalAxis.axisLabelSpacing);

    NormalisationTypeSelected.SelectedIndex = (int)normalisationType;
}

/// <summary>
/// Validates the data in the text boxes,
/// and populates the local axis data with the data from the text boxes
/// </summary>
/// <param name="incorrectValue">Returns true if any one of the data items is
invalid</param>
void PopulateAxisValues(ref bool incorrectValue)
{
    localHorizontalAxis.baseOffset =
(int)Functions.ValidateBetweenInclusive(Convert.ToInt32(axisParameterDataBoxes[0,
0].Text), 0, Data.GetRaceLaps(), "Horizontal Axis Base Offset", ref incorrectValue,
true);
    localHorizontalAxis.scaleFactor =
Functions.ValidateGreaterThanOrEqualTo(Convert.ToSingle(axisParameterDataBoxes[0,
1].Text), 0, "Horizontal Axis Scale", ref incorrectValue, true);
    localHorizontalAxis.startLocation =
(int)Functions.ValidateGreaterThan(Convert.ToInt32(axisParameterDataBoxes[0,
2].Text), 0, "Horizontal Axis Start", ref incorrectValue, true);
    localHorizontalAxis.axisLabelSpacing =
(int)Functions.ValidateGreaterThan(Convert.ToInt32(axisParameterDataBoxes[0,
3].Text), 1, "Horizontal Axis Label Spacing", ref incorrectValue, true);

    localVerticalAxis.baseOffset =
(int)Functions.ValidateBetweenInclusive(Convert.ToInt32(axisParameterDataBoxes[1,
0].Text), -75, 75, "Vertical Axis Base Offset", ref incorrectValue, true);
    localVerticalAxis.scaleFactor =
Functions.ValidateGreaterThanOrEqualTo(Convert.ToSingle(axisParameterDataBoxes[1,
1].Text), 0, "Vertical Axis Scale", ref incorrectValue, true);
    localVerticalAxis.startLocation =
(int)Functions.ValidateGreaterThan(Convert.ToInt32(axisParameterDataBoxes[1,
2].Text), 0, "Vertical Axis Start", ref incorrectValue, true);
    localVerticalAxis.axisLabelSpacing =
(int)Functions.ValidateGreaterThan(Convert.ToInt32(axisParameterDataBoxes[1,
3].Text), 1, "Vertical Axis Label Spacing", ref incorrectValue, true);

    localNormalisationType =
(NormalisationType)NormalisationTypeSelected.SelectedIndex;
}
}

using StratSim.Model;
using StratSim.Model.Files;

```

```

using StratSim.View.MyFlowLayout;
using StratSim.ViewModel;
using System;
using System.Diagnostics;
using System.Drawing;
using System.IO;
using System.Windows.Forms;

namespace StratSim.View.Panels
{
    /// <summary>
    /// Control to be displayed in the content tab control allowing
    /// data to be loaded from PDF files
    /// </summary>
    public class DataInput : MyPanel
    {
        //properties
        const int leftBorder = 10;
        const int topBorder = 10;
        const int defaultHeight = 15;
        const int defaultWidth = 200;

        Size cbxDefault = new Size(defaultWidth, defaultHeight);
        Size btnDefault = new Size(100, 25);

        TimingDataType fileType = TimingDataType.LapTimeData;
        string dataFromClipboard = "";

        DataController DataController = new DataController();

        //objects
        ComboBox SelectRace = new ComboBox();
        ComboBox SelectSession = new ComboBox();
        ComboBox SelectData = new ComboBox();
        Label EnterText = new Label();
        Button SelectText = new Button();
        Button AnalyseData = new Button();

        /// <summary>
        /// Starts the data input panel with nothing selected
        /// </summary>
        public DataInput(MainForm FormToAdd)
            : base(450, 240, "Load Data", FormToAdd, Properties.Resources.Input)
        {
            InitialiseObjects();
            LoadTrackList();
        }

        /// <summary>
        /// Starts the data input panel with the specified round and session selected
        /// in the combo boxes
        /// </summary>
        public DataInput(int roundIndex, int sessionIndex, MainForm FormToAdd)
            : base(450, 250, Data.Tracks[roundIndex].name + " " +
Data.SessionNames[sessionIndex], FormToAdd, Properties.Resources.Input)
        {
            InitialiseObjects();
            LoadTrackList();
            SelectRace.SelectedIndex = roundIndex;
            SelectSession.SelectedIndex = sessionIndex;
        }

        /// <summary>
        /// Sets the combo boxes to display the race and session indices
        /// </summary>
        public void ResetPanel(int roundIndex, int sessionIndex)
        {

```

```

        SelectRace.SelectedIndex = roundIndex;
        SelectSession.SelectedIndex = sessionIndex;
    }

    void cbxSelectRace_SelectedIndexChanged(object sender, EventArgs e)
    {
        if (SelectSession.Items.Count == 0)
            LoadSessionList();
        Data.RaceIndex = SelectRace.SelectedIndex;
    }

    void cbxSelectSession_SelectedIndexChanged(object sender, EventArgs e)
    {
        UpdateDataSessionNumber();
        LoadDataTypeComboBox();
    }

    void cbxSelectData_SelectedIndexChanged(object sender, EventArgs e)
    {
        OpenSelectedPDFFile(SelectSession.SelectedIndex, SelectData.SelectedIndex,
SelectRace.Text);
    }

    void btnSelectText_Click(object sender, EventArgs e)
    {
        dataFromClipboard = Clipboard.GetText();
        DataController.ProcessData(dataFromClipboard, fileType);
    }

    void btnAnalyseData_Click(object sender, EventArgs e)
    {
        StartParameterCalculations();
    }

    public void InitialiseObjects()
    {
        SelectRace.Location = new Point(leftBorder, topBorder + 25);
        SelectRace.Size = cbxDefault;
        SelectRace.Text = "Select race";
        SelectRace.FlatStyle = FlatStyle.Flat;
        SelectRace.SelectedIndexChanged += cbxSelectRace_SelectedIndexChanged;
        this.Controls.Add(SelectRace);

        SelectSession.Location = new Point(leftBorder, topBorder * 2 +
defaultHeight + 25);
        SelectSession.Size = cbxDefault;
        SelectSession.Text = "Select session";
        SelectSession.FlatStyle = FlatStyle.Flat;
        SelectSession.SelectedIndexChanged +=
cbxSelectSession_SelectedIndexChanged;
        this.Controls.Add(SelectSession);

        SelectData.Location = new Point(leftBorder, topBorder * 3 + defaultHeight *
2 + 25);
        SelectData.Size = cbxDefault;
        SelectData.Text = "Select type of data";
        SelectData.FlatStyle = FlatStyle.Flat;
        SelectData.SelectedIndexChanged += cbxSelectData_SelectedIndexChanged;
        this.Controls.Add(SelectData);

        SelectText.Location = new Point(leftBorder * 2 + 200, topBorder + 25);
        SelectText.Size = btnDefault;
        SelectText.Text = "Confirm data entry";
        SelectText.FlatStyle = FlatStyle.Flat;
        SelectText.Click += btnSelectText_Click;
        SelectText.Visible = true;
        this.Controls.Add(SelectText);
    }
}

```

```

        AnalyseData.Location = new Point(leftBorder * 2 + 200, topBorder * 2 +
btnDefault.Height + 25);
        AnalyseData.Size = btnDefault;
        AnalyseData.Text = "Analyse Data";
        AnalyseData.FlatStyle = FlatStyle.Flat;
        AnalyseData.Click += btnAnalyseData_Click;
        this.Controls.Add(AnalyseData);
    }

    void LoadTrackList()
    {
        foreach (Track t in Data.Tracks)
        {
            SelectRace.Items.Add(t.name);
        }
    }
    void LoadSessionList()
    {
        for (int sessionIndex = 0; sessionIndex <= 4; sessionIndex++)
        {
            SelectSession.Items.Add(Data.SessionNames[sessionIndex]);
        }
    }
    void UpdateDataSessionNumber()
    {
        Data.SessionIndex = SelectSession.SelectedIndex;
    }
    void LoadDataTypeComboBox()
    {
        SelectData.Items.Clear();

        if (SelectData.Items.Count == 0)
        {
            SelectData.Items.Add("Lap Times");
            if (Data.SessionIndex >= 3)
            {
                SelectData.Items.Add("Speed Trap");
            }
        }

        if (SelectData.Items.Count == 1 && Data.SessionIndex >= 3)
        {
            SelectData.Items.Add("Speed Trap");
        }

        if (SelectData.Items.Count == 1)
            SelectData.SelectedIndex = 0;
    }

    void StartParameterCalculations()
    {
        bool allSessionsLoaded = true;
        string sessionsNotLoaded = "";
        int session = 0;

        //set up error message for sessions not yet loaded
        foreach (bool b in Data.sessionDataLoaded)
        {
            if (b == false)
            {
                allSessionsLoaded = false;
                sessionsNotLoaded += '\n' + " " + (session == 4 ? "Speeds" :
Data.SessionNames[session]);
            }
            session++;
        }
    }
}

```

```

    if (allSessionsLoaded)
    {
        PanelControlEvents.OnLoadPaceParametersFromRace();
        PanelControlEvents.OnShowPaceParameters(Form);
    }
    else
    {
        //start a dialog
        string message = "Data from the following sessions has not been loaded.  

This may cause incorrect results. Continue?";
        message += sessionsNotLoaded;
        string caption = "Data Not Fully Loaded";

        if (Functions.StartDialog(message, caption))
        {
            PanelControlEvents.OnLoadPaceParametersFromRace();
            PanelControlEvents.OnShowPaceParameters(Form);
        }
    }
}

public void OpenSelectedPDFFile(int sessionIndex, int dataType, string
raceName)
{
    string sessionName = Data.SessionNames[sessionIndex];

    string fileName = GetPDFFileName(sessionName, dataType);

    string path = Path.Combine(Data.Settings.TimingDataBaseFolder + raceName);
    string root = (Path.GetPathRoot(path)).Substring(0, 2);
    string openCommand = "Start " + Data.Settings.PDFReaderName + " ";
    string fileToOpen = fileName;

    string command = @"/C " + root + @"&& cd " + path + @"&& " + openCommand +
fileToOpen;

    //Run the command
    RunCommand(command);

    //Set the file type
    fileType = GetTypeOfFile(fileName, sessionIndex);
}

/// <summary>
/// Runs the specified command in the cmd.exe window
/// CAUTION: SECURITY ISSUE IF USED INCORRECTLY
/// </summary>
/// <param name="command">The command to run</param>
void RunCommand(string command)
{
    Process process = new Process();
    ProcessStartInfo startInfo = new ProcessStartInfo();

    //Set up the window
    startInfo.WindowStyle = ProcessWindowStyle.Hidden;
    startInfo.FileName = "cmd.exe";
    startInfo.RedirectStandardInput = true;
    startInfo.UseShellExecute = false;
    startInfo.Arguments = command;

    //Start the process
    process.StartInfo = startInfo;
    process.Start();
}

```

```

TimingDataType GetTypeOfFile(string fullFileName, int sessionIndex)
{
    string shortFileName = "";
    int _fileType = 0;

    //Remove the session name
    int lettersToRemove = 0;
    switch (sessionIndex)
    {
        case 0: lettersToRemove = 23; break;
        case 1: lettersToRemove = 24; break;
        case 2: lettersToRemove = 23; break;
        case 3: lettersToRemove = 19; break;
        case 4: lettersToRemove = 5; break;
    }
    fullFileName = Path.GetFileNameWithoutExtension(fullFileName);
    shortFileName = fullFileName.Remove(0, lettersToRemove);

    //Set the file type based on the remaining letters
    switch (shortFileName)
    {
        case "Lap Times": _fileType = 0; break;
        case "Lap Analysis": _fileType = 0; break;
        case "Speed Trap": _fileType = 1; break;
        default: Program.InfoPanel.WriteData("Error analysing file"); break;
    }

    return (TimingDataType)_fileType;
}

/// <summary>
/// Gets the name of a text file containing the specified data
/// </summary>
/// <param name="raceName"></param>
/// <param name="sessionNumber"></param>
/// <returns>The name of the PDF file to be expected given this combination of
inputs</returns>
string GetNewTextFileName(string raceName, string sessionNumber)
{
    string newFileName = "";

    newFileName = raceName;
    newFileName += sessionNumber;
    if (fileType == 0)
    {
        newFileName += " Lap Times.txt";
    }
    else
    {
        newFileName += " Speed Trap.txt";
    }

    return newFileName;
}
string GetPDFFileName(string session, int dataType)
{
    string fileName = "";

    fileName += session;
    fileName += " ";

    if (session == "Qualifying") { fileName += "session "; }

    switch (dataType)
    {
        case 0:
            if (session != "Race")

```

```

        { fileName += "Lap Times.pdf"; }
        else
        { fileName += "Lap Analysis.pdf"; }
        break;
    case 1: fileName += "Speed Trap.pdf"; break;
}

return fileName;
}
}
}

using StratSim.Model;
using StratSim.View.MyFlowLayout;
using StratSim.View.UserControls;
using StratSim.ViewModel;
using System;
using System.Collections.Generic;
using System.Drawing;
using System.Windows.Forms;

namespace StratSim.View.Panels
{
    /// <summary>
    /// Represents a list of drivers with options to select which drivers are
    /// displayed on a graph
    /// and which driver is selected for detailed analysis
    /// Also allows timing data from the race simulation to be displayed
    /// </summary>
    public class DriverSelectPanel : MyPanel
    {
        NormalisedRadioButton[] radioButtons;
        ShowDriverCheckBox[] checkBoxes;
        CheckBox showAll;
        Label[,] driverLabels;
        Label[,] gapLabels;
        ComboBox lapNumber;

        List<int> selectedDriverIndices = new List<int>();

        const int YSpacing = 23;

        CumulativeTimeGraph Graph;
        int currentLapNumber;

        bool showingTimeGaps;

        public delegate void DriverSelectionChangedEventHandler();
        public event DriverSelectionChangedEventHandler DriverSelectionsChanged;

        public delegate void NormalisedDriverChangedEventHandler();
        public event NormalisedDriverChangedEventHandler NormalisedDriverChanged;

        public DriverSelectPanel(MainForm FormToAdd)
            : base(180, 550, "Drivers", FormToAdd, Properties.Resources.Driver_Select)
        {
            showingTimeGaps = false;
            currentLapNumber = Data.GetRaceLaps();

            InitialiseControls();
            AddControls();

            MyEvents.UpdateIntervals += MyEvents_UpdateIntervals;
        }
    }
}

```

```

        SetPanelProperties(DockTypes.TopRight, AutosizeTypes.AutoHeight,
FillStyles.FullHeight, this.Size);
    }

    public void SetGraph(CumulativeTimeGraph graph)
    {
        Graph = graph;
    }

    void InitialiseControls()
    {
        showAll = new CheckBox();
        driverLabels = new Label[Data.NumberOfDrivers, 2];
        gapLabels = new Label[Data.NumberOfDrivers, 4];
        radioButtons = new NormalisedRadioButton[Data.NumberOfDrivers];
        checkBoxes = new ShowDriverCheckBox[Data.NumberOfDrivers];

        for (int driverIndex = 0; driverIndex < Data.NumberOfDrivers;
driverIndex++)
        {
            driverLabels[driverIndex, 0] = new Label();
            driverLabels[driverIndex, 1] = new Label();
            radioButtons[driverIndex] = new NormalisedRadioButton(driverIndex);
            checkBoxes[driverIndex] = new ShowDriverCheckBox(driverIndex);
        }
    }

    void AddControls()
    {
        //The widths of the columns
        int[] Widths = { 20, 110, 20, 20, 20, 30, 30, 20 };
        int XLocation;
        int YLocation = MyPadding.Top;
        MyToolTip tooltip;

        XLocation = MyPadding.Left;
        XLocation += Widths[0] + Widths[1] + Widths[2];

        showAll = new CheckBox();
        showAll.Location = new Point(XLocation, YLocation);
        showAll.Width = Widths[3];
        showAll.Checked = true;
        showAll.FlatStyle = FlatStyle.Flat;
        tooltip = new MyToolTip(showAll, "Show/Hide all traces");
        showAll.CheckedChanged += showAll_CheckedChanged;
        this.Controls.Add(showAll);

        YLocation += YSpacing;

        for (int driverIndex = 0; driverIndex < Data.NumberOfDrivers;
driverIndex++)
        {
            XLocation = MyPadding.Left;

            driverLabels[driverIndex, 0].Location = new Point(XLocation, YLocation);
            driverLabels[driverIndex, 0].Text =
Convert.ToString(Data.Drivers[driverIndex].DriverNumber);
            driverLabels[driverIndex, 0].Width = Widths[0];
            this.Controls.Add(driverLabels[driverIndex, 0]);
            XLocation += Widths[0];

            driverLabels[driverIndex, 1].Location = new Point(XLocation, YLocation);
            driverLabels[driverIndex, 1].ForeColor =
Data.Drivers[driverIndex].LineColour;
            driverLabels[driverIndex, 1].Text =
Data.Drivers[driverIndex].DriverName;
            driverLabels[driverIndex, 1].Width = Widths[1];
        }
    }
}

```

```

        this.Controls.Add(driverLabels[driverIndex, 1]);
        XLocation += Widths[1];

        radioButtons[driverIndex].Location = new Point(XLocation, YLocation);
        radioButtons[driverIndex].CustomCheckedChanged +=
NormalisedRadioButtonChecked;
        radioButtons[driverIndex].Width = Widths[2];
        radioButtons[driverIndex].FlatStyle = FlatStyle.Flat;
        this.Controls.Add(radioButtons[driverIndex]);
        XLocation += Widths[2];

        checkBoxes[driverIndex].Location = new Point(XLocation, YLocation);
        checkBoxes[driverIndex].CustomButtonChecked += DriverCheckBoxChanged;
        checkBoxes[driverIndex].Width = Widths[3];
        checkBoxes[driverIndex].FlatStyle = FlatStyle.Flat;
        this.Controls.Add(checkBoxes[driverIndex]);
        XLocation += Widths[3];

        if (showingTimeGaps)
        {
            for (int column = 0; column < 4; column++)
            {
                gapLabels[driverIndex, column] = new Label();
                gapLabels[driverIndex, column].Width = Widths[column + 4];
                gapLabels[driverIndex, column].Location = new Point(XLocation,
YLocation + 5);
                XLocation += Widths[column + 4];
                this.Controls.Add(gapLabels[driverIndex, column]);
            }
            tooltip = new MyToolTip(gapLabels[driverIndex, 0], "Position");
            tooltip = new MyToolTip(gapLabels[driverIndex, 1], "Gap");
            tooltip = new MyToolTip(gapLabels[driverIndex, 2], "Interval");
            tooltip = new MyToolTip(gapLabels[driverIndex, 3], "Stops");
        }

        YLocation += YSpacing;
    }

    if (showingTimeGaps) //Show the timing data text boxes
    {
        lapNumber = new ComboBox();
        lapNumber.Location = new Point(MyPadding.Left, YLocation);

        int laps = Data.GetRaceLaps();
        for (int lapIndex = 1; lapIndex <= laps; lapIndex++)
        {
            lapNumber.Items.Add(lapIndex);
        }
        lapNumber.SelectedIndex = laps - 1;
        lapNumber.FlatStyle = FlatStyle.Flat;

        lapNumber.SelectedIndexChanged += lapNumber_SelectedIndexChanged;
        this.Controls.Add(lapNumber);
    }

    void lapNumber_SelectedIndexChanged(object sender, EventArgs e)
    {
        int oldLapNumber = currentLapNumber;
        currentLapNumber = lapNumber.SelectedIndex + 1;
        MyEvents.OnLapNumberChanged(oldLapNumber, currentLapNumber);
    }

    /// <summary>
    /// Redraws all controls on the form and redraws them when
    /// the show time gaps value is changed

```

```

    /// </summary>
    /// <param name="gapLabelsCurrentlyDisplayed">Specifies whether the time gaps
    should be shown or hidden</param>
    void ResetPanel(bool gapLabelsCurrentlyDisplayed)
    {
        showAll.CheckedChanged -= showAll_CheckedChanged;
        this.Controls.Remove(showAll);
        for (int driverIndex = 0; driverIndex < Data.NumberOfDrivers;
driverIndex++)
        {
            this.Controls.Remove(driverLabels[driverIndex, 0]);
            this.Controls.Remove(driverLabels[driverIndex, 1]);
            radioButtons[driverIndex].CustomCheckedChanged -=
NormalisedRadioButtonChecked;
            this.Controls.Remove(radioButtons[driverIndex]);
            checkBoxes[driverIndex].CustomButtonChecked -= DriverCheckBoxChanged;
            this.Controls.Remove(checkBoxes[driverIndex]);
        }

        if (gapLabelsCurrentlyDisplayed)
        {
            for (int driverIndex = 0; driverIndex < Data.NumberOfDrivers;
driverIndex++)
            {
                for (int j = 0; j < 4; j++)
                {
                    this.Controls.Remove(gapLabels[driverIndex, j]);
                }
            }

            lapNumber.SelectedIndexChanged -= lapNumber_SelectedIndexChanged;
            this.Controls.Remove(lapNumber);
            this.Width = 180;
        }
        else
        {
            this.Width = 280;
        }

        SetPanelProperties(DockTypes.TopRight, AutosizeTypes.AutoHeight,
FillStyles.FullHeight, this.Size);

        //Add the controls again
        AddControls();
    }

    void MyEvents_UpdateIntervals(racePosition[,] positions, int lapNumber)
    {
        if (lapNumber == currentLapNumber && showingTimeGaps)
        {
            WriteIntervalData(positions);
        }
    }

    void WriteIntervalData(racePosition[,] positions)
    {
        int positionIndex = -1;
        int lapNumber = currentLapNumber - 1;

        for (int driverIndex = 0; driverIndex < Data.NumberOfDrivers;
driverIndex++)
        {
            positionIndex = -1;
            while (positions[++positionIndex, lapNumber].driver != driverIndex)
            {
            }
            gapLabels[driverIndex, 0].Text =
Convert.ToString(positions[positionIndex, lapNumber].position + 1);
        }
    }
}

```

```

        gapLabels[driverIndex, 1].Text =
Convert.ToString(positions[positionIndex, lapNumber].gap);
        gapLabels[driverIndex, 2].Text =
Convert.ToString(positions[positionIndex, lapNumber].interval);
        gapLabels[driverIndex, 3].Text =
Convert.ToString(Data.Drivers[driverIndex].StopsBeforeLap(lapNumber));

        PositionDriverControls(positionIndex, driverIndex);
    }
}

/// <summary>
/// Repositions a row of driver controls according to their position in the
race
/// </summary>
/// <param name="position">The race position of the driver being
relocated</param>
/// <param name="driverIndex">The index of the driver being relocated</param>
void PositionDriverControls(int position, int driverIndex)
{
    int YPosition = MyPadding.Top + (position + 1) * YSpacing;
    driverLabels[driverIndex, 0].Top = YPosition;
    driverLabels[driverIndex, 1].Top = YPosition;
    checkBoxes[driverIndex].Top = YPosition;
    radioButtons[driverIndex].Top = YPosition;
    gapLabels[driverIndex, 0].Top = YPosition + 5;
    gapLabels[driverIndex, 1].Top = YPosition + 5;
    gapLabels[driverIndex, 2].Top = YPosition + 5;
    gapLabels[driverIndex, 3].Top = YPosition + 5;
}

void showAll_CheckedChanged(object sender, EventArgs e)
{
    if (showAll.Checked)
    {
        UpdateCheckboxes(ShowTracesState.All);
    }
    else
    {
        UpdateCheckboxes(ShowTracesState.None);
    }
    Graph.UpdateGraph(Data.DriverIndex);
}

/// <summary>
/// Re-normalises the graph when a change to the check boxes requires it.
/// </summary>
void RenormaliseGraphFromDriverSelectPanel(int driverToNormalise)
{
    Data.DriverIndex = driverToNormalise;
    if (NormalisedDriverChanged != null)
        NormalisedDriverChanged();
    Graph.ChangeNormalised(driverToNormalise);
}

void DriverCheckBoxChanged(int checkedDriver)
{
    bool showTrace = checkBoxes[checkedDriver].Checked;
    ShowHideTraces(checkedDriver, showTrace);
}

void ShowHideTraces(int driverToShowOrHide, bool showTrace)
{
    bool renormaliseAfterUpdate = (selectedDriverIndices.Count == 0);

    //Handle the effects of show/hide
    if (showTrace) { selectedDriverIndices.Add(driverToShowOrHide); }
}

```

```

        else
        {
            selectedDriverIndices.Remove(driverToShowOrHide);
            showAll.CheckedChanged -= showAll_CheckedChanged;
            showAll.Checked = false;
            showAll.CheckedChanged += showAll_CheckedChanged;
        }

        //If hiding the normalised driver, change the normalised driver
        if (!showTrace && driverToShowOrHide == Data.DriverIndex)
        {
            RenormaliseGraphFromDriverSelectPanel(Graph.GetBestDriver());
        }

        //Fire event to signify that the list of shown drivers has changed
        if (DriverSelectionsChanged != null)
            DriverSelectionsChanged();

        Graph.ShowTraces();

        //If this is the only trace on the graph, re-normalise on the only
        remaining trace.
        if (renormaliseAfterUpdate)
            RenormaliseGraphFromDriverSelectPanel(driverToShowOrHide);
    }

    void NormalisedRadioButtonChecked(int checkedDriver)
    {
        if (checkBoxes[checkedDriver].Checked) //set as the normalised driver
            ChangeNormalisedDriver(checkedDriver);
        else //no changes to be made
            UpdateRadioButtons(Data.DriverIndex);
    }

    void ChangeNormalisedDriver(int checkedDriver)
    {
        Data.DriverIndex = checkedDriver;
        if (NormalisedDriverChanged != null)
            NormalisedDriverChanged();
        Graph.ChangeNormalised(checkedDriver);
    }

    public void UpdateRadioButtons(int normalisedDriverIndex)
    {
        foreach (NormalisedRadioButton r in radioButtons)
        {
            r.CheckedChanged -= r.NormalisedRadioButton_CheckedChanged;
        }

        foreach (NormalisedRadioButton r in radioButtons)
        {
            r.Checked = (r.DriverIndex == normalisedDriverIndex);
        }

        foreach (NormalisedRadioButton r in radioButtons)
        {
            r.CheckedChanged += r.NormalisedRadioButton_CheckedChanged;
        }
    }

    public void UpdateCheckboxes(ShowTracesState displayState)
    {
        //If the show traces state is either show all or show none.
        if (displayState != ShowTracesState.Selected)
        {
            selectedDriverIndices.Clear();
        }
    }
}

```

```

        foreach (ShowDriverCheckBox c in checkBoxes)
        {
            c.CheckedChanged -= c.ShowDriverCheckBox_CheckedChanged;
        }

        foreach (ShowDriverCheckBox c in checkBoxes)
        {
            if (displayState == ShowTracesState.All)
            {
                c.Checked = true;
                selectedDriverIndices.Add(c.DriveIndex);
                showAll.CheckedChanged -= showAll_CheckedChanged;
                showAll.Checked = true;
                showAll.CheckedChanged += showAll_CheckedChanged;
            }
            else
            {
                c.Checked = false;
            }
        }

        foreach (ShowDriverCheckBox c in checkBoxes)
        {
            c.CheckedChanged += c.ShowDriverCheckBox_CheckedChanged;
        }
    }

    public List<int> SelectedDriverIndices
    {
        get { return selectedDriverIndices; }
        set { selectedDriverIndices = value; }
    }

    public bool ShowTimeGaps
    {
        get { return showingTimeGaps; }
        set
        {
            if (showingTimeGaps != value)
            {
                showingTimeGaps = value;
                ResetPanel(!showingTimeGaps);
            }
        }
    }
}

using StratSim.Model;
using StratSim.View.MyFlowLayout;
using System;
using System.Drawing;
using System.Windows.Forms;

namespace StratSim.View.Panels
{
    /// <summary>
    /// Represents a panel which displays information about the current state of the
    /// system.
    /// Includes data about the current driver and track
    /// </summary>
    public class InfoPanel : MyPanel
    {

```

```

    Panel driverInfo;
    Panel trackInfo;
    Panel output;

    TextBox driverName, driverNumber, driverTeam;
    TextBox trackName, trackLaps, trackTimeLoss;
    TextBox outputText;

    bool showDriverInfo;
    bool showTrackInfo;

    public InfoPanel(MainForm FormToAdd)
        : base(150, 300, "Info", FormToAdd, Properties.Resources.Info)
    {
        showDriverInfo = false;
        showTrackInfo = false;

        InitialiseControls();
        AddControls();

        SetPanelProperties(DockTypes.TopLeft, AutosizeTypes.AutoHeight,
FillStyles.None, this.Size);
    }

    /// <summary>
    /// Writes data to the output text box
    /// </summary>
    /// <param name="dataToWrite">The string to be written to the text box.
    /// This should be informative about what the system is currently
processing.</param>
    public void WriteData(string dataToWrite)
    {
        outputText.Text += "\r\n";
        outputText.Text += dataToWrite;
    }

    void InitialiseControls()
    {
        driverInfo = new Panel();
        trackInfo = new Panel();
        output = new Panel();

        driverName = new TextBox();
        driverNumber = new TextBox();
        driverTeam = new TextBox();
        trackName = new TextBox();
        trackLaps = new TextBox();
        trackTimeLoss = new TextBox();
        outputText = new TextBox();
    }

    /// <summary>
    /// Populates and adds the controls to the panel
    /// </summary>
    void AddControls()
    {
        int defaultLeftPosition = this.MyPadding.Left;
        int defaultWidth = this.Width - this.MyPadding.Left - this.MyPadding.Right;
        int PositionY = this.MyPadding.Top;
        //markPositionY gives the top position of the current panel
        int markPositionY = PositionY;

        if (showDriverInfo)
        {
            driverInfo.Top = PositionY;
            markPositionY = PositionY;
            PositionY += controlSpacing;
        }
    }
}

```

```

        driverInfo.Width = defaultWidth;
        driverInfo.Left = defaultLeftPosition;

        driverName.BackColor = Data.CurrentDriver.LineColour;
        if (Data.CurrentDriver.LineColour.GetBrightness() < 0.5) {
driverName.ForeColor = Color.White; }
        else { driverName.ForeColor = Color.Black; }
        driverName.Text = Data.CurrentDriver.DriverName;
        driverName.Location = new Point(defaultLeftPosition, PositionY -
markPositionY);
        PositionY += driverName.Height + controlSpacing;
        driverInfo.Controls.Add(driverName);

        driverNumber.Text = Convert.ToString(Data.CurrentDriver.DriverNumber);
        driverNumber.Location = new Point(defaultLeftPosition, PositionY -
markPositionY);
        PositionY += driverNumber.Height + controlSpacing;
        driverInfo.Controls.Add(driverNumber);

        driverTeam.Text = Data.CurrentDriver.Team;
        driverTeam.Location = new Point(defaultLeftPosition, PositionY -
markPositionY);
        PositionY += driverTeam.Height + controlSpacing;
        driverInfo.Controls.Add(driverTeam);

        driverInfo.Height = PositionY - markPositionY;
        this.Controls.Add(driverInfo);
    }

    markPositionY = PositionY;

    if (showTrackInfo)
    {
        trackInfo.Top = PositionY;
        markPositionY = PositionY;
        PositionY += controlSpacing;

        trackInfo.Width = defaultWidth;
        trackInfo.Left = defaultLeftPosition;

        trackName.Text = Data.CurrentTrack.name;
        trackName.Location = new Point(defaultLeftPosition, PositionY -
markPositionY);
        PositionY += trackName.Height + controlSpacing;
        trackInfo.Controls.Add(trackName);

        trackLaps.Text = Convert.ToString(Data.CurrentTrack.laps) + " laps";
        trackLaps.Location = new Point(defaultLeftPosition, PositionY -
markPositionY);
        PositionY += trackLaps.Height + controlSpacing;
        trackInfo.Controls.Add(trackLaps);

        trackTimeLoss.Text = Convert.ToString(Data.CurrentTrack.pitStopLoss) +
"s pit loss";
        trackTimeLoss.Location = new Point(defaultLeftPosition, PositionY -
markPositionY);
        PositionY += trackTimeLoss.Height + controlSpacing;
        trackInfo.Controls.Add(trackTimeLoss);

        trackInfo.Height = PositionY - markPositionY;
        this.Controls.Add(trackInfo);
    }

    output.Top = PositionY;
    markPositionY = PositionY;
    PositionY += controlSpacing;

```

```

        output.Width = defaultWidth;
        output.Left = defaultLeftPosition;

        outputText.Multiline = true;
        outputText.Height = 120;
        outputText.Location = new Point(defaultLeftPosition, PositionY -
markPositionY);
        PositionY += outputText.Height + controlSpacing;
        output.Controls.Add(outputText);

        outputText.ScrollBars = ScrollBars.Vertical;

        output.Height = PositionY - markPositionY;
        this.Controls.Add(output);
    }

    public bool ShowDriverInfo
    {
        get { return showDriverInfo; }
        set
        {
            showDriverInfo = value;
            AddControls();
        }
    }
    public bool ShowTrackInfo
    {
        get { return showTrackInfo; }
        set
        {
            showTrackInfo = value;
            AddControls();
        }
    }
}

using StratSim.Model;
using StratSim.View.MyFlowLayout;
using StratSim.ViewModel;
using System;
using System.Drawing;
using System.Windows.Forms;

namespace StratSim.View.Panels
{
    /// <summary>
    /// Panel that displays quick shortcuts to the program functions.
    /// Allows for data to be controlled through button clicks.
    /// Provides methods for hiding and showing the controls on the panel depending
    on system state.
    /// </summary>
    public class NewStartPanel : MyPanel
    {
        Label SelectRace;
        ComboBox RaceSelectBox;
        Button Archives;
        Button DataInput;
        Panel PaceParameters;
        Button PPFFromRace;
        Button PPFFromFile;
        Label PaceParametersTitle;
        Button ViewParameters;
        Panel Strategies;
        Button ViewStrategies;
    }
}

```

```

Button SFromData;
Button SFromFile;
Label StrategiesTitle;
Panel Race;
Label RaceTitle;
Button ViewRace;

bool paceParametersLoadedFromFile, strategiesLoadedFromFile;

public NewStartPanel(MainForm Form)
    : base(450, 170, "Start", Form, Properties.Resources.Start)
{
    InitializeComponent();
    LoadTracksComboBox();
    MyEvents.FinishedLoadingParameters += OnPaceParametersFinishedLoading;
    MyEvents.FinishedLoadingStrategies += OnStrategiesFinishedLoading;
}

void LoadTracksComboBox()
{
    foreach (Track t in Data.Tracks)
    {
        RaceSelectBox.Items.Add(t.name);
    }
    RaceSelectBox.SelectedIndexChanged += RaceSelectBox_SelectedIndexChanged;
}

void InitializeComponent()
{
    Designer code for initialising objects on panel removed
}

void OnPaceParametersFinishedLoading()
{
    PaceParameters.Visible = true;
    Strategies.Visible = true;

    PPFromFile.Enabled = !paceParametersLoadedFromFile;
    PPFromRace.Enabled = paceParametersLoadedFromFile;

    ViewParameters.Enabled = true;
    ViewStrategies.Enabled = false;
    Race.Visible = false;
}
void OnStrategiesFinishedLoading()
{
    PaceParameters.Visible = true;
    ViewParameters.Enabled = true;
    ViewStrategies.Enabled = true;

    SFromData.Enabled = strategiesLoadedFromFile;
    SFromFile.Enabled = !strategiesLoadedFromFile;

    Race.Visible = true;
}

void DataInput_Click(object sender, EventArgs e)
{
    PanelControlEvents.OnShowDataInput(Form);
}
void ViewRace_Click(object sender, EventArgs e)
{
    PanelControlEvents.OnStartRaceFromStrategies();
}
void ViewStrategies_Click(object sender, EventArgs e)
{
}

```

```

        PanelControlEvents.OnShowStrategies(Form);
    }
    void ViewParameters_Click(object sender, EventArgs e)
    {
        PanelControlEvents.OnShowPaceParameters(Form);
    }
    void RaceSelectBox_SelectedIndexChanged(object sender, EventArgs e)
    {
        Data.RaceIndex = RaceSelectBox.SelectedIndex;

        PaceParameters.Visible = true;
        ViewParameters.Enabled = false;
        ViewStrategies.Enabled = false;
        Strategies.Visible = false;
        Race.Visible = false;

        PPFromFile.Enabled = true;
        PPFromRace.Enabled = true;
        SFromData.Enabled = true;
        SFromFile.Enabled = true;

        PanelControlEvents.OnRemoveGraphPanels(Form);
    }
    void PPFromRace_Click(object sender, EventArgs e)
    {
        SFromData.Enabled = true;
        SFromFile.Enabled = true;
        paceParametersLoadedFromFile = false;
        PanelControlEvents.OnLoadPaceParametersFromRace();
    }
    void PPFromFile_Click(object sender, EventArgs e)
    {
        SFromData.Enabled = true;
        SFromFile.Enabled = true;
        paceParametersLoadedFromFile = true;
        PanelControlEvents.OnLoadPaceParametersFromFile();
    }
    void SFromFile_Click(object sender, EventArgs e)
    {
        strategiesLoadedFromFile = true;
        ViewStrategies.Enabled = false;
        SFromData.Enabled = true;
        PanelControlEvents.OnLoadStrategiesFromFile();
    }
    void SFromData_Click(object sender, EventArgs e)
    {
        strategiesLoadedFromFile = false;
        ViewStrategies.Enabled = false;
        PanelControlEvents.OnLoadStrategiesFromData();
    }
    void Archives_Click(object sender, EventArgs e)
    {
        PanelControlEvents.OnShowArchives(Form);
    }
}

using StratSim.Model;
using StratSim.View.MyFlowLayout;
using StratSim.View.UserControls;
using StratSim.ViewModel;
using System;
using System.Drawing;
using System.Windows.Forms;

```

```

namespace StratSim.View.Panels
{
    /// <summary>
    /// Represents a panel which displays the driver's pace parameters on a panel.
    /// The data is tabulated and contains formatting and validation information.
    /// </summary>
    public class PaceParameters : MyPanel
    {
        const int leftBorder = 15;
        const int horizontalBorder = 10;
        const int defaultHeight = 15;

        Label[] headingsLabels = new Label[11];
        string[] headingsText = { "Number", "Team", "Name", "Top Speed", "Tyre Delta",
            "Prime Deg", "Option Deg", "Pace", "Fuel Effect", "Fuel Consumption", "P2 fuel" };
        string[] toolTipText = { "Driver Number", "Team Number", "Driver Name", "Top Speed",
            "Difference between tyre speeds",
            "Degradation per lap on prime tyre", "Degradation per lap on option tyre",
            "Qualifying lap time",
            "seconds per kilogram loss due to fuel", "kg per lap fuel consumption",
            "Fuel used for P2 fastest lap time" };
        int[] widths = { 50, 70, 110, 60, 50, 50, 50, 70, 100, 90 };
        ParameterTextBox[,] parameterBoxes = new ParameterTextBox[8,
Data.NumberOfDrivers];
        Label[,] nameBoxes = new Label[3, Data.NumberOfDrivers];

        PaceParameterData paceParameterData;

        ToolStripDropDownButton thisToolStripDropDown;
        ToolStripButton save, load, resetDropDown, updateDropDown;
        ToolStripDropDownButton goToStrategiesDropDown;

        bool initialised;

        /// <summary>
        /// Creates a new instance of the Pace Parameters class.
        /// </summary>
        /// <param name="FormToAdd">The form on which the panel will be added</param>
        /// <param name="loadFromFile">Represents whether the data is loaded from a
file saved in main memory,
        /// or is processed from current timing data</param>
        public PaceParameters(MainForm FormToAdd, bool loadFromFile)
            : base(800, 700, "Parameters", FormToAdd,
Properties.Resources.Pace_Parameters)
        {
            initialised = false;
            paceParameterData = new PaceParameterData(PaceParameters_DataLoaded);
            paceParameterData.InitialiseData(loadFromFile);

            OpenedInNewForm += PaceParameters_OpenedInNewForm;
            PanelClosed += PaceParameters_PanelClosed;

            AutoScroll = true;

            OriginalSize = this.Size;
        }

        void PaceParameters_OpenedInNewForm(MainForm NewForm)
        {
            AddToolStrip();
        }

        public void AddToolStrip()
        {
            MyEvents.OnShowToolStrip(thisToolStripDropDown);
        }
    }
}

```

```

void PositionControls()
{
    int topBorder = MyPadding.Top + 10;
}

void InitialiseControls()
{
    int leftPosition = leftBorder;
    int topBorder = MyPadding.Top + 10;

    Label tempLabel;
    MyToolTip myToolTip;
    for (int column = 0; column <= 10; column++)
    {
        tempLabel = new Label();
        tempLabel.Name = "heading" + column;
        tempLabel.Text = headingsText[column];

        tempLabel.Location = new Point(leftPosition, topBorder);
        leftPosition += widths[column] + leftBorder;
        tempLabel.Size = new Size(widths[column], defaultHeight);
        myToolTip = new MyToolTip(tempLabel, toolTipText[column]);

        headingsLabels[column] = tempLabel;
        this.Controls.Add(headingsLabels[column]);
    }

    Label tempLabel2;
    ParameterTextBox tempTextBox;

    leftPosition = leftBorder;

    for (int columnIndex = 0; columnIndex <= 2; columnIndex++)
    {
        for (int rowIndex = 0; rowIndex < Data.NumberOfDrivers; rowIndex++)
        {
            tempLabel2 = new Label();
            tempLabel2.Location = new Point(leftPosition, topBorder +
(defaultHeight * (rowIndex + 1)) + (horizontalBorder * (rowIndex + 1)));
            tempLabel2.Size = new Size(widths[columnIndex], defaultHeight);
            tempLabel2.Text = paceParameterData.GetData(columnIndex, rowIndex);

            nameBoxes[columnIndex, rowIndex] = tempLabel2;
            this.Controls.Add(nameBoxes[columnIndex, rowIndex]);
        }
        leftPosition += widths[columnIndex] + leftBorder;
    }

    for (int columnIndex = 3; columnIndex <= 10; columnIndex++)
    {
        for (int rowIndex = 0; rowIndex < Data.NumberOfDrivers; rowIndex++)
        {
            if (columnIndex == 5) { tempTextBox = new ParameterTextBox(rowIndex,
columnIndex, paceParameterData.GetPaceData(columnIndex - 3, rowIndex), 0,
paceParameterData.GetPaceData(3, rowIndex)); }
            else
            {
                if (columnIndex == 4) { tempTextBox = new
ParameterTextBox(rowIndex, columnIndex, paceParameterData.GetPaceData(columnIndex -
3, rowIndex), 0, true); }
                else
                {
                    if (columnIndex == 6) { tempTextBox = new
ParameterTextBox(rowIndex, columnIndex, paceParameterData.GetPaceData(columnIndex -
3, rowIndex), paceParameterData.GetPaceData(2, rowIndex), false); }
                }
            }
        }
    }
}

```

```

        {
            tempTextBox = new ParameterTextBox(rowIndex, columnIndex,
paceParameterData.GetPaceData(columnIndex - 3, rowIndex), 0, false);
        }
    }

    tempTextBox.Location = new Point(leftPosition, topBorder +
(defaultHeight * (rowIndex + 1)) + (horizontalBorder * (rowIndex + 1)));
    tempTextBox.Size = new Size(widths[columnIndex], defaultHeight);

    parameterBoxes[columnIndex - 3, rowIndex] = tempTextBox;
    parameterBoxes[columnIndex - 3, rowIndex].OnParameterChanged +=
paceParameterData.UpdateParameter;
    parameterBoxes[columnIndex - 3, rowIndex].OnParameterChanged +=
PaceParameters_OnParameterChanged;

    this.Controls.Add(parameterBoxes[columnIndex - 3, rowIndex]);
}
leftPosition += widths[columnIndex] + leftBorder;
}

thisToolStripDropDown = new ToolStripDropDownButton("Pace Parameters");

save = new ToolStripButton("Save", Properties.Resources.Save);
save.Click += save_Click;
thisToolStripDropDown.DropDownItems.Add(save);

load = new ToolStripButton("Load", Properties.Resources.Open);
load.Click += load_Click;
thisToolStripDropDown.DropDownItems.Add(load);

resetDropDown = new ToolStripButton("Reset", Properties.Resources.Reset);
resetDropDown.Click += resetDropDown_Click;
thisToolStripDropDown.DropDownItems.Add(resetDropDown);

updateDropDown = new ToolStripButton("Update",
Properties.Resources.Update);
updateDropDown.Click += updateDropDown_Click;
thisToolStripDropDown.DropDownItems.Add(updateDropDown);

goToStrategiesDropDown = new ToolStripDropDownButton("Start Strategies",
Properties.Resources.Strategies);
RaceDropDown FromFile = new RaceDropDown(0);
FromFile.ButtonClicked += goToStrategies_Click;
FromFile.Text = "From File";
goToStrategiesDropDown.DropDownItems.Add(FromFile);
RaceDropDown FromData = new RaceDropDown(1);
FromData.ButtonClicked += goToStrategies_Click;
FromData.Text = "From Data";
goToStrategiesDropDown.DropDownItems.Add(FromData);

thisToolStripDropDown.DropDownItems.Add(goToStrategiesDropDown);
thisToolStripDropDown.Width = 200;
goToStrategiesDropDown.Width = 200;
}

void PaceParameters_OnParameterChanged(TextChangedEventArgs e)
{
    if (e.ParameterIndex - 3 == 2)
    {
        parameterBoxes[3, e.DriverIndex].LowerBound =
paceParameterData.GetPaceData(2, e.DriverIndex);
    }
    if (e.ParameterIndex - 3 == 3)
    {
}
}

```

```

        parameterBoxes[2, e.DriverIndex].UpperBound =
paceParameterData.GetPaceData(3, e.DriverIndex);
    }
}

void goToStrategies_Click(int buttonIndex, Type buttonType)
{
    if (buttonIndex == 0)
    {
        PanelControlEvents.OnLoadStrategiesFromFile();
    }
    else
    {
        PanelControlEvents.OnLoadStrategiesFromData();
    }
    PanelControlEvents.OnShowStrategies(Form);
}
void updateDropDown_Click(object sender, EventArgs e)
{
    paceParameterData.SetDriverPaceData();
}
void save_Click(object sender, EventArgs e)
{
    paceParameterData.SaveData();
}
void load_Click(object sender, EventArgs e)
{
    LoadDataFromFile();
}
void resetDropDown_Click(object sender, EventArgs e)
{
    paceParameterData.RefreshData();
    PopulatePaceData();
}
void PaceParameters_DataLoaded()
{
    if (!isInitialised)
    {
        InitialiseControls();
        PositionControls();
        AddToolStrip();
        isInitialised = true;
    }
    PopulatePaceData();
}
void PaceParameters_PanelClosed()
{
    MyEvents.OnRemoveToolStrip(thisToolStripDropDown);
}
public void LoadDataFromFile()
{
    paceParameterData.LoadData();
}

void PopulatePaceData()
{
    float parameter;
    for (int driverIndex = 0; driverIndex < Data.NumberOfDrivers;
driverIndex++)
    {
        for (int dataIndex = 0; dataIndex < 8; dataIndex++)
        {
            parameter = paceParameterData.GetPaceData(dataIndex, driverIndex);
            parameterBoxes[dataIndex, driverIndex].SetProperties(parameter,
paceParameterData.IsDefault(parameter, dataIndex, driverIndex),
paceParameterData.IsAnomaly(parameter, dataIndex));
        }
    }
}

```

```

        }

    public bool IsInitialised
    { get { return isInitialised; } }
}

using StratSim.Model;
using StratSim.View.MyFlowLayout;
using StratSim.ViewModel;
using System;
using System.Drawing;
using System.Globalization;
using System.Windows.Forms;

namespace StratSim.View.Panels
{
    /// <summary>
    /// Represents a panel which displays all of the currently-held settings data.
    /// Provides functionality for updating this settings data
    /// </summary>
    public class SettingsPanel : MyPanel
    {

        Button OK, Cancel;
        TabControl SettingsTabControl;
       TabPage PDFSettings;
        TextBox txtPDFReader;
        Label lblPDFReader;
        tabPage FilePaths;
        TextBox txtTimingDataFolder;
        Label lblTimingDataPath;
        tabPage DriversandTracks;
        TextBox txtDataFolder;
        Label lblDataFolder;
        tabPage PaceDefaults;
        TextBox txtTopSpeed;
        Label lblTopSpeed;
        TextBox txtPrimeDegradation;
        Label lblPrimeTyreDeg;
        TextBox txtOptionDegradation;
        Label lblOptionDeg;
        TextBox txtTyreDelta;
        Label lblDefaultDelta;
        tabPage Overtaking;
        TextBox txtBackmarkerLoss;
        Label lblBackmarkerLoss;
        TextBox txtTimeLoss;
        Label lblTimeLoss;
        TextBox txtSpeedDelta;
        Label lblSpeedDelta;
        TextBox txtPaceDelta;
        Label lblPaceDelta;
        Label lblTimeLosses;
        Label lblRequirements;
        TextBox txtFuelEffect;
        Label lblFuelEffect;
        TextBox txtPace;
        Label lblDefaultPace;
        TextBox txtImprovementPerLap;
        TextBox txtEvolution4;
        TextBox txtEvolution2;
        TextBox txtEvolution3;
    }
}

```

```

    TextBox txtEvolution1;
    TextBox txtEvolution0;
    Label lblTrackImprovement;
    Label lblTrackEvolution;
    TextBox txtFuelLoad;
    Label lblFuelLoad;
    Label lblPitStopLoss;
    TextBox txtTimeGap;
    Label lblFollowTimeGap;

    public SettingsPanel(MainForm FormToAdd)
        : base(600, 250, "Settings", FormToAdd, Properties.Resources.Settings)
    {
        InitialiseComponent();
        LocateComponents();
        PopulateTextboxes();

        FlowLayoutEvents.MainPanelResizeEnd += CustomResize;

        SetPanelProperties(DockTypes.BottomLeft, AutosizeTypes.Constant,
FillStyles.None, this.Size);
    }

    void CustomResize()
    {
        LocateComponents();
    }

    void LocateComponents()
    {
        OK.Location = new Point(Width - 200, Height - 40);
        Cancel.Location = new Point(Width - 100, Height - 40);
    }

    /// <summary>
    /// Saves the current settings data and writes it to files if it is valid
    /// </summary>
    void ConfirmChanges()
    {
        bool incorrectValue = false;
        SaveSettingsData(ref incorrectValue);
        if (!incorrectValue)
        {
            Data.Settings.WriteSettingsData();
        }
        MyEvents.OnSettingsModified();
    }

    /// <summary>
    /// Reverts the settings data to previous values
    /// </summary>
    void RevertChanges()
    {
        Data.Settings.LoadData();
        PopulateTextboxes();
    }

    /// <summary>
    /// Populates the text boxes on the panel with settings data
    /// </summary>
    void PopulateTextboxes()
    {
        txtPaceDelta.Text = Convert.ToString(Data.Settings.RequiredPaceDelta);
        txtSpeedDelta.Text = Convert.ToString(Data.Settings.RequiredSpeedDelta);
        txtTimeLoss.Text = Convert.ToString(Data.Settings.TimeLoss);
        txtBackmarkerLoss.Text = Convert.ToString(Data.Settings.BackmarkerLoss);
        txtTimeGap.Text = Convert.ToString(Data.Settings.TimeGap);
    }

```

```

        txtTyreDelta.Text = Convert.ToString(Data.Settings.DefaultCompoundDelta);
        txtOptionDegradation.Text =
Convert.ToString(Data.Settings.DefaultOptionDegradation);
        txtPrimeDegradation.Text =
Convert.ToString(Data.Settings.DefaultPrimeDegradation);
        txtTopSpeed.Text = Convert.ToString(Data.Settings.DefaultTopSpeed);
        txtFuelEffect.Text = Convert.ToString(Data.Settings.DefaultFuelEffect);
        txtPace.Text = Convert.ToString(Data.Settings.DefaultPace);
        txtFuelLoad.Text = Convert.ToString(Data.Settings.DefaultP2Fuel);
        txtDataFolder.Text = Data.Settings.DataFilePath;
        txtEvolution0.Text = Convert.ToString(Data.Settings.TrackEvolution[0]);
        txtEvolution1.Text = Convert.ToString(Data.Settings.TrackEvolution[1]);
        txtEvolution2.Text = Convert.ToString(Data.Settings.TrackEvolution[2]);
        txtEvolution3.Text = Convert.ToString(Data.Settings.TrackEvolution[3]);
        txtEvolution4.Text = Convert.ToString(Data.Settings.TrackEvolution[4]);
        txtImprovementPerLap.Text =
Convert.ToString(Data.Settings.TrackImprovement);
        txtPDFReader.Text = Data.Settings.PDFReaderName;
        txtTimingDataFolder.Text = Data.Settings.TimingDataBaseFolder;
    }

    /// <summary>
    /// Sets the values of the text boxes to the current settings data.
    /// </summary>
    void SaveSettingsData(ref bool incorrectValue)
    {
        var dp = CultureInfo.InvariantCulture.NumberFormat;
        incorrectValue = false;

        Data.Settings.RequiredPaceDelta =
Functions.ValidateNotEqualTo(float.Parse(txtPaceDelta.Text, dp), 0, "Required Pace
Delta", ref incorrectValue, true);
        Data.Settings.RequiredSpeedDelta =
Functions.ValidateNotEqualTo(float.Parse(txtSpeedDelta.Text, dp), 0, "Required
Speed Delta", ref incorrectValue, true);
        Data.Settings.TimeLoss =
Functions.ValidateGreater Than(float.Parse(txtTimeLoss.Text, dp), 0, "Time Loss",
ref incorrectValue, true);
        Data.Settings.BackmarkerLoss =
Functions.ValidateGreater Than(float.Parse(txtBackmarkerLoss.Text, dp), 0,
"BackMarker Time Loss", ref incorrectValue, true);
        Data.Settings.TimeGap =
Functions.ValidateBetweenExclusive(float.Parse(txtTimeGap.Text, dp), 0, 1, "Time
Gap In Traffic", ref incorrectValue, true);
        Data.Settings.DefaultCompoundDelta =
Functions.ValidateLessThan(float.Parse(txtTyreDelta.Text, dp), 0, "Default Compound
Delta", ref incorrectValue, true);
        Data.Settings.DefaultOptionDegradation =
Functions.ValidateGreater Than(float.Parse(txtOptionDegradation.Text, dp), 0,
"Default Option Degradation", ref incorrectValue, true);
        Data.Settings.DefaultPrimeDegradation =
Functions.ValidateGreater Than(float.Parse(txtPrimeDegradation.Text, dp), 0,
"Default Prime Degradation", ref incorrectValue, true);
        Data.Settings.DefaultTopSpeed =
Functions.ValidateGreater Than(float.Parse(txtTopSpeed.Text, dp), 0, "Default Top
Speed", ref incorrectValue, true);
        Data.Settings.DefaultFuelEffect =
Functions.ValidateBetweenExclusive(float.Parse(txtFuelEffect.Text, dp), 0, 0.1F,
"Default Fuel Effect", ref incorrectValue, true);
        Data.Settings.DefaultPace =
Functions.ValidateGreater Than(float.Parse(txtPace.Text, dp), 0, "Default Pace", ref
incorrectValue, true);
        Data.Settings.DefaultP2Fuel =
Functions.ValidateGreater Than(float.Parse(txtFuelLoad.Text, dp), 0, "P2 Fuel Load",
ref incorrectValue, true);
        Data.Settings.DataFilePath = txtDataFolder.Text;
        Data.Settings.TrackEvolution[0] = float.Parse(txtEvolution0.Text, dp);
    }
}

```

```

        Data.Settings.TrackEvolution[1] = float.Parse(txtEvolution1.Text, dp);
        Data.Settings.TrackEvolution[2] = float.Parse(txtEvolution2.Text, dp);
        Data.Settings.TrackEvolution[3] = float.Parse(txtEvolution3.Text, dp);
        Data.Settings.TrackEvolution[4] = float.Parse(txtEvolution4.Text, dp);
        Data.Settings.TrackImprovement = float.Parse(txtImprovementPerLap.Text,
dp);
        Data.Settings.PDFReaderName = txtPDFReader.Text;
        Data.Settings.TimingDataBaseFolder = @txtTimingDataFolder.Text;
    }

    void InitialiseComponent()
    {
        Designer code for initialising objects on the panel removed
    }

    void OK_Click(object sender, EventArgs e)
    {
        ConfirmChanges();
    }

    void Cancel_Click(object sender, EventArgs e)
    {
        RevertChanges();
    }
}

using StratSim.Model;
using StratSim.View.MyFlowLayout;
using StratSim.View.UserControls;
using StratSim.ViewModel;
using System;
using System.Collections.Generic;
using System.Drawing;
using System.Drawing.Drawing2D;
using System.Windows.Forms;

namespace StratSim.View.Panels
{
    /// <summary>
    /// Structure containing data to fully define:
    ///     <para>- Position</para>
    ///     <para>- Offset</para>
    ///     <para>- Scale</para>
    ///     <para>- Label spacing</para>
    /// of a graph axis
    /// </summary>
    public struct axisParameters
    {
        public int startLocation;
        public int baseOffset;
        public float scaleFactor;
        public int axisLabelSpacing;
    };

    /// <summary>
    /// Represents the way that a graph can be normalised
    /// </summary>
    public enum NormalisationType { OnAverageLap, OnEveryLap };

    /// <summary>
    /// Represents a line graph of cumulative time per unit x-axis.
    /// Allows the graph to be normalised in different ways
    /// </summary>
    public class CumulativeTimeGraph : MyPanel
    {
}

```

```

List<StrategyLine> tracesToDrawOnScreen = new List<StrategyLine>();
List<StrategyLine> originalTraces = new List<StrategyLine>();
List<lapDataPoint>[] coordinatesForLookupOnClick;

axisParameters horizontalAxis;
axisParameters verticalAxis;
NormalisationType graphNormalisationType;

List<Label> horizontalAxisLabels = new List<Label>();
List<Label> verticalAxisLabels = new List<Label>();

int normalisedDriverIndex = 0;
int raceLaps;
int lapNumber;

Rectangle graphArea;

Color clearColour = Color.White;
DriverSelectPanel DriverPanel;

public delegate void GraphTraceClickEventHandler(int driverIndexClicked, int
lapNumberOfClick);
public event GraphTraceClickEventHandler GraphRightClick;

public CumulativeTimeGraph(NormalisationType normalisationType, MainForm
formToLink)
    : base(500, 400, "Graph", formToLink, Properties.Resources.Graph)
{
    SetRaceLaps(Data.GetRaceLaps());
    lapNumber = raceLaps;

    PanelClosed += StrategyGraph_PanelClosed;
    PanelOpened += StrategyGraph_PanelOpened;

    normalisationType = graphNormalisationType;
    SetupAxes();

    SetPanelProperties(DockTypes.Top2, AutosizeTypes.Free, FillStyles.None,
this.Size);
    MyPadding = new Padding(5, 25, 5, 5);
}

void StrategyGraph_PanelOpened()
{
    //Subscribe to events
    MyEvents.AxesModified += MyEvents_AxesModified;
    FlowLayoutEvents.MainPanelResizeStart += MyEvents_MainPanelResizeStart;
    FlowLayoutEvents.MainPanelResizeEnd += MyEvents_MainPanelResizeEnd;
    PanelSelected += StrategyGraph_PanelSelected;
    OpenedInNewForm += StrategyGraph_OpenedInNewForm;
    MyEvents.LapNumberChanged += MyEvents_LapNumberChanged;
    MouseClick += StrategyGraph_MouseClick;

    //Setup axes based on panel size
    SetupAxes();
}

void StrategyGraph_PanelClosed()
{
    //Unsubscribe from events
    MyEvents.AxesModified -= MyEvents_AxesModified;
    FlowLayoutEvents.MainPanelResizeStart -= MyEvents_MainPanelResizeStart;
    FlowLayoutEvents.MainPanelResizeEnd -= MyEvents_MainPanelResizeEnd;
    PanelSelected -= StrategyGraph_PanelSelected;
    OpenedInNewForm -= StrategyGraph_OpenedInNewForm;
    MyEvents.LapNumberChanged -= MyEvents_LapNumberChanged;
    MouseClick -= StrategyGraph_MouseClick;
}

```

```

}

void MyEvents_LapNumberChanged(int oldLapNumber, int newLapNumber)
{
    int displayUpToLap = (newLapNumber + 10 > raceLaps ? raceLaps :
newLapNumber + 10);
    lapNumber = displayUpToLap;
    ScaleAxes(displayUpToLap);
    Invalidate();
}

void StrategyGraph_PanelSelected()
{
    graphArea = SetGraphArea();
}

void StrategyGraph_OpenedInNewForm(MainForm Form)
{
    Rectangle newGraphArea = SetGraphArea();
    ScaleAxes(graphArea, newGraphArea);

    graphArea = newGraphArea;
}

void MyEvents_MainPanelResizeStart()
{
    graphArea = SetGraphArea();
}

void MyEvents_MainPanelResizeEnd()
{
    Rectangle newGraphArea = SetGraphArea();
    ScaleAxes(graphArea, newGraphArea);

    graphArea = newGraphArea;
}

void MyEvents_AxesModified(axisParameters XAxis, axisParameters YAxis,
NormalisationType normalisation, bool UserModified)
{
    if (UserModified)
    {
        SetupAxes(XAxis, YAxis, normalisation);
    }
}

public void UpdateGraph(int normalisedDriverIndex)
{
    tracesToShowOnScreen = GetTraces(originalTraces, normalisedDriverIndex);
    coordinatesForLookupOnClick = PopulateClickCoordinates();
    Invalidate();
}

/// <summary>
/// Draws the graph to the screen
/// </summary>
/// <param name="tracesToShow">A list of StrategyLine traces to show on the
graph</param>
/// <param name="showAllOnGraph">Represents whether all traces should be
shown, or only the traces specified in the driver select panel</param>
/// <param name="changeNormalised">Represents whether the graph should be re-
normalised on the fastest trace</param>
public void DrawGraph(List<StrategyLine> tracesToShow, bool showAllOnGraph,
bool changeNormalised)
{
    int bestDriver;
    if (changeNormalised)

```

```

    {
        bestDriver = GetBestDriver(tracesToShow);
        Data.DriverIndex = bestDriver;

        DriverPanel.UpdateRadioButtons(bestDriver);
        DriverPanel.UpdateCheckboxes(ShowTracesState.All);
    }
    else
    {
        bestDriver = Data.DriverIndex;
    }
    originalTraces = tracesToShow;

    UpdateGraph(bestDriver);
}

/// <summary>
/// Gets a list of normalised strategy lines based on the driver selected for
normalisation
/// </summary>
List<StrategyLine> GetTraces(List<StrategyLine> originalTraces, int
normalisedDriverIndex)
{
    List<StrategyLine> tracesToReturn = new List<StrategyLine>();
    float bestTime;
    if (originalTraces.Count != 0)
    { bestTime = originalTraces.Find(t => t.DriverIndex ==
normalisedDriverIndex).TotalTime; }
    else
    { bestTime = 0; }

    tracesToReturn = NormaliseAllLines(bestTime, raceLaps, originalTraces);

    return tracesToReturn;
}

/// <summary>
/// Draws the graph onto the panel, based on the locally stored traces and
axes
/// </summary>
/// <param name="e">The paint event arguments generated when the Paint event
is fired</param>
protected override void OnPaint(PaintEventArgs e)
{
    e.Graphics.SmoothingMode = SmoothingMode.AntiAlias;
    base.OnPaint(e);
    e.Graphics.Clear(clearColour);
    ClearLabels();
    DrawAxes(e.Graphics, lapNumber);
    DrawLines(e.Graphics, lapNumber);
}

/// <summary>
/// Normalises all of the strategy lines in the list of strategy lines passed
to the method
/// </summary>
/// <param name="bestTime">The time of the fastest driver</param>
/// <param name="laps">The number of laps in the race</param>
/// <param name="originalTraces">The traces to normalise</param>
/// <returns>A list of strategy lines which are normalised using the selected
normalisation method</returns>
List<StrategyLine> NormaliseAllLines(float bestTime, float laps,
List<StrategyLine> originalTraces)
{
    float adjustPerLap = bestTime / laps;
    StrategyLine normalisationTrace = originalTraces.Find(t => t.DriverIndex ==
normalisedDriverIndex);
}

```

```

List<StrategyLine> tracesToReturn = new List<StrategyLine>();

foreach (StrategyLine s in originalTraces)
{
    if (DriverPanel.SelectedDriverIndices.Exists(i => i == s.DriverIndex))
    {
        switch (graphNormalisationType)
        {
            case NormalisationType.OnAverageLap:
                {
                    tracesToReturn.Add(GetNormalisedStrategyLine(s,
adjustPerLap));
                    break;
                }
            case NormalisationType.OnEveryLap:
                {
                    tracesToReturn.Add(GetNormalisedStrategyLine(s,
normalisationTrace));
                    break;
                }
        }
    }
}

return tracesToReturn;
}

/// <summary>
/// Normalises a strategy on an average time
/// </summary>
/// <param name="line">The line to normalise</param>
/// <param name="adjustPerLap">The amount by which to adjust the trace each
lap</param>
/// <returns>The new normalised line</returns>
StrategyLine GetNormalisedStrategyLine(StrategyLine line, float adjustPerLap)
{
    StrategyLine normalisedLine = new StrategyLine(line.DriverIndex);
    lapDataPoint tempPoint;

    float lappedCarAdjustment = 0;
    float actualTime = 0;

    int lapIndex = 0;
    foreach (lapDataPoint p in line.Coordinates)
    {
        //Sets the actual time taken for the strategy to complete,
        //before any adjustments are applied
        if (lapIndex == line.Coordinates.Count - 1)
            actualTime = p.time;

        //set the properties of the new point
        tempPoint.driver = p.driver;
        tempPoint.lap = p.lap;
        tempPoint.time = (p.time - adjustPerLap * p.lap + lappedCarAdjustment);
        tempPoint.draw = p.draw;

        //Deals with cars being lapped
        //Decreases the time if they are too slow, or increases them if
        //they are too fast
        if (tempPoint.time > (adjustPerLap / 2))
        {
            lappedCarAdjustment -= adjustPerLap;
            tempPoint.time -= adjustPerLap;
            tempPoint.draw = false;
        }
        if (tempPoint.time < -(adjustPerLap / 2))

```

```

    {
        lappedCarAdjustment += adjustPerLap;
        tempPoint.time += adjustPerLap;
        tempPoint.draw = false;
    }

    normalisedLine.AddPoint(tempPoint);
    ++lapIndex;
}
//Reset the total time for the strategy to the original time.
normalisedLine.TotalTime = actualTime;

return normalisedLine;
}

/// <summary>
/// Normalises a strategy line on every lap.
/// </summary>
/// <param name="line">The line to normalise</param>
/// <param name="normalisationTrace">The trace used for normalisation</param>
/// <returns>The new normalised line</returns>
StrategyLine GetNormalisedStrategyLine(StrategyLine line, StrategyLine
normalisationTrace)
{
    StrategyLine normalisedLine = new StrategyLine(line.DriverIndex);
    lapDataPoint tempPoint;

    int normalisedDriver = normalisationTrace.DriverIndex;
    float lappedCarCyleLimit;

    float thisLapAdjust = 0;
    float lappedCarAdjustment = 0;
    float actualTime = 0;

    int lapIndex = 0;
    foreach (lapDataPoint p in line.Coordinates)
    {
        if (lapIndex == 0)
        {
            lappedCarCyleLimit = normalisationTrace.Coordinates[lapIndex].time;
        }
        else
        {
            lappedCarCyleLimit = normalisationTrace.Coordinates[lapIndex].time -
normalisationTrace.Coordinates[lapIndex - 1].time;
        }
        thisLapAdjust = normalisationTrace.Coordinates[lapIndex].time;

        if (lapIndex == line.Coordinates.Count - 1)
            actualTime = p.time;

        tempPoint.driver = p.driver;
        tempPoint.lap = p.lap;
        tempPoint.time = p.time - thisLapAdjust + lappedCarAdjustment;
        tempPoint.draw = p.draw;

        if (tempPoint.time > (lappedCarCyleLimit / 2))
        {
            lappedCarAdjustment -= lappedCarCyleLimit;
            tempPoint.time -= lappedCarCyleLimit;
            tempPoint.draw = false;
        }
        if (tempPoint.time < -(lappedCarCyleLimit / 2))
        {
            lappedCarAdjustment += lappedCarCyleLimit;
            tempPoint.time += lappedCarCyleLimit;
            tempPoint.draw = false;
        }
    }
}

```

```

        }

        normalisedLine.AddPoint(tempPoint);
        ++lapIndex;
    }
    normalisedLine.TotalTime = actualTime;

    return normalisedLine;
}

/// <summary>
/// Removes all graph axis labels and clears the list of labels
/// </summary>
void ClearLabels()
{
    foreach (Label l in horizontalAxisLabels)
    {
        this.Controls.Remove(l);
    }
    foreach (Label l in verticalAxisLabels)
    {
        this.Controls.Remove(l);
    }
    horizontalAxisLabels.Clear();
    verticalAxisLabels.Clear();
}

/// <summary>
/// Draws alll of the traces that are to be drawn to the screen
/// </summary>
/// <param name="g">The graphics to use to draw the lines on screen</param>
/// <param name="upToLap">The number of laps to show on the graph</param>
void DrawLines(Graphics g, int upToLap)
{
    foreach (StrategyLine lineToDelete in tracesToDeleteOnScreen)
    {
        lineToDelete.DrawLine(horizontalAxis, verticalAxis, g, upToLap);
    }
}

/// <summary>
/// Gets the driver with the fastest trace out of a list of specified traces
/// </summary>
/// <param name="traces">The traces to search for the best driver</param>
/// <returns>The index of the driver with the fastest trace</returns>
public int GetBestDriver(List<StrategyLine> traces)
{
    float bestTime = 0;
    int driversChecked = 0;
    int driverIndex = 0;

    foreach (StrategyLine s in traces)
    {
        //Most wanted holder loop
        if (DriverPanel.SelectedDriverIndices.Count == 0 ||
DriverPanel.SelectedDriverIndices.Exists(index => index == s.DriverIndex))
        {
            if (driversChecked == 0)
            {
                bestTime = s.TotalTime;
                driverIndex = s.DriverIndex;
            }
            if (s.TotalTime < bestTime)
            {
                bestTime = s.TotalTime;
                driverIndex = s.DriverIndex;
            }
        }
    }
}

```

```

        driversChecked++;
    }

    return driverIndex;
}
/// <summary>
/// Gets the best driver out of the list of traces currently
/// selected to be displayed on screen
/// </summary>
/// <returns>The index of the driver with the fastest trace</returns>
public int GetBestDriver()
{
    return GetBestDriver(tracesToDrawOnScreen);
}

public void StrategyGraph_MouseClick(object sender, MouseEventArgs e)
{
    lapDataPoint clickLocation;
    int driverIndex;
    int lapNumber;

    if (e.Button == MouseButtons.Right)
    {
        //Convert the location of the point to a lapDataPoint
        clickLocation = GetPositionOfPoint(e.Location, horizontalAxis,
verticalAxis);
        if (clickLocation.lap > 0 && clickLocation.lap < Data.GetRaceLaps())
        {
            //Find the clicked driver index
            driverIndex = FindClickedDriver(clickLocation);

            //If the driver index is within range
            if (driverIndex >= 0)
            {
                lapNumber = clickLocation.lap;

                string confirmEvent = "Confirm event for " +
Data.Drivers[driverIndex].DriverName + " on lap " + Convert.ToString(lapNumber);
                bool fireEvent = Functions.StartDialog(confirmEvent, "Confirm");

                if (GraphRightClick != null && fireEvent)
                    GraphRightClick(driverIndex, lapNumber);
            }
        }
    }
}

/// <summary>
/// Converts a panel coordinate location into a lapDataPoint
/// </summary>
/// <param name="point">The point to be converted</param>
/// <param name="HorizontalAxis">The horizontal axis used for
conversion</param>
/// <param name="VerticalAxis">The vertical axis used for conversion</param>
/// <returns>The lapDataPoint closest to the specified point</returns>
public static lapDataPoint GetPositionOfPoint(Point point, axisParameters
HorizontalAxis, axisParameters VerticalAxis)
{
    lapDataPoint pointOnAxes;
    pointOnAxes.driver = -1;
    pointOnAxes.draw = true;

    int x = point.X;
    int y = point.Y;
}

```

```

        pointOnAxes.lap = (int)((x - HorizontalAxis.startLocation) /
HorizontalAxis.scaleFactor) + HorizontalAxis.baseOffset + 1;
        pointOnAxes.time = ((y - VerticalAxis.startLocation) /
VerticalAxis.scaleFactor) + VerticalAxis.baseOffset;

    return pointOnAxes;
}

/// <summary>
/// Sets the axes to default locations proportional to the size of the graph
panel
/// </summary>
public void SetupAxes()
{
    horizontalAxis.startLocation = 50;

    //Get the available graph area
    graphArea = SetGraphArea();

    horizontalAxis.baseOffset = 0;
    horizontalAxis.scaleFactor = (float)(graphArea.Width) / (float)(raceLaps);
    horizontalAxis.axisLabelSpacing = (int)((25 * raceLaps) /
(graphArea.Width));
    verticalAxis.baseOffset = 0;
    verticalAxis.scaleFactor = (float)((graphArea.Height) / 100);
    verticalAxis.startLocation = (graphArea.Height / 2) + 10;
    verticalAxis.axisLabelSpacing = (int)(50 / verticalAxis.scaleFactor);

    MyEvents.OnAxesModified(horizontalAxis, verticalAxis,
graphNormalisationType, false);
}

/// <summary>
/// Sets the axes to the specified values
/// </summary>
void SetupAxes(axisParameters horizontalAxis, axisParameters verticalAxis,
NormalisationType normalisation)
{
    this.horizontalAxis = horizontalAxis;
    this.verticalAxis = verticalAxis;
    graphNormalisationType = normalisation;
    UpdateGraph(normalisedDriverIndex);
}

/// <summary>
/// Scales the axes after the available graph area is changed
/// </summary>
/// <param name="oldSize">The old graph area</param>
/// <param name="newSize">The new graph area</param>
void ScaleAxes(Rectangle oldSize, Rectangle newSize)
{
    float scaleFactorX = (float)newSize.Width / (float)oldSize.Width;
    float scaleFactorY = (float)newSize.Height / (float)oldSize.Height;

    horizontalAxis.axisLabelSpacing = (int)Math.Round(scaleFactorX *
horizontalAxis.axisLabelSpacing);
    if (horizontalAxis.axisLabelSpacing == 0) { horizontalAxis.axisLabelSpacing =
1; }
    verticalAxis.axisLabelSpacing = (int)Math.Round(scaleFactorY *
verticalAxis.axisLabelSpacing);
    if (verticalAxis.axisLabelSpacing == 0) { verticalAxis.axisLabelSpacing =
1; }

    horizontalAxis.scaleFactor *= scaleFactorX;
    verticalAxis.scaleFactor *= scaleFactorY;
}

```

```

        verticalAxis.startLocation = (int)(scaleFactorY *
verticalAxis.startLocation);

        MyEvents.OnAxesModified(horizontalAxis, verticalAxis,
graphNormalisationType, false);
    }
    /// <summary>
    /// Shows a detailed section of the axes around the specified lap
    /// </summary>
    /// <param name="displayUpToLap">The lap to show in detail</param>
    void ScaleAxes(int displayUpToLap)
    {
        int displayFromLap = (displayUpToLap - 15 < 0 ? 0 : displayUpToLap - 15);
        horizontalAxis.axisLabelSpacing = 1;
        horizontalAxis.baseOffset = displayFromLap;
        horizontalAxis.scaleFactor = (float)(graphArea.Width) /
(float)(displayUpToLap - displayFromLap);
        MyEvents.OnAxesModified(horizontalAxis, verticalAxis,
graphNormalisationType, false);
    }

    /// <summary>
    /// Draws the graph axes on the panel
    /// </summary>
    /// <param name="g">The OnPaint method graphics used to draw the axes</param>
    /// <param name="upToLap">The lap up to which the graph axes are to be
drawn</param>
    void DrawAxes(Graphics g, int upToLap)
    {
        Label tempLabel;
        int lapNumber = 0;
        int labelIndex = 0;

        int laps = upToLap;

        var axisPen = new Pen(Color.Black);
        var minorAxisPen = new Pen(Color.LightGray);

        //horizontal axis
        //draw the labels
        for (int labelLocation = horizontalAxis.startLocation; (labelLocation <=
graphArea.Right) && (lapNumber < laps); labelLocation += 1 +
Math.Abs((int)(horizontalAxis.scaleFactor * horizontalAxis.axisLabelSpacing)))
        {
            tempLabel = new Label();
            tempLabel.Location = new Point(labelLocation - 10, graphArea.Bottom);
            lapNumber = (int)Math.Round(((labelLocation -
horizontalAxis.startLocation) / horizontalAxis.scaleFactor) +
horizontalAxis.baseOffset);
            tempLabel.Text = Convert.ToString(lapNumber);
            tempLabel.Width = 20;
            horizontalAxisLabels.Add(tempLabel);
            this.Controls.Add(horizontalAxisLabels[labelIndex++]);

            //draw the vertical dividing lines
            g.DrawLine(minorAxisPen, labelLocation, graphArea.Top + 10,
labelLocation, graphArea.Bottom - 10);
        }

        //draw the major axis
        g.DrawLine(axisPen, graphArea.Right, verticalAxis.startLocation +
(int)(verticalAxis.baseOffset * verticalAxis.scaleFactor),
horizontalAxis.startLocation, verticalAxis.startLocation +
(int)(verticalAxis.baseOffset * verticalAxis.scaleFactor));

        //vertical axis
        //draw the labels
    }
}

```

```

        labelIndex = 0;
        //above the axis
        for (int labelLocation = verticalAxis.startLocation +
(int)(verticalAxis.baseOffset * verticalAxis.scaleFactor); labelLocation <=
graphArea.Bottom - 10; labelLocation += 1 + (int)(verticalAxis.scaleFactor *
verticalAxis.axisLabelSpacing))
        {
            tempLabel = new Label();
            tempLabel.Location = new Point(horizontalAxis.startLocation - 31,
labelLocation - 6);
            tempLabel.Text = Convert.ToString((int)((-labelLocation +
verticalAxis.startLocation) / verticalAxis.scaleFactor) +
verticalAxis.baseOffset));
            tempLabel.Width = 30;
            verticalAxisLabels.Add(tempLabel);
            this.Controls.Add(verticalAxisLabels[labelIndex++]);
        }

        //below the axis
        for (int labelLocation = verticalAxis.startLocation +
(int)(verticalAxis.baseOffset * verticalAxis.scaleFactor); labelLocation >=
graphArea.Top + 10; labelLocation -= 1 + (int)(verticalAxis.scaleFactor *
verticalAxis.axisLabelSpacing))
        {
            tempLabel = new Label();
            tempLabel.Location = new Point(horizontalAxis.startLocation - 31,
labelLocation - 6);
            tempLabel.Text = Convert.ToString((int)((-labelLocation +
verticalAxis.startLocation) / verticalAxis.scaleFactor) +
verticalAxis.baseOffset));
            tempLabel.Width = 30;
            verticalAxisLabels.Add(tempLabel);
            this.Controls.Add(verticalAxisLabels[labelIndex++]);
        }

        //draw the major axis
        g.DrawLine(axisPen, horizontalAxis.startLocation, graphArea.Bottom - 10,
horizontalAxis.startLocation, graphArea.Top + 10);

    }

    /// <summary>
    /// Shows traces on the graph if the driver index is selected for show
    /// </summary>
    public void ShowTraces()
    {
        foreach (StrategyLine l in tracesToDrawOnScreen)
        {
            l.Show = DriverPanel.SelectedDriverIndices.Exists(i => i ==
l.DriverIndex);
        }

        UpdateGraph(normalisedDriverIndex);
    }

    /// <summary>
    /// Gets the area available for traces on the graph
    /// </summary>
    /// <returns>A rectangle representing the available area on the panel for the
graph traces</returns>
    Rectangle SetGraphArea()
    {
        Rectangle Area = new Rectangle();
        Area.Location = new Point(horizontalAxis.startLocation, 35);
        Area.Size = new Size(this.ClientRectangle.Width - 20 -
horizontalAxis.startLocation, this.ClientRectangle.Height - 35 - 20);
    }
}

```

```

        return Area;
    }

    /// <summary>
    /// Changes the trace on which the graph is normalised,
    /// and redraws the graph
    /// </summary>
    /// <param name="SelectedDriver">The new driver to normalise the graph
on</param>
    public void ChangeNormalised(int SelectedDriver)
    {
        Data.DriverIndex = normalisedDriverIndex = SelectedDriver;
        DriverPanel.UpdateCheckboxes(ShowTracesState.Selected);
        DriverPanel.UpdateRadioButtons(normalisedDriverIndex);
        UpdateGraph(SelectedDriver);
    }

    /// <returns>An array of lists of points representing the points on the graph
at which a line exists.
    /// The array index is the lap number; the list index is an arbitrary driver
index, not related to the
    /// actual driver array index</returns>
    List<lapDataPoint>[] PopulateClickCoordinates()
    {
        List<lapDataPoint>[] clickCoordinates = new List<lapDataPoint>[raceLaps];

        for (int lapNumber = 0; lapNumber < raceLaps; lapNumber++)
        {
            clickCoordinates[lapNumber] = new List<lapDataPoint>();
            foreach (StrategyLine driverTrace in tracesToDrawOnScreen)
            {
                if (driverTrace.Show)
                {
                    if (driverTrace.Coordinates[lapNumber].draw) {
clickCoordinates[lapNumber].Add(driverTrace.Coordinates[lapNumber]); }
                }
            }
        }

        return clickCoordinates;
    }

    /// <summary>
    /// Finds the driver index of the nearest driver to the location of a click.
    /// Most wanted holder routine.
    /// </summary>
    /// <param name="clickLocation">The point at which the graph was
clicked</param>
    /// <returns>The index of the driver who is clicked</returns>
    int FindClickedDriver(lapDataPoint clickLocation)
    {
        int mostWantedDriverIndex = -1;
        double closestDistance = 0;
        double currentDistance = 0;

        int driversChecked = 0;

        //for each point on the requested lap
        foreach (lapDataPoint possibleMatch in
coordinatesForLookupOnClick[clickLocation.lap])
        {
            currentDistance = GetDistanceFromClickLocation(possibleMatch,
clickLocation);

            //if the get distance does not return an error
            if (currentDistance > 0)
            {

```

```

        //if this is the first driver checked
        if (driversChecked == 0)
        {
            closestDistance = currentDistance;
            mostWantedDriverIndex = possibleMatch.driver;
        }
        else
        {
            //if the current driver is closer than the previous closest
            if (currentDistance < closestDistance)
            {
                closestDistance = currentDistance;
                mostWantedDriverIndex = possibleMatch.driver;
            }
        }
        driversChecked++;
    }

    return mostWantedDriverIndex;
}

/// <summary></summary>
/// <returns>A value representing the gap between the point being checked and
the point being searched for</returns>
double GetDistanceFromClickLocation(lapDataPoint pointBeingChecked,
lapDataPoint pointToSearchFor)
{
    double distanceFromIntendedLocation = -1;

    if (pointBeingChecked.lap == pointToSearchFor.lap)
    {
        distanceFromIntendedLocation = Math.Pow(pointBeingChecked.time -
pointToSearchFor.time, 2);
    }

    return distanceFromIntendedLocation;
}

/// <summary>
/// Sets the number of laps the graph will display
/// </summary>
/// <param name="value">The number of laps to set the graph to display</param>
public void SetRaceLaps(int value)
{ raceLaps = value; }

public void SetDriverPanel(DriverSelectPanel driverPanel)
{ DriverPanel = driverPanel; }

public List<StrategyLine> Traces
{ get { return tracesToDrawOnScreen; } }
public lapDataPoint[] ClickCoordinates
{ get { return coordinatesForLookupOnClick; } }
}

using StratSim.Model;
using StratSim.View.MyFlowLayout;
using StratSim.View.UserControls;
using StratSim.ViewModel;
using System;
using System.Collections.Generic;
using System.Drawing;
using System.Windows.Forms;

namespace StratSim.View.Panels

```

```
{
    /// <summary>
    /// Represents a panel which displays data about the driver's strategies.
    /// Allows for the data to be edited within the panel.
    /// </summary>
    public class StrategyViewer : MyPanel
    {
        const int leftBorder = 10;
        const int topBorder = 25;
        const int horizontalBorder = 10;
        const int defaultHeight = 15;
        const int defaultWidth = 70;
        const int wideWidth = 100;

        Size txtDefault = new Size(defaultWidth, defaultHeight);
        Size lblDefault = new Size(defaultWidth, defaultHeight);

        List<StintPanel> StintPanels = new List<StintPanel>();
        ResultsPanel Results;
        List<StrategyLine> Traces = new List<StrategyLine>();
        CumulativeTimeGraph graph;
        DriverSelectPanel driverPanel;

        StrategyViewerData strategyViewerData;

        ToolStripDropDownButton thisToolStripDropDown;
        ToolStripButton save, load, resetDropDown, updateDropDown, goToRaceDropDown;

        int raceLaps;
        bool isInitialised;

        public StrategyViewer(MainForm FormToAdd, bool loadFromFile)
            : base(300, 600, "Strategies", FormToAdd, Properties.Resources.Strategies)
        {
            isInitialised = false;
            strategyViewerData = new StrategyViewerData(StrategyViewerData_DataLoaded);
            strategyViewerData.InitailiseData(loadFromFile);

            PanelClosed += StrategyViewer_PanelClosed;
            OpenedInNewForm += StrategyViewer_OpenedInNewForm;
            PanelOpened += StrategyViewer_PanelOpened;
            MyEvents.StrategyModificationsComplete +=
                MyEvents_StrategyModificationsComplete;

            AutoScroll = true;
            raceLaps = Data.GetRaceLaps();
        }

        public void SetGraph(CumulativeTimeGraph Graph)
        {
            graph = Graph;
        }
        public void SetDriverPanel(DriverSelectPanel DriverPanel)
        {
            if (DriverPanel != driverPanel) //Prevents multiple subscribing to events
            {
                driverPanel = DriverPanel;
                driverPanel.NormalisedDriverChanged +=
                    driverPanel_NormalisedDriverChanged;
            }
        }

        void StrategyViewer_OpenedInNewForm(MainForm NewForm)
        {
            AddToolStrip();
        }
    }
}
```

```

public void AddToolStrip()
{
    MyEvents.OnShowToolStrip(thisToolStripDropDown);
}

void graph_PitStopAddedOnClick(int driverIndex, int lapNumber)
{
    Strategy thisStrategy = strategyViewerData.GetStrategy(driverIndex);
    bool stopWithinRange;
    int nearestStop = thisStrategy.GetNearestPitStop(lapNumber, 1, out
stopWithinRange);

    if (stopWithinRange)
    {
        thisStrategy.Stints = thisStrategy.RemovePitStop(nearestStop);
    }
    else
    {
        thisStrategy.Stints = thisStrategy.AddPitStop(lapNumber);
    }

    thisStrategy.UpdateStrategyParameters();
    MyEvents.OnStrategyModified(Data.Drivers[driverIndex], thisStrategy,
false);
}

void driverPanel_NormalisedDriverChanged()
{
    DisplayStints(Data.DriverIndex);
}
/// <summary>
/// When strategy modifications are completed, re-populates controls
/// </summary>
/// <param name="showAllOnGraph">Represents whether all traces should now be
shown on the graph</param>
void MyEvents_StrategyModificationsComplete(bool showAllOnGraph, bool
changeNormalisedDriver)
{
    DrawGraph(showAllOnGraph, changeNormalisedDriver);
    driverPanel.UpdateRadioButtons(Data.DriverIndex);
    DisplayStints(Data.DriverIndex);
}
/// <summary>
/// Once data is loaded, displays and populates the controls on the panel
/// </summary>
void StrategyViewerData_DataLoaded()
{
    MyEvents.OnFinishedLoadingStrategies();
    if (!isInitialised)
    {
        InitialiseControls();
        AddToolStrip();
        isInitialised = true;
    }
    DrawGraph(true, true);
    DisplayStints(Data.DriverIndex);
}

void InitialiseControls()
{
    thisToolStripDropDown = new ToolStripDropDownButton("Strategy Data");

    save = new ToolStripButton("Save", Properties.Resources.Save);
    save.Click += save_Click;
    thisToolStripDropDown.DropDownItems.Add(save);

    load = new ToolStripButton("Load", Properties.Resources.Open);
}

```

```

        load.Click += load_Click;
        thisToolStripDropDown.DropDownItems.Add(load);

        resetDropDown = new ToolStripButton("Reset", Properties.Resources.Reset);
        resetDropDown.Click += resetDropDown_Click;
        thisToolStripDropDown.DropDownItems.Add(resetDropDown);

        updateDropDown = new ToolStripButton("Update",
Properties.Resources.Update);
        updateDropDown.Click += updateDropDown_Click;
        thisToolStripDropDown.DropDownItems.Add(updateDropDown);

        goToRaceDropDown = new ToolStripButton("Start Race",
Properties.Resources.Race);
        goToRaceDropDown.Click += goToRaceDropDown_Click;
        thisToolStripDropDown.DropDownItems.Add(goToRaceDropDown);

        thisToolStripDropDown.DropDown.Width = 200;

        SetGraph(new CumulativeTimeGraph(NormalisationType.OnAverageLap, Form));
        Form.IOController.StartGraph(graph);
    }

    void goToRaceDropDown_Click(object sender, EventArgs e)
    {
        PanelControlEvents.OnStartRaceFromStrategies();
    }
    void updateDropDown_Click(object sender, EventArgs e)
    {
        strategyViewerData.SetDriverStrategies();
    }
    void save_Click(object sender, EventArgs e)
    {
        strategyViewerData.SaveData();
    }
    void load_Click(object sender, EventArgs e)
    {
        strategyViewerData.LoadData();
    }
    void resetDropDown_Click(object sender, EventArgs e)
    {
        strategyViewerData.RefreshData();
    }
    void StrategyViewer_PanelClosed()
    {
        MyEvents.OnRemoveToolStrip(thisToolStripDropDown);

        MyEvents.StrategyModificationsComplete -=
MyEvents_StrategyModificationsComplete;
        strategyViewerData.DataLoaded -= StrategyViewerData_DataLoaded;
        graph.GraphRightClick -= graph_PitStopAddedOnClick;
    }
    void StrategyViewer_PanelOpened()
    {
        MyEvents.StrategyModificationsComplete +=
MyEvents_StrategyModificationsComplete;
        strategyViewerData.DataLoaded += StrategyViewerData_DataLoaded;
        graph.GraphRightClick += graph_PitStopAddedOnClick;
    }

    /// <summary>
    /// Displays the race stints on the panel after the stints have been modified
    /// </summary>
    /// <param name="driverToDisplay">The driver whose strategy is to be
displayed</param>
    void DisplayStints(int driverToDisplay)
    {

```

```

        RemoveStintPanels();
        ShowStintPanels(driverToDisplay);
        graph.UpdateGraph(driverToDisplay);
    }

    /// <summary>
    /// Removes the stint panels from the panel
    /// </summary>
    void RemoveStintPanels()
    {
        foreach (StintPanel stintPanel in StintPanels)
        {
            this.Controls.Remove(stintPanel);
            stintPanel.Dispose();
        }
        StintPanels.Clear();

        this.Controls.Remove(Results);
    }

    /// <summary>
    /// Loads driver strategies and from file
    /// </summary>
    public void LoadDataFromFile()
    {
        strategyViewerData.LoadData();
    }

    /// <summary>
    /// Sets the driver strategies to the currently held data
    /// </summary>
    public void UpdateData()
    {
        strategyViewerData.SetDriverStrategies();
    }

    /// <summary>
    /// Shows the panels and populates with data from the stored strategies
    /// </summary>
    /// <param name="driverToShow">The driver whose data is being
    displayed</param>
    void ShowStintPanels(int driverToShow)
    {
        int stint = 0;
        StintPanel tempPanel;

        Strategy strategyToShow = strategyViewerData.GetStrategy(driverToShow);

        foreach (Stint s in strategyToShow.Stints)
        {
            tempPanel = new StintPanel(stint, Data.Drivers[driverToShow],
strategyToShow);
            tempPanel.Location = new Point(leftBorder, topBorder * (1 + stint) + (80
* stint));
            StintPanels.Add(tempPanel);
            this.Controls.Add(tempPanel);
            stint++;
        }

        Results = new ResultsPanel(driverToShow, strategyToShow);
        Results.Location = new Point(leftBorder, topBorder * (1 +
StintPanels.Count) + (80 * StintPanels.Count));
        this.Controls.Add(Results);
    }

    /// <summary>
    /// Creates the graph traces and calls the graph draw method to display the
    strategies on a graph

```

```

    /// </summary>
    /// <param name="showAllOnGraph">Represents whether all traces on the graph
    will be shown after the update</param>
    /// <param name="changeNormalised">Represents whether the normalised driver
    should be set to the fastest driver,
    /// or maintained as the current driver</param>
    void DrawGraph(bool showAllOnGraph, bool changeNormalised)
    {
        //create traces.
        lapDataPoint tempPoint;
        StrategyLine pointList;
        Strategy thisStrategy;
        int lapsThroughRace = 0;
        float cumulativeTime = 0;

        Traces.Clear();

        for (int driverIndex = 0; driverIndex < Data.NumberOfDrivers;
driverIndex++)
        {
            pointList = new StrategyLine(driverIndex);
            lapsThroughRace = 0;
            cumulativeTime = 0;

            thisStrategy = strategyViewerData.GetStrategy(driverIndex);

            foreach (float lap in thisStrategy.LapTimes)
            {
                cumulativeTime += lap;
                tempPoint.driver = driverIndex;
                tempPoint.lap = lapsThroughRace++;
                tempPoint.time = cumulativeTime;
                tempPoint.draw = true;

                pointList.AddPoint(tempPoint);
            }
            //Set the total time of the line to the final value of cumulative time
            pointList.TotalTime = cumulativeTime;
            Traces.Add(pointList);
        }

        graph.DrawGraph(Traces, showAllOnGraph, changeNormalised);
    }

    public CumulativeTimeGraph GetGraph()
    { return this.graph; }

    public bool IsInitialised
    { get { return isInitialised; } }

}

}

using StratSim.Model;
using StratSim.View.MyFlowLayout;
using System;
using System.Drawing;
using System.Windows.Forms;

namespace StratSim.View.Panels
{
    /// <summary>
    /// Represents a panel displaying archived data from the PDF files
    /// Allows data to be viewed inside the panel
    /// </summary>
    public class TimingArchives : MyPanel

```

```

{
    ComboBox driverCombo;
    Label driverLabel;
    Label raceLabel;
    ComboBox raceCombo;
    ComboBox sessionCombo;
    Label sessionLabel;
    ListBox lapTimes;
    Button getData;

    public TimingArchives(MainForm FormToAdd)
        : base(400, 160, "Archives", FormToAdd, Properties.Resources.Archives)
    {
        InitialiseControls();
        getData.Click += GetData_Click;

        PopulateComboBoxes();
        SetRaceIndex(Data.RaceIndex);
    }

    void GetData_Click(object sender, EventArgs e)
    {
        Data.DriverIndex = driverCombo.SelectedIndex;
        Data.SessionIndex = sessionCombo.SelectedIndex;
        LoadTimingData(driverCombo.SelectedIndex, raceCombo.SelectedIndex,
        sessionCombo.SelectedIndex);
    }

    /// <summary>
    /// Populates the list box with the lap times required
    /// </summary>
    /// <param name="driver">The driver to show</param>
    /// <param name="race">The race from which to display data</param>
    /// <param name="session">The session to display data from</param>
    void LoadTimingData(int driver, int race, int session)
    {
        string lapIdentifier;

        lapTimes.Items.Clear();

        foreach (Stint s in Data.Drivers[driver].RaceWeekendSessionStints[session])
        {
            for (int lapInStint = 0; lapInStint < s.lapTimes.Count; lapInStint++)
            {
                lapIdentifier = Convert.ToString(s.lapTimes[lapInStint]);
                if (lapInStint == 0) { lapIdentifier = "OUT " + lapIdentifier; }
                if (lapInStint == s.lapTimes.Count - 1) { lapIdentifier += " IN"; }
                lapTimes.Items.Add(lapIdentifier);
            }
        }
    }

    /// <summary>
    /// Loads data into the combo boxes
    /// </summary>
    void PopulateComboBoxes()
    {
        foreach (Driver d in Data.Drivers)
        {
            driverCombo.Items.Add(d.DriverName);
        }

        foreach (Track t in Data.Tracks)
        {
            raceCombo.Items.Add(t.name);
        }
    }
}

```

```

        foreach (string s in Data.SessionNames)
    {
        sessionCombo.Items.Add(s);
    }
}

void raceCombo_SelectedIndexChanged(object sender, EventArgs e)
{
    Data.RaceIndex = raceCombo.SelectedIndex;
    Program.PopulateDriverDataFromFiles(raceCombo.SelectedIndex);
}

void InitialiseControls()
{
    driverCombo = new ComboBox();
    driverLabel = new Label();
    raceLabel = new Label();
    raceCombo = new ComboBox();
    sessionCombo = new ComboBox();
    sessionLabel = new Label();
    lapTimes = new ListBox();
    getData = new Button();

    int topBorder = 10;
    int leftBorder = 10;

    Size defaultSize = new Size(100, 20);

    int leftPosition = MyPadding.Left + leftBorder;
    int topPosition = MyPadding.Top + topBorder;

    int col2LeftPosition = leftPosition + defaultSize.Width + leftBorder;
    int col3LeftPosition = col2LeftPosition + defaultSize.Width + leftBorder;

    raceLabel.Location = new Point(leftPosition, topPosition);
    raceLabel.Size = defaultSize;
    raceLabel.Text = "Race";

    raceCombo.Location = new Point(col2LeftPosition, topPosition);
    raceCombo.Size = defaultSize;
    raceCombo.FlatStyle = FlatStyle.Flat;
    raceCombo.SelectedIndexChanged += raceCombo_SelectedIndexChanged;
    topPosition += defaultSize.Height + topBorder;

    driverLabel.Location = new Point(leftPosition, topPosition);
    driverLabel.Size = defaultSize;
    driverLabel.Text = "Driver";

    driverCombo.Location = new Point(col2LeftPosition, topPosition);
    driverCombo.Size = defaultSize;
    driverCombo.FlatStyle = FlatStyle.Flat;
    driverCombo.SelectedIndexChanged += (s, e) => Data.DriverIndex =
driverCombo.SelectedIndex;
    topPosition += defaultSize.Height + topBorder;

    sessionLabel.Location = new Point(leftPosition, topPosition);
    sessionLabel.Size = defaultSize;
    sessionLabel.Text = "Session";

    sessionCombo.Location = new Point(col2LeftPosition, topPosition);
    sessionCombo.Size = defaultSize;
    sessionCombo.FlatStyle = FlatStyle.Flat;
    topPosition += defaultSize.Height + topBorder;

    getData.Location = new Point((leftPosition + (2 * defaultSize.Width +
leftBorder) - getData.Width) / 2, topPosition);
    getData.Size = new Size(100, 25);
}

```

```
    getData.FlatStyle = FlatStyle.Flat;
    getData.Text = "Get Data";

    topPosition = MyPadding.Top + topBorder;
    lapTimes.Location = new Point(col3LeftPosition, topPosition);
    lapTimes.Size = new Size(150, 150);
    lapTimes.BorderStyle = System.Windows.Forms.BorderStyle.None;

    Controls.Add(getData);
    Controls.Add(lapTimes);
    Controls.Add(sessionCombo);
    Controls.Add(sessionLabel);
    Controls.Add(raceCombo);
    Controls.Add(raceLabel);
    Controls.Add(driverLabel);
    Controls.Add(driverCombo);
}

public void SetRaceIndex(int value)
{ raceCombo.SelectedIndex = value; }
}
}
```

StratSim.View.UserControls

```

using StratSim.Model;
using StratSim.View.Panels;
using System;
using System.Collections.Generic;
using System.Drawing;
using System.Windows.Forms;

namespace StratSim.View.UserControls
{
    /// <summary>
    /// Represents options for the status of a 'Show All' check box
    /// </summary>
    public enum ShowTracesState { All, None, Selected };

    /// <summary>
    /// Represents a radio button which fires a custom event containing the index of
    /// the driver
    /// associated with the control.
    /// </summary>
    class NormalisedRadioButton : RadioButton
    {
        int driverIndex;

        /// <summary>
        /// Creates a new instance of a NormalisedRadioButton, associated with the
        /// specified driver index
        /// </summary>
        /// <param name="driverIndex">The driver index associated with the
        /// control</param>
        public NormalisedRadioButton(int driverIndex)
        {
            this.driverIndex = driverIndex;
            CheckedChanged += NormalisedRadioButton_CheckedChanged;
        }

        public void NormalisedRadioButton_CheckedChanged(object sender, EventArgs e)
        {
            if (this.Checked)
                CustomCheckedChanged(driverIndex);
        }

        public int DriverIndex
        {
            get { return driverIndex; }
            set { driverIndex = value; }
        }

        /// <summary>
        /// Event containing data about the driver index associated with the control
        /// </summary>
        /// <param name="checkedDriver">The driver index associated with the pressed
        /// control</param>
        public delegate void CheckedEventHandler(int checkedDriver);
        public event CheckedEventHandler CustomCheckedChanged;
    }

    /// <summary>
    /// Represents a check box that contains an event on click containing data about
    /// the driver index associated with the control
    /// </summary>
    class ShowDriverCheckBox : CheckBox
    {
        int driverIndex;

        /// <summary>

```

```

    /// Creates a new instance of a ShowDriverCheckBox, associated to the
    /// specified driver index
    /// </summary>
    /// <param name="driverIndex">The driver index associated to the
    control</param>
    public ShowDriverCheckBox(int driverIndex)
    {
        this.driverIndex = driverIndex;
        CheckedChanged += ShowDriverCheckBox_CheckedChanged;
    }

    public void ShowDriverCheckBox_CheckedChanged(object sender, EventArgs e)
    {
        CustomButtonChecked(driverIndex);
    }

    public int DriverIndex
    {
        get { return driverIndex; }
        set { driverIndex = value; }
    }

    public delegate void ShowCheckedEventHandler(int checkedDriver);
    public event ShowCheckedEventHandler CustomButtonChecked;
}

}

using System.Windows.Forms;

namespace StratSim.View.UserControls
{
    /// <summary>
    /// Represents a tool tip with a custom delay and formatting
    /// </summary>
    class MyToolTip : ToolTip
    {
        /// <summary>
        /// Adds a tool tip with the specified text to a specified control
        /// </summary>
        /// <param name="control">The control to provide the tool tip for</param>
        /// <param name="text">The text to display in the tool tip</param>
        public MyToolTip(Control control, string text)
        {
            AutoPopDelay = 2500;
            InitialDelay = 250;
            ReshowDelay = 500;
            ShowAlways = true;
            this.BackColor = System.Drawing.Color.White;

            SetToolTip(control, text);
        }
    }
}

using StratSim.Model;
using System;
using System.Drawing;
using System.Windows.Forms;

namespace StratSim.View.UserControls
{
    /// <summary>
    /// Represents a text box that displays a pace parameter datum.

```

```

/// Contains methods for validating and formatting the text box based on the
contents.
/// </summary>
class ParameterTextBox : TextBox
{
    /// <summary>
    /// Enum containing options for the way the control will be validated
    /// </summary>
    internal enum ValidationMethod { LessThan, GreaterThan, Between };

    string parameterValue = "";

    public int DriverIndex, ParameterIndex;
    public float originalValue;
    public bool textChanged = false;

    ValidationMethod validationMethod;

    float upperBound, lowerBound;

    /// <summary>
    /// Creates a new instance of a ParameterTextBox which is validated either
    greater than or less than a specified boundary.
    /// </summary>
    /// <param name="initialValue">The initial value to display in the text
    box</param>
    /// <param name="bound">The limiting value for validation of the input</param>
    /// <param name="boundIsUpperBound">True if the control must be less than the
    specified bound; otherwise false</param>
    public ParameterTextBox(int driverIndex, int parameterIndex, float
    initialValue, float bound, bool boundIsUpperBound)
    {
        if (boundIsUpperBound)
        { upperBound = bound; validationMethod = ValidationMethod.LessThan; }
        else
        { lowerBound = bound; validationMethod = ValidationMethod.GreaterThan; }

        InitialiseComponent(driverIndex, parameterIndex, initialValue);
    }
    /// <summary>
    /// Creates a new instance of a ParameterTextBox which is validated between
    two specified values
    /// </summary>
    public ParameterTextBox(int driverIndex, int parameterIndex, float
    initialValue, float _lowerBound, float _upperBound)
    {
        validationMethod = ValidationMethod.Between;
        lowerBound = _lowerBound;
        upperBound = _upperBound;

        InitialiseComponent(driverIndex, parameterIndex, initialValue);
    }

    void InitialiseComponent(int driverIndex, int parameterIndex, float
    initialValue)
    {
        DriverIndex = driverIndex;
        ParameterIndex = parameterIndex;
        originalValue = initialValue;
        this.BorderStyle = System.Windows.Forms.BorderStyle.None;

        Leave += ParameterTextBox_Leave;
        TextChanged += ParameterTextBox_TextChanged;
    }
}
/// <summary>

```

```

    /// Converts a value to be displayed into a string that can be set as the text
for the control.
    /// </summary>
    public string GetParameterText(float valueToDisplay)
    {
        return Convert.ToString(Math.Round(valueToDisplay, 3));
    }

    /// <summary>
    /// Sets the text box properties including color and text
    /// </summary>
    /// <param name="value">Parameter value to display</param>
    /// <param name="IsDefault">Value representing whether the value is equal to
the default value</param>
    /// <param name="IsAnomaly">Value representing whether the value is an
anomaly</param>
    public void SetProperties(float value, bool IsDefault, bool IsAnomaly)
    {
        if (IsDefault)
        {
            BackColor = Color.Yellow;
        }
        else
        {
            BackColor = Color.Cyan;
            if (!textChanged) { BackColor = Color.White; }
        }

        if (IsAnomaly)
        {
            ForeColor = Color.Red;
        }
        else
        {
            ForeColor = DefaultForeColor;
        }

        parameterValue = Convert.ToString(Math.Round(value, 3));
        Text = parameterValue;
        textChanged = false;
    }

    void ParameterTextBox_TextChanged(object sender, EventArgs e)
    {
        textChanged = true;
    }

    void ParameterTextBox_Leave(object sender, EventArgs e)
    {
        ParameterValue = Text;
    }

    /// <summary>
    /// Validates the value that is intended for the cell
    /// </summary>
    /// <param name="value">The intended cell contents</param>
    /// <returns>True if the value can be accepted</returns>
    bool Validate(float value)
    {
        bool incorrect = false;

        switch (validationMethod)
        {
            case ValidationMethod.LessThan: Functions.ValidateLessThan(value,
upperBound, "", ref incorrect, true); break;
            case ValidationMethod.GreaterThan: Functions.ValidateGreaterThan(value,
lowerBound, "", ref incorrect, true); break;
        }
    }
}

```

```

        case ValidationMethod.Between: Functions.ValidateBetweenExclusive(value,
lowerBound, upperBound, "", ref incorrect, true); break;
    }

    return !incorrect;
}

public float UpperBound
{
    get { return upperBound; }
    set { upperBound = value; }
}

public float LowerBound
{
    get { return lowerBound; }
    set { lowerBound = value; }
}
public string ParameterValue
{
    get { return parameterValue; }
    set
    {
        float parameterNumber;

        bool valid = float.TryParse(value, out parameterNumber);

        if (valid)
        {
            if (Validate(parameterNumber))
            {
                parameterValue = Convert.ToString(Math.Round(parameterNumber, 3));
                var parameterChangeEventArgs = new TextChangedEventArgs(this);
                OnParameterChanged(parameterChangeEventArgs);
            }
            else
            {
                Text = parameterValue;
                Program.InfoPanel.WriteData("Incorrect Value");
            }
        }
        else
        {
            Text = parameterValue;
            Program.InfoPanel.WriteData("Invalid Format");
        }
    }
}

public delegate void textChangedEventHandler(TextChangedEventArgs e);
public event textChangedEventHandler OnParameterChanged;
}

/// <summary>
/// Event arguments containing data about the text box when the text is changed.
/// </summary>
class TextChangedEventArgs : EventArgs
{
    string value;
    int driverIndex;
    int parameterIndex;
    ParameterTextBox textBox;

    public string Value
    { get { return value; } }
    public int DriverIndex
    { get { return driverIndex; } }
}

```

```

public int ParameterIndex
{ get { return parameterIndex; } }
public ParameterTextBox TextBox
{ get { return textBox; } }

/// <summary>
/// Creates new TextChangedEventArgs
/// </summary>
/// <param name="t">The text box that fired the original event</param>
public TextChangedEventArgs(ParameterTextBox t)
{
    value = t.Text;
    driverIndex = t.DriverIndex;
    parameterIndex = t.ParameterIndex;
    textBox = t;
}
}

using StratSim.Model;
using StratSim.ViewModel;
using System;
using System.Drawing;
using System.Windows.Forms;

namespace StratSim.View.UserControls
{
    /// <summary>
    /// Represents a table layout panel linked to a specified driver and strategy,
    /// to be displayed inside a stint panel and linked to a specific stint.
    /// Contains custom formatting and sizing.
    /// </summary>
    class StintButtonsLayoutPanel : TableLayoutPanel
    {
        StintControlButton Up, Down, AddBefore, Remove;
        Size btnDefault = new Size(21, 21);
        Strategy thisStrategy;
        Driver thisDriver;
        int stintIndex;

        public StintButtonsLayoutPanel(Driver driver, int stintIndex, Strategy
strategy)
        {
            thisDriver = driver;
            this.stintIndex = stintIndex;
            thisStrategy = strategy;

            InitialiseButtons();

            this.AutoSize = true;
            this.Size = new Size(42, 42);
            this.RowCount = 2;
            this.ColumnCount = 2;
        }

        void InitialiseButtons()
        {
            MyToolTip tooltip;
            if (stintIndex != 0)
            {
                Up = new StintControlButton(0);
                Up.Size = btnDefault;
                Up.BackgroundImage = Properties.Resources.Stint_Up;
                tooltip = new MyToolTip(Up, "Swap with previous stint");
                Up.OnButtonClicked += ButtonClicked;
            }
        }
    }
}

```

```

        this.Controls.Add(Up, 0, 0);
    }

    if (stintIndex != thisStrategy.NoOfStints - 1)
    {
        Down = new StintControlButton(1);
        Down.Size = btnDefault;
        Down.BackgroundImage = Properties.Resources.Stint_Down;
        tooltip = new MyToolTip(Down, "Swap with next stint");
        Down.OnButtonClicked += ButtonClicked;
        this.Controls.Add(Down, 0, 1);
    }

    AddBefore = new StintControlButton(2);
    AddBefore.Size = btnDefault;
    AddBefore.BackgroundImage = Properties.Resources.Stint_Add;
    tooltip = new MyToolTip(AddBefore, "Add a new stint before this one");
    AddBefore.OnButtonClicked += ButtonClicked;
    this.Controls.Add(AddBefore, 1, 0);

    if (thisStrategy.NoOfStints > 2)
    {
        Remove = new StintControlButton(3);
        Remove.Size = btnDefault;
        Remove.BackgroundImage = Properties.Resources.Stint_Remove;
        tooltip = new MyToolTip(Remove, "Remove this stint from the strategy");
        Remove.OnButtonClicked += ButtonClicked;
        this.Controls.Add(Remove, 1, 1);
    }
}

/// <summary>
/// Handles the button clicked events on any of the buttons within the
control.
/// </summary>
/// <param name="controlNumber">The control number that fired the
event</param>
void ButtonClicked(int controlNumber)
{
    int startLapNumber = thisStrategy.Stints[stintIndex].startLap;
    int midLapNumber = (int)((thisStrategy.Stints[stintIndex].stintLength) / 2)
+ startLapNumber;

    //Performs the required action on the strategy:
    switch (controlNumber)
    {
        case 0: thisStrategy.SwapStints(stintIndex - 1, stintIndex); break;
//moves stint up
        case 1: thisStrategy.SwapStints(stintIndex, stintIndex + 1); break;
//moves stint down
        case 2: thisStrategy.Stints = thisStrategy.AddPitStop(midLapNumber);
break; //splits stint
        case 3: thisStrategy.Stints =
thisStrategy.RemovePitStop(startLapNumber); break; //merges stint with previous
    }

    //Updates the strategy's parameters
    thisStrategy.UpdateStrategyParameters();
    MyEvents.OnStrategyModified(thisDriver, thisStrategy, false);
}

public Driver Driver
{
    get { return thisDriver; }
    set { thisDriver = value; }
}

```

```

    public int StintIndex
    {
        get { return stintIndex; }
        set { stintIndex = value; }
    }

/// <summary>
/// Represents a button containing a custom event, which passes data
/// about the button when it is pressed.
/// </summary>
class StintControlButton : Button
{
    int controlNumber;

    public delegate void StintButtonClickEventHandler(int controlNumber);
    public event StintButtonClickEventHandler OnButtonClicked;

    public StintControlButton(int controlNumber)
    {
        this.controlNumber = controlNumber;
        this.Click += StintControlButton_OnClick;
        this.FlatStyle = System.Windows.Forms.FlatStyle.Flat;
        this.Padding = new Padding(0);
        this.BackColor = Color.White;
    }

    void StintControlButton_OnClick(object sender, EventArgs e)
    {
        //Fire the custom event containing data about the control
        OnButtonClicked(controlNumber);
    }

    public int ControlNumber
    { get { return controlNumber; } }
}

using StratSim.Model;
using StratSim.ViewModel;
using System;
using System.Drawing;
using System.Windows.Forms;

namespace StratSim.View.UserControls
{
    /// <summary>
    /// Represents a panel which displays data about a specified strategy.
    /// Contains methods for modifying the strategy if necessary.
    /// </summary>
    class StintPanel : Panel
    {
        const int leftBorder = 15;
        const int topBorder = 25;
        const int horizontalBorder = 10;
        const int defaultHeight = 20;
        const int defaultWidth = 70;
        const int wideWidth = 100;

        Size txtDefault = new Size(defaultWidth, defaultHeight);
        Size lblDefault = new Size(defaultWidth, defaultHeight);

        StintButtonsLayoutPanel Buttons;
        TextBox stintLength;
        ComboBox tyreType;
        Label stintLabel, length, time, tyre, stintTime;
    }
}

```

```

Strategy thisStrategy;
Driver thisDriver;

int originalLength;
int upperBound;

private int stintIndex;

public StintPanel(int stint, Driver driver, Strategy strategy)
{
    this.Size = new Size(280, 100);
    stintIndex = stint;
    thisDriver = driver;
    thisStrategy = (Strategy)strategy.Clone();
    AddControls();
}

void AddControls()
{
    MyToolTip toolTip;

    stintLabel = new Label();
    stintLabel.Location = new Point(10, 10);
    stintLabel.Size = lblDefault;
    stintLabel.Text = "Stint " + (this.StintNumber + 1);
    this.Controls.Add(stintLabel);

    //Start the stint buttons layout panel.
    Buttons = new StintButtonsLayoutPanel(thisDriver, stintIndex,
thisStrategy);
    Buttons.Location = new Point(10, 40);
    this.Controls.Add(Buttons);

    length = new Label();
    length.Location = new Point(100, 10);
    length.Size = lblDefault;
    length.Text = "Length";
    this.Controls.Add(length);

    time = new Label();
    time.Location = new Point(100, 35);
    time.Size = lblDefault;
    time.Text = "Time";
    this.Controls.Add(time);

    tyre = new Label();
    tyre.Location = new Point(100, 60);
    tyre.Size = lblDefault;
    tyre.Text = "Tyre";
    this.Controls.Add(tyre);

    //calculate the length and upper bounds.
    originalLength = thisStrategy.Stints[stintIndex].stintLength;

    //upper bound depends on either previous or next stint
    if (stintIndex == thisStrategy.NoOfStints - 1) //if stint is last in
strategy
    {
        upperBound = originalLength + thisStrategy.Stints[stintIndex -
1].stintLength;
    }
    else
    {
        upperBound = originalLength + thisStrategy.Stints[stintIndex +
1].stintLength;
    }
}

```

```

        stintLength = new TextBox();
        stintLength.Size = txtDefault;
        stintLength.Location = new Point(200, 10);
        stintLength.Text = Convert.ToString(originalLength);
        stintLength.BorderStyle = System.Windows.Forms.BorderStyle.None;
        toolTip = new MyToolTip(stintLength, "The laps in this stint");
        stintLength.LostFocus += stintLength_LostFocus;
        this.Controls.Add(stintLength);

        stintTime = new Label();
        stintTime.Size = lblDefault;
        stintTime.Location = new Point(200, 35);
        stintTime.Text =
Convert.ToString(thisStrategy.Stints[stintIndex].TotalTime());
        stintTime.BorderStyle = System.Windows.Forms.BorderStyle.None;
        toolTip = new MyToolTip(stintTime, "The total time for this stint");
        this.Controls.Add(stintTime);

        tyreType = new ComboBox();
        foreach (var type in (TyreType[])Enum.GetValues(typeof(TyreType)))
        {
            tyreType.Items.Add(Convert.ToString(type));
        }
        tyreType.Size = txtDefault;
        tyreType.Location = new Point(200, 60);
        tyreType.SelectedIndex = (int)thisStrategy.Stints[stintIndex].tyreType;
        tyreType.FlatStyle = FlatStyle.Flat;
        tyreType.DropDownStyle = ComboBoxStyle.DropDownList;
        tyreType.SelectedIndexChanged += tyreType_SelectedIndexChanged;
        toolTip = new MyToolTip(tyreType, "The tyre type for this stint");
        this.Controls.Add(tyreType);
    }

    /// <summary>
    /// Occurs when the user moves away from the text box containing the stint
length
    /// </summary>
    void stintLength_LostFocus(object sender, EventArgs e)
    {
        int newStintLaps;

        //Validates the input
        bool incorrectValue = !int.TryParse(stintLength.Text, out newStintLaps);

        if (!incorrectValue)
        {
            newStintLaps = (int)Functions.ValidateBetweenExclusive(newStintLaps, 0,
upperBound, "stint length", ref incorrectValue, true);
        }
        if (!incorrectValue)
        {
            thisStrategy.Stints = thisStrategy.ChangeStintLength(stintIndex,
newStintLaps);
        }

        //If something has changed, update the stint and the strategy.
        if (newStintLaps != originalLength)
        {
            originalLength = newStintLaps;
            stintLength.LostFocus -= stintLength_LostFocus;
            thisStrategy.UpdateStrategyParameters();
            MyEvents.OnStrategyModified(thisDriver, thisStrategy, false);
        }
    }

    /// <summary>

```

```

/// Occurs when the user changes the tyre type selected for this stint.
/// </summary>
void tyreType_SelectedIndexChanged(object sender, EventArgs e)
{
    //Changes the tyre type of the stint.
    thisStrategy.ChangeStintTyreType(stintIndex,
(TyreType)tyreType.SelectedIndex);

    tyreType.SelectedIndexChanged -= tyreType_SelectedIndexChanged;
}

public Driver Driver
{
    get{return thisDriver;}
    set{thisDriver = value;}
}
public int StintNumber
{
    get{return stintIndex;}
    set{stintIndex = value;}
}

/// <summary>
/// Represents a panel that contains summary information about a strategy.
/// </summary>
class ResultsPanel : FlowLayoutPanel
{
    Strategy strategy;
    int driverIndex;

    const int leftBorder = 15;
    const int topBorder = 25;
    const int horizontalBorder = 10;
    const int defaultHeight = 20;
    const int defaultWidth = 70;
    const int wideWidth = 100;

    Size txtDefault = new Size(defaultWidth, defaultHeight);
    Size lblDefault = new Size(defaultWidth - 10, defaultHeight + 5);
    Size lblWide = new Size(wideWidth, defaultHeight + 5);

    Label totalTime, averageLap, pitStops, fastestLapLabel, fuelRequired;
    TextBox time, a_lap, stops, f_lap, fuel;

    /// <summary>
    /// Creates a new instance of a ResultsPanel, linked to the specified driver
    and strategy.
    /// </summary>
    public ResultsPanel(int driverIndex, Strategy thisStrategy)
    {
        this.AutoSize = true;
        this.MaximumSize = new Size(400, 150);
        this.FlowDirection = FlowDirection.TopDown;
        this.Margin = new System.Windows.Forms.Padding(10, 10, 5, 5);
        this.driverIndex = driverIndex;
        this.strategy = thisStrategy;
        AddControls();
    }

    void AddControls()
    {
        totalTime = new Label();
        totalTime.Size = lblWide;
        totalTime.Text = "Total Time:";
        this.Controls.Add(totalTime);
    }
}

```

```

        averageLap = new Label();
        averageLap.Size = lblWide;
        averageLap.Text = "Average Lap:";
        this.Controls.Add(averageLap);

        fastestLapLabel = new Label();
        fastestLapLabel.Size = lblWide;
        fastestLapLabel.Text = "Fastest Lap:";
        this.Controls.Add(fastestLapLabel);

        pitStops = new Label();
        pitStops.Size = lblWide;
        pitStops.Text = "Pit Stops:";
        this.Controls.Add(pitStops);

        fuelRequired = new Label();
        fuelRequired.Size = lblWide;
        fuelRequired.Text = "Fuel Required:";
        this.Controls.Add(fuelRequired);

        time = new TextBox();
        time.Size = txtDefault;
        time.Text = Convert.ToString(strategy.TotalTime);
        time.BorderStyle = System.Windows.Forms.BorderStyle.None;
        this.Controls.Add(time);

        a_lap = new TextBox();
        a_lap.Size = txtDefault;
        a_lap.BorderStyle = System.Windows.Forms.BorderStyle.None;
        a_lap.Text = Convert.ToString(strategy.TotalTime /
strategy.LapTimes.Length);
        this.Controls.Add(a_lap);

        float fastestLap = Data.Settings.DefaultPace;
        foreach (Stint s in strategy.Stints)
        {
            if (s.FastestLap() < fastestLap)
                fastestLap = s.FastestLap();
        }

        f_lap = new TextBox();
        f_lap.Size = txtDefault;
        f_lap.Text = Convert.ToString(fastestLap);
        f_lap.BorderStyle = System.Windows.Forms.BorderStyle.None;
        this.Controls.Add(f_lap);

        string pitStopList = "";
        foreach (int pit in strategy.PitStops)
        {
            pitStopList += Convert.ToString(pit);
            pitStopList += ", ";
        }
        pitStopList.TrimEnd(',', ' ');

        stops = new TextBox();
        stops.Size = txtDefault;
        stops.Text = pitStopList;
        stops.BorderStyle = System.Windows.Forms.BorderStyle.None;
        this.Controls.Add(stops);

        fuel = new TextBox();
        fuel.Size = txtDefault;
        fuel.Text = Convert.ToString(strategy.FuelUsed);
        fuel.BorderStyle = System.Windows.Forms.BorderStyle.None;
        this.Controls.Add(fuel);
    }
}

```

```

        public int DriverIndex
    {
        get{return driverIndex;}
        set{driverIndex = value;}
    }
}

using StratSim.Model;
using StratSim.View.Panels;
using System.Collections.Generic;
using System.Drawing;

namespace StratSim.View.UserControls
{
    /// <summary>
    /// Represents a point to be drawn on a cumulative time graph
    /// </summary>
    public struct lapDataPoint
    {
        public int lap;
        public float time;
        public int driver;
        public bool draw;
    };

    /// <summary>
    /// Contains a collection of points that, when linked, represents a trace
    /// on a cumulative time graph.
    /// </summary>
    public class StrategyLine
    {
        List<lapDataPoint> points;
        int driverIndex;
        float sumOfTime = 0;
        bool show = true;

        public StrategyLine(List<lapDataPoint> Points, int driverIndex)
        {
            points = Points;
            this.driverIndex = driverIndex;
        }

        public StrategyLine(int driverIndex)
        {
            this.driverIndex = driverIndex;
            points = new List<lapDataPoint>();
        }

        /// <summary>
        /// Draws a the line represented by the list of points using the specified
graphics
        /// </summary>
        /// <param name="horizontalAxis">The horizontal axis parameters</param>
        /// <param name="verticalAxis">The vertical axis parameters</param>
        /// <param name="g">The graphics to use to draw the line</param>
        /// <param name="upToLap">The last lap to display</param>
        public void DrawLine(axisParameters horizontalAxis, axisParameters
verticalAxis, Graphics g, int upToLap)
        {
            //If the line is to be shown
            if (show)
            {
                Point startPoint = new Point();
                Point endPoint = new Point();
                Color lineColour = Data.Drivers[driverIndex].LineColour;
            }
        }
    }
}

```

```

    Pen pen = new Pen(lineColour, 1);

    //Lap number is 0-based
    endPoint.X = GetPointOrdinate(points[0].lap + 1, horizontalAxis);
    endPoint.Y = GetPointOrdinate(points[0].time, verticalAxis);

    //Cycle through the points
    for (int pointIndex = 1; pointIndex < points.Count; pointIndex++)
    {
        //Set the new start point to the old end point
        startPoint = endPoint;

        //Update the end point
        endPoint.X = GetPointOrdinate(points[pointIndex].lap + 1,
horizontalAxis);
        endPoint.Y = GetPointOrdinate(points[pointIndex].time, verticalAxis);

        //If the point is within range and is to be shown
        if ((startPoint.X > horizontalAxis.startLocation) &&
(points[pointIndex].draw == true) && (points[pointIndex].lap <= upToLap))
            g.DrawLine(pen, startPoint, endPoint);
    }
    pen.Dispose();
}

/// <summary>
/// Gets the ordinate on the specified axis that is represented by the
locating value
/// </summary>
/// <param name="locator">The ordinate that is to be displayed</param>
/// <param name="axis">The axis on which the ordinate is required</param>
/// <returns>The value of the ordinate that the point is to be drawn
at</returns>
int GetPointOrdinate(float locator, axisParameters axis)
{
    float locatorToRepresent = locator - axis.baseOffset;
    int position = (int)(axis.startLocation + (axis.scaleFactor *
locatorToRepresent));

    return position;
}

/// <summary>
/// Adds a point to the end of the list of points to be displayed
/// </summary>
/// <param name="pointToAdd">The point to add to the list</param>
public void AddPoint(lapDataPoint pointToAdd)
{
    points.Add(pointToAdd);
}

public List<lapDataPoint> Coordinates
{ get { return points; } }

public int DriverIndex
{
    get { return driverIndex; }
    set { driverIndex = value; }
}
/// <summary>
/// Gets or sets a value representing the total required time for the
specified driver to complete the race.
/// </summary>
public float TotalTime
{
    get { return sumOfTime; }
    set { sumOfTime = value; }
}

```

```
public bool Show
{
    get { return show; }
    set { show = value; }
}
```

StratSim.ViewModel

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace StratSim.ViewModel
{
    /// <summary>
    /// Interface for any class that controls data input and output.
    /// </summary>
    /// <typeparam name="T">The type of data that is controlled by the
    class</typeparam>
    public interface IFileController<T>
    {
        void ReadFromFile(string fileName);
        void WriteToFile(string fileName);
        T FileData { get; }
    }
}

using StratSim.View.Panels;
using System.Windows.Forms;

namespace StratSim.ViewModel
{
    /// <summary>
    /// Class containing generic events for the program
    /// </summary>
    class MyEvents
    {
        public MyEvents()
        {

        }

        /// <summary>
        /// Fires the RemoveContentPanel event, removing the specified panel from the
        content tab control
        /// </summary>
        /// <param name="panelIndex">The index of the panel to be removed.</param>
        public static void OnRemoveContentPanel(int panelIndex)
        {
            if (RemoveContentPanel != null)
                RemoveContentPanel(panelIndex);
        }

        public delegate void RemoveContentPanelEventHandler(int PanelIndex);
        public static event RemoveContentPanelEventHandler RemoveContentPanel;

        /// <summary>
        /// Fires the SettingsModified event
        /// </summary>
        public static void OnSettingsModified()
        {
            if (SettingsModified != null)
                SettingsModified();
        }

        public delegate void SettingsModifiedEventHandler();
        public static event SettingsModifiedEventHandler SettingsModified;

        /// <summary>
        /// Fires the FinishedLoadingParameters event
        /// </summary>

```

```

public static void OnFinishedLoadingParameters()
{
    if (FinishedLoadingParameters != null)
        FinishedLoadingParameters();
}
/// <summary>
/// Fires the FinishedLoadingStrategies event
/// </summary>
public static void OnFinishedLoadingStrategies()
{
    if (FinishedLoadingStrategies != null)
        FinishedLoadingStrategies();
}

public delegate void FinishedLoadingEventHandler();
public static FinishedLoadingEventHandler FinishedLoadingParameters;
public static FinishedLoadingEventHandler FinishedLoadingStrategies;

/// <summary>
/// Fires the UpdateIntervals event
/// </summary>
/// <param name="positions">The array of race positions as generated by the
race</param>
/// <param name="lapNumber">The lap on which the intervals are being updated.
Lap number is zero based.</param>
public static void OnUpdateIntervals(Model.racePosition[,] positions, int
lapNumber)
{
    UpdateIntervals(positions, lapNumber + 1);
}

public delegate void UpdateIntervalDataEventHandler(Model.racePosition[,]
positions, int lapNumber);
public static event UpdateIntervalDataEventHandler UpdateIntervals;

/// <summary>
/// Fires the LapNumberChanged event, causing the graph to zoom in on a
particular width
/// </summary>
/// <param name="oldLapNumber">The original lap number</param>
/// <param name="newLapNumber">The new lap number</param>
public static void OnLapNumberChanged(int oldLapNumber, int newLapNumber)
{
    LapNumberChanged(oldLapNumber, newLapNumber);
    UpdateIntervals(Model.Data.Race.Positions, newLapNumber);
}

public delegate void LapNumberChangedEventHandler(int oldLapNumber, int
newLapNumber);
public static event LapNumberChangedEventHandler LapNumberChanged;

/// <summary>
/// Fires the ShowToolStrip event, causing the specified tool strip to be
shown on the forms.
/// </summary>
public static void OnShowToolStrip(ToolStripDropDownItem DropDownItem)
{
    ShowToolStrip(DropDownItem);
}

/// <summary>
/// Fires the RemoveToolStrip event, causing the specified tool strip to be
removed from the forms.
/// </summary>
public static void OnRemoveToolStrip(ToolStripDropDownItem DropDownItem)
{
    RemoveToolStrip(DropDownItem);
}

```

```

public delegate void ToolStripEventHandler(ToolStripDropDownItem cm);
public static event ToolStripEventHandler ShowToolStrip;
public static event ToolStripEventHandler RemoveToolStrip;

/// <summary>
/// Fires the strategy modified event after a strategy has been modified
/// </summary>
public static void OnStrategyModified(Model.Driver driver, Model.Strategy
strategy, bool showAllOnGraph)
{
    if (StrategyModified != null)
        StrategyModified(driver.DriverIndex, strategy, showAllOnGraph);
}

/// <summary>
/// Fires the strategy modifications complete event when all changes to a
strategy have been implemented
/// </summary>
public static void OnStrategyModificationsComplete(bool showAllOnGraph, bool
changeNormalisedDriver)
{
    if (StrategyModificationsComplete != null)
        StrategyModificationsComplete(showAllOnGraph, changeNormalisedDriver);
}

public delegate void StrategyModificationsCompleteEventHandler(bool
showAllOnGraph, bool changeNormalisedDriver);
public static event StrategyModificationsCompleteEventHandler
StrategyModificationsComplete;

public delegate void StrategyModifiedEventHandler(int driverIndex,
Model.Strategy strategy, bool showAllOnGraph);
public static event StrategyModifiedEventHandler StrategyModified;

/// <summary>
/// Fires the AxesModified event
/// </summary>
/// <param name="horizontalAxis">The new horizontal axis</param>
/// <param name="verticalAxis">The new vertical axis</param>
/// <param name="normalisation">The new graph normalisation type</param>
/// <param name="UserModified">True if the user has forced the change; false
if it is computer generated</param>
public static void OnAxesModified(axisParameters horizontalAxis,
axisParameters verticalAxis, NormalisationType normalisation, bool UserModified)
{
    if (AxesModified != null)
        AxesModified(horizontalAxis, verticalAxis, normalisation, UserModified);
}

public delegate void AxesModifiedEventHandler(axisParameters horizontalAxis,
axisParameters verticalAxis, NormalisationType normalisation, bool UserModified);
public static event AxesModifiedEventHandler AxesModified;
}

using StratSim.Model;
using StratSim.View.UserControls;
using System;
using System.Drawing;
using System.IO;
using System.Windows.Forms;

namespace StratSim.ViewModel
{
    /// <summary>

```

```

/// Contains methods for manipulating, updating, reading and writing
/// pace parameter data to and from file.
/// </summary>
class PaceParameterData : IFileController<float[,]>, IDisposable
{
    string[,] labelData = new string[3, Data.NumberOfDrivers];
    float[,] paceData = new float[8, Data.NumberOfDrivers];

    float[,] fileData = new float[8, Data.NumberOfDrivers];

    bool loadedFromFile;
    bool modified;

    SaveFileDialog SaveFileDialog;
    OpenFileDialog OpenFileDialog;

    public PaceParameterData(DataLoadedEventHandler handlerOnLoad)
    {
        modified = false;

        DataLoaded += handlerOnLoad;
        PanelControlEvents.BeforeLoadStrategiesFromData += NotifyIfModified;
        PanelControlEvents.BeforeLoadStrategiesFromFile += NotifyIfModified;
        MyEvents.SettingsModified += MyEvents_SettingsModified;
    }

    /// <summary>
    /// If data has been modified and not updated, displays a warning message
    warning the user of this
    /// before the data is used in processing.
    /// </summary>
    void NotifyIfModified()
    {
        if (modified)
        {
            string message = "Data has been modified and not updated. Save changes
now?";
            string caption = "Save Changes";
            if (Functions.StartDialog(message, caption))
            {
                SetDriverPaceData();
            }
        }
    }

    public void InitialiseData(bool loadFromFile)
    {
        loadedFromFile = loadFromFile;

        if (loadFromFile)
        {
            LoadData();
        }
        else
        {
            PopulateDataFromDrivers();
        }
    }

    void MyEvents_SettingsModified()
    {
        if (!loadedFromFile)
        {
            CheckUpdate();
        }
    }
}

```

```

public void RefreshData()
{
    if (loadedFromFile)
    {
        PopulateDataFromFile();
    }
    else
    {
        PopulateDataFromDrivers();
    }
}
public void RefreshData(bool refreshFromFile)
{
    if (refreshFromFile)
    {
        PopulateDataFromFile();
    }
    else
    {
        PopulateDataFromDrivers();
    }
    modified = false;
}

/// <summary>
/// Populates the locally stored data from the driver data held within the
program.
/// </summary>
void PopulateDataFromDrivers()
{
    int driverIndex = 0;

    foreach (Driver d in Data.Drivers)
    {
        labelData[0, driverIndex] = Convert.ToString(d.DriverNumber);
        labelData[1, driverIndex] = Convert.ToString(d.Team);
        labelData[2, driverIndex] = d.DriverName;
        paceData[0, driverIndex] = d.TopSpeed;
        paceData[1, driverIndex] = d.TyrePaceDelta;
        paceData[2, driverIndex] = d.TyreDegradation[TyreType.Prime];
        paceData[3, driverIndex] = d.TyreDegradation[TyreType.Option];
        paceData[4, driverIndex] = d.LowFuelPace;
        paceData[5, driverIndex] = d.FuelEffectPerKilo;
        paceData[6, driverIndex] = d.FuelConsumptionPerLap;
        paceData[7, driverIndex] = d.FuelLoadInP2;

        driverIndex++;
    }
    if (DataLoaded != null)
        DataLoaded();
}
/// <summary>
/// Populates driver pace data from a csv file selected by the user.
/// </summary>
void PopulateDataFromFile()
{
    for (int driverIndex = 0; driverIndex < Data.NumberOfDrivers;
driverIndex++)
    {
        //Set the label data as a string
        labelData[0, driverIndex] =
Convert.ToString(Data.Drivers[driverIndex].DriverNumber);
        labelData[1, driverIndex] = Data.Drivers[driverIndex].Team;
        labelData[2, driverIndex] = Data.Drivers[driverIndex].DriverName;

        //Set the pace data array
        for (int j = 0; j < 8; j++)

```

```

        {
            paceData[j, driverIndex] = fileData[j, driverIndex];
        }
    }
    if (DataLoaded != null)
        DataLoaded();
}

/// <summary>
/// Sets the driver pace data to the data currently held in the buffer.
/// </summary>
public void SetDriverPaceData()
{
    int driverIndex = 0;
    foreach (Driver d in Data.Drivers)
    {
        d.TopSpeed = paceData[0, driverIndex];
        d.TyrePaceDelta = paceData[1, driverIndex];
        d.TyreDegradation[TyreType.Prime] = paceData[2, driverIndex];
        d.TyreDegradation[TyreType.Option] = paceData[3, driverIndex];
        d.LowFuelPace = paceData[4, driverIndex];
        d.FuelEffectPerKilo = paceData[5, driverIndex];
        d.FuelConsumptionPerLap = paceData[6, driverIndex];
        d.FuelLoadInP2 = paceData[7, driverIndex];

        driverIndex++;
    }
    modified = false;
}
/// <summary>
/// Sets one item of pace data to the specified value.
/// </summary>
void SetPaceData(int dataIndex, int driverIndex, float value)
{
    paceData[dataIndex, driverIndex] = value;
    modified = true;
}

public float GetPaceData(int dataIndex, int driverIndex)
{
    return paceData[dataIndex, driverIndex];
}
public string GetLabelData(int parameterIndex, int driverIndex)
{
    return labelData[parameterIndex, driverIndex];
}

public string GetData(int parameterIndex, int driverIndex)
{
    if (parameterIndex <= 2)
    {
        return GetLabelData(parameterIndex, driverIndex);
    }
    else
    {
        if (parameterIndex > 2 && parameterIndex <= 10)
        {
            return Convert.ToString(GetPaceData(parameterIndex - 3,
driverIndex));
        }
        else
        {
            Program.InfoPanel.WriteData("Data index out of range");
            return "";
        }
    }
}

```

```

public void SaveData()
{
    SaveFileDialog = new SaveFileDialog();
    SaveFileDialog.InitialDirectory = Data.Settings.TimingDataBaseFolder;
    SaveFileDialog.Title = "Save File";
    SaveFileDialog.DefaultExt = ".csv";
    SaveFileDialog.FileOk += SaveFileDialog_FileOk;
    SaveFileDialog.ShowDialog();
}
void SaveFileDialog_FileOk(object sender,
System.ComponentModel.CancelEventArgs e)
{
    string file = SaveFileDialog.FileName;
    WriteToFile(file);
}
public void WriteToFile(string fileName)
{
    Program.InfoPanel.WriteData("Writing Data To File " + fileName);

    StreamWriter s = new StreamWriter(fileName);

    s.WriteLine("Driver,Speed,Delta,PrimeDeg,OptionDeg,FuelEffect,FuelConsumption,Fue
lLoad");
    for (int driverIndex = 0; driverIndex < Data.NumberOfDrivers;
driverIndex++)
    {
        s.Write(Data.Drivers[driverIndex].DriverName);
        for (int dataIndex = 0; dataIndex < 8; dataIndex++)
        {
            s.Write(",");
            s.Write(paceData[dataIndex, driverIndex]);
        }
        s.WriteLine();
    }

    s.Dispose();
}

public void LoadData()
{
    OpenFileDialog = new OpenFileDialog();
    OpenFileDialog.InitialDirectory = Data.Settings.TimingDataBaseFolder;
    OpenFileDialog.Title = "Open File";
    OpenFileDialog.DefaultExt = ".csv";
    OpenFileDialog.FileOk += OpenFileDialog_FileOk;
    OpenFileDialog.ShowDialog();
}

void OpenFileDialog_FileOk(object sender,
System.ComponentModel.CancelEventArgs e)
{
    loadedFromFile = true;
    string file = OpenFileDialog.FileName;
    ReadFromFile(file);
    PopulateDataFromFile();
}

public void ReadFromFile(string fileName)
{
    if (File.Exists(fileName))
    {
        Program.InfoPanel.WriteData("Reading data from file " + fileName);

        StreamReader s = new StreamReader(fileName);
        //read titles to remove.
        s.ReadLine();
    }
}

```

```

        for (int driverIndex = 0; driverIndex < Data.NumberOfDrivers;
driverIndex++)
    {
        string[] inputData = s.ReadLine().Split(',');
        for (int dataIndex = 0; dataIndex < 8; dataIndex++)
        {
            fileData[dataIndex, driverIndex] = float.Parse(inputData[dataIndex
+ 1]);
        }
    }
    else
    {
        Program.InfoPanel.WriteData("File " + fileName + " did not exist");
    }
}

/// <summary>
/// Sets the formatting of the text box when its value is changed
/// </summary>
/// <param name="e">The event arguments generated from the value changed
event</param>
void SetColours(TextChangedEventArgs e)
{
    float parameterValue = float.Parse(e.Value);
    //Set the fill colour if the value is not the default
    if (IsDefault(parameterValue, e.ParameterIndex - 3, e.DriverIndex)) {
e.TextBox.BackColor = Color.Yellow; }
    else
    {
        e.TextBox.BackColor = Color.Cyan;
        if (!e.TextBox.TextChanged) { e.TextBox.BackColor = Color.White; }
    }

    //Set the fore colour if the value is outside of a given range.
    if (IsAnomaly(parameterValue, e.ParameterIndex - 3)) { e.TextBox.ForeColor
= Color.Red; }
    else { e.TextBox.ForeColor = ParameterTextBox.DefaultForeColor; }
}

/// <summary>
/// Updates a parameter value after the text box has been altered
/// </summary>
/// <param name="e">The text changed event arguments generated when the text
box text has been changed</param>
public void UpdateParameter(TextChangedEventArgs e)
{
    SetPaceData(e.ParameterIndex - 3, e.DriverIndex, float.Parse(e.Value));
    SetColours(e);

    if (e.ParameterIndex == 10)
    {
        string message = "Changing this value will cause changes to all other
pace parameters. "
            + "To see these changes, the values must be updated."
            + "\r\nUpdate values now?";
        string caption = "Update Values?";

        if (Functions.StartDialog(message, caption))
        {
            EditDriverParameters();
        }
    }
}

/// <summary>

```

```

    /// Returns true if the current value is the same as the default value,
    stored in the settings file.
    /// </summary>
    public bool IsDefault(float currentValue, int dataIndex, int driverIndex)
    {
        bool sameAsDefault = false;

        switch (dataIndex)
        {
            case 0: sameAsDefault = (paceData[dataIndex, driverIndex] ==
Data.Settings.DefaultTopSpeed); break;
            case 1: sameAsDefault = (paceData[dataIndex, driverIndex] ==
Data.Settings.DefaultCompoundDelta); break;
            case 2: sameAsDefault = (paceData[dataIndex, driverIndex] ==
Data.Settings.DefaultPrimeDegradation); break;
            case 3: sameAsDefault = (paceData[dataIndex, driverIndex] ==
Data.Settings.DefaultOptionDegradation); break;
            case 4: sameAsDefault = (paceData[dataIndex, driverIndex] ==
Data.Settings.DefaultPace); break;
            case 5: sameAsDefault = (paceData[dataIndex, driverIndex] ==
Data.Settings.DefaultFuelEffect); break;
            case 6: sameAsDefault = (paceData[dataIndex, driverIndex] ==
Data.Tracks[Data.RaceIndex].fuelPerLap); break;
            case 7: sameAsDefault = (paceData[dataIndex, driverIndex] ==
Data.Settings.DefaultP2Fuel); break;
            default: sameAsDefault = false; break;
        }

        return sameAsDefault;
    }

    /// <summary>
    /// Returns true if the current value is outside of 25% of the default value,
    stored in the settings.
    /// </summary>
    public bool IsAnomaly(float currentValue, int dataIndex)
    {
        bool withinBounds = true;
        float defaultValue;

        switch (dataIndex)
        {
            case 0: defaultValue = Data.Settings.DefaultTopSpeed; break;
            case 1: defaultValue = Data.Settings.DefaultCompoundDelta; break;
            case 2: defaultValue = Data.Settings.DefaultPrimeDegradation; break;
            case 3: defaultValue = Data.Settings.DefaultOptionDegradation; break;
            case 5: defaultValue = Data.Settings.DefaultFuelEffect; break;
            case 6: defaultValue = Data.Tracks[Data.RaceIndex].fuelPerLap; break;
            case 7: defaultValue = Data.Settings.DefaultP2Fuel; break;
            default: defaultValue = 0; break;
        }

        if (dataIndex != 4)
            withinBounds = (Math.Abs(currentValue - defaultValue) <=
Math.Abs(defaultValue * 0.4));

        return !withinBounds;
    }

    /// <summary>
    /// Recalculates the driver pace paramters after a significant change.
    /// </summary>
    public void EditDriverParameters()
    {
        //Recalculate all pace parameters
        Functions.CalculatePaceParameters(Data.RaceIndex);
        //Refresh the data held about the pace parameters
    }
}

```

```

        RefreshData(false);
        Program.InfoPanel.WriteData("Driver parameters edited");
    }

    /// <summary>
    /// Displays an error message to the user if a new strategy optimisation is
    started without
    /// completing the parameter update.
    /// </summary>
    void CheckUpdate()
    {
        string message = "Settings have been altered. This may affect the
calculated pace parameters." +
            "Recalculate parameters now?";
        string caption = "Recalculate Parameters";
        if (Functions.StartDialog(message, caption))
        {
            EditDriverParameters();
        }
    }

    public void Dispose()
    {
        SaveFileDialog.Dispose();
        OpenFileDialog.Dispose();
    }

    public delegate void DataLoadedEventHandler();
    public event DataLoadedEventHandler DataLoaded;

    public float[,] FileData
    { get { return fileData; } }
}

}

using StratSim.View.MyFlowLayout;

namespace StratSim.ViewModel
{
    /// <summary>
    /// Contains events which control hiding and showing of panels on a form
    /// </summary>
    class PanelControlEvents
    {
        /////Shows the version information window.
        public static void OnShowVersionInfo()
        {
            Program.ShowVersionInfo();
        }

        public delegate void LoadFromFileEventHandler();
        public static event LoadFromFileEventHandler LoadPaceParametersFromFile;
        public static event LoadFromFileEventHandler LoadStrategiesFromFile;
        public static event LoadFromFileEventHandler BeforeLoadStrategiesFromFile;
        public static event LoadFromFileEventHandler StrategiesLoaded;
        public delegate void LoadFromRaceEventHandler();
        public static event LoadFromRaceEventHandler LoadPaceParametersFromRace;
        public delegate void LoadFromDataEventHandler();
        public static event LoadFromDataEventHandler LoadStrategiesFromData;
        public static event LoadFromDataEventHandler BeforeLoadStrategiesFromData;
        public delegate void StartRaceEventHandler();
        public static event StartRaceEventHandler StartRaceFromStrategies;
        public static event StartRaceEventHandler BeforeStartRaceFromStrategies;
        public delegate void ViewPanelEventHandler(MainForm form);
        public static event ViewPanelEventHandler ShowInfoPanel;
        public static event ViewPanelEventHandler ShowSettingsPanel;
    }
}

```

```

public static event ViewPanelEventHandler ShowContentTabControl;
public static event ViewPanelEventHandler ShowDriverSelectPanel;
public static event ViewPanelEventHandler ShowGraph;
public static event ViewPanelEventHandler ShowAxes;
public static event ViewPanelEventHandler ShowPaceParameters;
public static event ViewPanelEventHandler ShowStrategies;
public static event ViewPanelEventHandler ShowArchives;
public static event ViewPanelEventHandler ShowDataInput;
public static event ViewPanelEventHandler RemoveGraphPanels;

//The following routines fire events to show particular panels.
public static void OnShowInfoPanel(MainForm form)
{
    if (ShowInfoPanel != null)
        ShowInfoPanel(form);
}
public static void OnShowSettingsPanel(MainForm form)
{
    if (ShowSettingsPanel != null)
        ShowSettingsPanel(form);
}
public static void OnShowContentTabControl(MainForm form)
{
    if (ShowContentTabControl != null)
        ShowContentTabControl(form);
}
public static void OnShowDriverSelectPanel(MainForm form)
{
    if (ShowDriverSelectPanel != null)
        ShowDriverSelectPanel(form);
}
public static void OnShowGraph(MainForm form)
{
    if (ShowGraph != null)
        ShowGraph(form);
}
public static void OnShowAxes(MainForm form)
{
    if (ShowAxes != null)
        ShowAxes(form);
}
public static void OnLoadPaceParametersFromFile()
{
    if (LoadPaceParametersFromFile != null)
        LoadPaceParametersFromFile();
}
public static void OnLoadStrategiesFromFile()
{
    //The three stages to the event allow different stages of processing to be
    completed in the correct order.
    if (BeforeLoadStrategiesFromFile != null)
        BeforeLoadStrategiesFromFile();
    if (LoadStrategiesFromFile != null)
        LoadStrategiesFromFile();
    if (StrategiesLoaded != null)
        StrategiesLoaded();
}

//Loads pace parameters and strategies into main memory.
public static void OnLoadPaceParametersFromRace()
{
    if (LoadPaceParametersFromRace != null)
        LoadPaceParametersFromRace();
}
public static void OnLoadStrategiesFromData()
{
    if (BeforeLoadStrategiesFromData != null)

```

```

        BeforeLoadStrategiesFromData();
        if (LoadStrategiesFromData != null)
            LoadStrategiesFromData();
    }
    public static void OnStartRaceFromStrategies()
    {
        if (BeforeStartRaceFromStrategies != null)
            BeforeStartRaceFromStrategies();
        if (StartRaceFromStrategies != null)
            StartRaceFromStrategies();
    }
    public static void OnShowPaceParameters(MainForm form)
    {
        if (ShowPaceParameters != null)
            ShowPaceParameters(form);
    }
    public static void OnShowStrategies(MainForm form)
    {
        if (ShowStrategies != null)
            ShowStrategies(form);
    }
    public static void OnShowArchives(MainForm form)
    {
        if (ShowArchives != null)
            ShowArchives(form);
    }
    public static void OnShowDataInput(MainForm form)
    {
        if (ShowDataInput != null)
            ShowDataInput(form);
    }
    /// <summary>
    /// Fires the RemoveGraphPanels event, clearing the main graph panels from the
    screen to allow the screen to resize accordingly.
    /// </summary>
    public static void OnRemoveGraphPanels(MainForm form)
    {
        if (RemoveGraphPanels != null)
            RemoveGraphPanels(form);
    }
}

using StratSim.Model;
using System;
using System.Collections.Generic;
using System.IO;
using System.Windows.Forms;

namespace StratSim.ViewModel
{
    /// <summary>
    /// Contains methods for manipulating, updating, reading and writing
    /// strategy data to and from file.
    /// </summary>
    class StrategyViewerData : IFileController<Strategy[]>, IDisposable
    {
        Strategy[] strategies = new Strategy[Data.NumberOfDrivers];
        Strategy[] fileData = new Strategy[Data.NumberOfDrivers];

        bool loadedFromFile;
        bool modified;

        SaveFileDialog SaveFileDialog;
        OpenFileDialog OpenFileDialog;
}

```

```

    /// <summary>
    /// Creates a new instance of the StrategyViewerData class.
    /// </summary>
    /// <param name="handlerOnLoad">An event handler to link the the DataLoaded
event; it will be executed
    /// whenever the strategies are fully loaded</param>
public StrategyViewerData(DataLoadedEventHandler handlerOnLoad)
{
    modified = false;

    //Subscribes to the data loaded event so that data can be displayed when it
has been loaded
    DataLoaded += handlerOnLoad;
    MyEvents.SettingsModified += MyEvents_SettingsModified;
    MyEvents.StrategyModified += MyEvents_StrategyModified;
    PanelControlEvents.BeforeStartRaceFromStrategies +=
PanelControlEvents_StartRaceFromStrategies;
}

void PanelControlEvents_StartRaceFromStrategies()
{
    if (modified)
    {
        string message = "Data has been modified and not updated. Save changes
now?";
        string caption = "Save Changes";
        if (Functions.StartDialog(message, caption))
        {
            SetDriverStrategies();
        }
    }
}

public void InitialiseData(bool loadFromFile)
{
    loadedFromFile = loadFromFile;

    if (loadFromFile)
    {
        LoadData();
    }
    else
    {
        PopulateDataFromDrivers();
    }
}

void MyEvents_StrategyModified(int driverIndex, Strategy strategy, bool
showAllOnGraph)
{
    modified = true;
    SetStrategy(driverIndex, strategy);
    MyEvents.OnStrategyModificationsComplete(showAllOnGraph, false);
}
void MyEvents_SettingsModified()
{
    if (!loadedFromFile)
    {
        CheckUpdate();
    }
}

public void RefreshData()
{
    RefreshData(loadedFromFile);
}
public void RefreshData(bool refreshFromFile)

```

```

{
    if (refreshFromFile)
    {
        PopulateDataFromFile();
    }
    else
    {
        PopulateDataFromDrivers();
    }
    modified = false;
    MyEvents.OnStrategyModificationsComplete(true, true);
}

void PopulateDataFromDrivers()
{
    for (int driverIndex = 0; driverIndex < Data.NumberOfDrivers;
driverIndex++)
    {
        strategies[driverIndex] = new
Strategy(Data.Drivers[driverIndex].Strategy);
    }
    if (DataLoaded != null)
        DataLoaded();
}
void PopulateDataFromFile()
{
    for (int driverIndex = 0; driverIndex < Data.NumberOfDrivers;
driverIndex++)
    {
        strategies[driverIndex] = new Strategy(fileData[driverIndex]);
    }
    if (DataLoaded != null)
        DataLoaded();
}

/// <summary>
/// Gets a specific strategy from the list of strategies
/// </summary>
/// <param name="driverIndex">The index at which the strategy is held</param>
public Strategy GetStrategy(int driverIndex)
{
    return new Strategy(strategies[driverIndex]);
}
/// <summary>
/// Gets a specific stint from a specific strategy
/// </summary>
/// <param name="driverIndex">The index at which the strategy is held</param>
/// <param name="stintIndex">The index of the stint within the
strategy</param>
public Stint GetStint(int driverIndex, int stintIndex)
{
    return (Stint)strategies[driverIndex].Stints[stintIndex].Clone();
}
/// <summary>
/// Sets a locally held strategy to the specified strategy
/// </summary>
/// <param name="driverIndex">The index of the strategy to overwrite</param>
/// <param name="value">The new strategy to set as the locally held
strategy</param>
void SetStrategy(int driverIndex, Strategy value)
{
    strategies[driverIndex] = (Strategy)value.Clone();
}

/// <summary>
/// Sets all driver strategies (to be used in a race) to those held locally.
/// </summary>

```

```

    public void SetDriverStrategies()
    {
        for (int driverIndex = 0; driverIndex < Data.NumberOfDrivers;
driverIndex++)
        {
            Data.Drivers[driverIndex].Strategy =
(Strategy)strategies[driverIndex].Clone();
        }
        modified = false;
    }

    public void SaveData()
    {
        SaveFileDialog = new SaveFileDialog();
        SaveFileDialog.InitialDirectory = Data.Settings.TimingDataBaseFolder;
        SaveFileDialog.Title = "Save File";
        SaveFileDialog.DefaultExt = ".csv";
        SaveFileDialog.FileOk += SaveFileDialog_FileOk;
        SaveFileDialog.ShowDialog();
    }

    void SaveFileDialog_FileOk(object sender,
System.ComponentModel.CancelEventArgs e)
    {
        string file = SaveFileDialog.FileName;
        WriteToFile(file);
    }

    public void WriteToFile(string fileName)
    {
        //writes in database format.
        Program.InfoPanel.WriteData("Writing Data To File " + fileName);

        StreamWriter w = new StreamWriter(fileName);

        //Write headings:
        w.WriteLine("Driver,Start Lap,Length,Tyre");
        for (int driverIndex = 0; driverIndex < Data.NumberOfDrivers;
driverIndex++)
        {
            foreach (Stint s in strategies[driverIndex].Stints)
            {
                w.Write(Data.Drivers[driverIndex].DriverName);
                w.Write(',');
                s.WriteStintData(ref w);
                w.WriteLine();
            }
        }

        w.Dispose();
    }

    public void LoadData()
    {
        OpenFileDialog = new OpenFileDialog();
        OpenFileDialog.InitialDirectory = Data.Settings.TimingDataBaseFolder;
        OpenFileDialog.Title = "Open File";
        OpenFileDialog.DefaultExt = ".csv";
        OpenFileDialog.FileOk += OpenFileDialog_FileOk;
        OpenFileDialog.ShowDialog();
    }

    void OpenFileDialog_FileOk(object sender,
System.ComponentModel.CancelEventArgs e)
    {
        loadedFromFile = true;
        string file = OpenFileDialog.FileName;

        if (File.Exists(file))
        {
    
```

```

    try
    {
        ReadFromFile(file);
    }
    catch (IOException exception)
    {
        Console.Out.WriteLine("Error reading from {0}. Message: {1}", file,
exception.Message);
    }
    finally
    {
        PopulateDataFromFile();
    }
}
public void ReadFromFile(string fileName)
{
    Program.InfoPanel.WriteData("Reading data from file " + fileName);

    int driverIndex = 0;
    int stintIndex = 0;

    int startLap, length;
    TyreType tyreType;
    string driverName;

    List<Stint> listOfStintsToAdd = new List<Stint>();
    Stint tempStint = new Stint();

    StreamReader s = new StreamReader(fileName);
    //read titles to remove.
    s.ReadLine();

    stintIndex = 0;
    listOfStintsToAdd.Clear();

    while (!s.EndOfStream)
    {
        string[] inputData = s.ReadLine().Split(',');
        driverName = inputData[0];
        startLap = int.Parse(inputData[1]);
        length = int.Parse(inputData[2]);
        if (inputData[3] == "Prime") { tyreType = TyreType.Prime; }
        else { tyreType = TyreType.Option; }

        if (driverName != Data.Drivers[driverIndex].DriverName)
        {
            fileData[driverIndex] = new Strategy(Data.Drivers[driverIndex],
listOfStintsToAdd);
            driverIndex++;
            stintIndex = 0;
            listOfStintsToAdd.Clear();
        }

        tempStint = new Stint(Data.Drivers[driverIndex], startLap, tyreType,
length);
        listOfStintsToAdd.Add(tempStint);

        stintIndex++;
    }
    fileData[driverIndex] = new Strategy(Data.Drivers[driverIndex],
listOfStintsToAdd);
}

void CheckUpdate()

```

```
{  
    string message = "Settings have been altered. This may affect the way  
strategies are calculated." +  
        "Recalculate strategies now?";  
    string caption = "Recalculate Strategies";  
    if (Functions.StartDialog(message, caption))  
    {  
        Functions.OptimiseAllStrategies(Data.RaceIndex);  
        RefreshData(false);  
    }  
}  
  
public void Dispose()  
{  
    SaveFileDialog.Dispose();  
    OpenFileDialog.Dispose();  
}  
  
public delegate void DataLoadedEventHandler();  
public event DataLoadedEventHandler DataLoaded;  
  
public Strategy[] FileData  
{ get { return fileData; } }  
}  
}
```

## Unit Testing

### Contents

Data Input Testing.....	1
Overtake Testing.....	2
Race Testing.....	3
Quicksort Testing.....	6
Strategy Testing .....	8
Validation Testing.....	9

### Specification Unit Testing

```

using System;
using Microsoft.VisualStudio.TestTools.UnitTesting;

using StratSim;
using StratSim.Model;
using StratSim.Model.Files;
using StratSim.View.MyFlowLayout;
using StratSim.View.Panels;
using System.Collections.Generic;

namespace StratSimTest
{
    public class SpecificationUnitTesting
    {
        [TestClass]
        public class DataInputUnitTesting
        {
            [TestMethod]
            public void TestSetSpeedDataFromFile()
            {
                string lineOfData = "0,VETTEL,299.243";
                var data = new StratSim.Model.Data();
                var speedData = new StratSim.Model.Files.SpeedData();

                Driver.InitialiseDrivers();
                speedData.SetDriverSpeed(lineOfData);

                Assert.AreEqual(299.243F, Data.Drivers[0].topSpeed);
            }

            [TestMethod]
            public void TestGetLapTimeFromString()
            {
                string lapTimeString = "1:32.554 ";
                float lapTime;

                var lapData = new LapData();

                lapTime = lapData.GetLapTime(lapTimeString);

                Assert.AreEqual(92.554F, lapTime);
            }

            [TestMethod]
            public void TestIntegrationPaceParameterAssignment()
            {
                //direct from PDF files:
                float expectedTopSpeed = 336.1F;
                float expectedFastestLap = 83.755F;
                int raceIndex = 11; //Monza
            }
        }
    }
}

```

```

        Data data = new Data();
        Driver.InitialiseDrivers();
        Track.InitialiseTracks();

        MainForm form = new MainForm(0);
        Program.InfoPanel = new InfoPanel(form);
        Program.PopulateDriverDataFromFiles(raceIndex);
        Functions.CalculateStrategyParameters(raceIndex);

        Assert.AreEqual(expectedTopSpeed, Data.Drivers[0].topSpeed);
        Assert.AreEqual(expectedFastestLap, Data.Drivers[0].pace);
    }
}

[TestClass]
public class OvertakeUnitTesting
{
    [TestMethod]
    public void TestOvertakeProbabilityShouldOccur()
    {
        float paceDelta = 0.8F;
        float speedDelta = 20F;
        float totalDelta = -0.4F;
        float requiredPaceDelta = 0.4F;
        float requiredSpeedDelta = 10F;

        var race = new Race();

        float overtakeProbability = race.GetOvertakeProbability(paceDelta,
speedDelta, totalDelta, requiredPaceDelta, requiredSpeedDelta);

        Assert.AreEqual(1F, overtakeProbability);
    }

    [TestMethod]
    public void TestOvertakeProbabilityMightOccur()
    {
        float paceDelta = 0.6F;
        float speedDelta = 15F;
        float totalDelta = 0.2F;
        float requiredPaceDelta = 0.4F;
        float requiredSpeedDelta = 10F;

        var race = new Race();

        float overtakeProbability = race.GetOvertakeProbability(paceDelta,
speedDelta, totalDelta, requiredPaceDelta, requiredSpeedDelta);

        Assert.AreEqual(0.25F, overtakeProbability, 0.001);
    }

    [TestMethod]
    public void TestOvertakeProbabilityShouldNotOccur()
    {
        float paceDelta = 0.4F;
        float speedDelta = 10F;
        float totalDelta = 0.2F;
        float requiredPaceDelta = 0.4F;
        float requiredSpeedDelta = 10F;

        var race = new Race();

        float overtakeProbability = race.GetOvertakeProbability(paceDelta,
speedDelta, totalDelta, requiredPaceDelta, requiredSpeedDelta);

        Assert.AreEqual(0F, overtakeProbability);
    }
}

```

```

}

[TestClass]
public class RaceUnitTesting
{
    Race SetupARaceFromFiles(int raceIndex)
    {
        Data data = new Data();
        Driver.InitialiseDrivers();
        Track.InitialiseTracks();

        MainForm form = new MainForm(0);
        Program.InfoPanel = new InfoPanel(form);
        Program.PopulateDriverDataFromFiles(raceIndex);
        Functions.CalculateStrategyParameters(raceIndex);
        Functions.manageStrategies(raceIndex);
        Strategy[] strategies = new Strategy[Data.numberOfDrivers];

        for (int i = 0; i < Data.numberOfDrivers; i++)
        {
            strategies[i] = Data.Drivers[i].Strategy;
        }

        return new Race(raceIndex, strategies, form);
    }

    [TestMethod]
    public void TestGridOrderSimulation()
    {
        int raceIndex = 14; //Suzuka
        float pace = 0;
        var race = SetupARaceFromFiles(raceIndex);

        race.setupGrid();

        bool ascending = true;
        for (int i = 0; i < Data.numberOfDrivers - 1; i++)
        {
            if (race.Strategies[i].Driver.pace > race.Strategies[i + 1].Driver.pace) { ascending = false; }
            pace = race.Strategies[i].Driver.pace;
        }

        Assert.IsTrue(ascending);
    }

    Strategy[] SetupStrategiesForPitStopSimulation(int trackIndex, int
numberOfStrategies)
    {
        var data = new Data();
        Track.InitialiseTracks();
        Driver[] drivers = new Driver[numberOfStrategies];
        Strategy[] strategies = new Strategy[numberOfStrategies];
        MainForm form = new MainForm(0);
        var parameters = new[] { 320F, -0.8F, 0.1F, 0.3F, 200F, 2.5F, 0.02F, 60F
};

        for (int i = 0; i < numberOfStrategies; i++)
        {
            drivers[i] = new Driver(parameters);
            drivers[i].driverIndex = i;
            strategies[i] = drivers[i].OptimiseStrategy(drivers[i], trackIndex);
        }

        return strategies;
    }
}

```

```

        Strategy[] SetupRaceForPitStopSimulation(Strategy[] strategies, float[]
cumulativeTimes)
{
    Strategy[] tempStrategies = strategies;
    for (int i = 0; i < strategies.Length; i++)
    {
        tempStrategies[i].CumulativeTime = cumulativeTimes[i];
    }

    return tempStrategies;
}

List<PitStop> SetupPitStopListForPitStopSimulation(int driverToPit, int
trackIndex)
{
    return new List<PitStop> { new PitStop(driverToPit, 1, trackIndex) };
}

Strategy[] TestPitStopSimulation(int[] driversToPit, float[]
cumulativeTimes)
{
    int numberOfStrategies = 3;
    int trackIndex = 14;
    Strategy[] strategies = SetupStrategiesForPitStopSimulation(trackIndex,
numberOfStrategies);
    strategies = SetupRaceForPitStopSimulation(strategies, cumulativeTimes);
    List<PitStop> pitstops = new List<PitStop>();
    foreach (int driver in driversToPit)
    {
        pitstops.Add(new PitStop(driver, 1, trackIndex));
    }

    PitStop.updatePositions(ref strategies, pitstops, trackIndex);

    return strategies;
}

[TestMethod]
public void TestPitStopSimulationNoOvertakesLastDriver()
{
    //Arrange
    int[] driverToPit = new[] { 2 };
    float[] cumulativeTimes = new[] { 100F, 110F, 140F };

    //Act
    Strategy[] strategies = TestPitStopSimulation(driverToPit,
cumulativeTimes);

    //Assert
    Assert.AreEqual(0, strategies[0].Driver.driverIndex);
    Assert.AreEqual(1, strategies[1].Driver.driverIndex);
    Assert.AreEqual(2, strategies[2].Driver.driverIndex);
}

[TestMethod]
public void TestPitStopSimulationNoOvertakesFirstDriver()
{
    //Arrange
    int[] driverToPit = new[] { 0 };
    float[] cumulativeTimes = new[] { 100F, 130F, 140F };

    //Act
    Strategy[] strategies = TestPitStopSimulation(driverToPit,
cumulativeTimes);

    //Assert
    Assert.AreEqual(0, strategies[0].Driver.driverIndex);
}

```

```

        Assert.AreEqual(1, strategies[1].Driver.driverIndex);
        Assert.AreEqual(2, strategies[2].Driver.driverIndex);
    }

    [TestMethod]
    public void TestPitStopSimulationFirstDropsToLast()
    {
        //Arrange
        int[] driverToPit = new[] { 0 };
        float[] cumulativeTimes = new[] { 100F, 110F, 120F };

        //Act
        Strategy[] strategies = TestPitStopSimulation(driverToPit,
cumulativeTimes);

        //Assert
        Assert.AreEqual(1, strategies[0].Driver.driverIndex);
        Assert.AreEqual(2, strategies[1].Driver.driverIndex);
        Assert.AreEqual(0, strategies[2].Driver.driverIndex);
    }

    [TestMethod]
    public void TestPitStopSimulationMiddleDropsToLast()
    {
        //Arrange
        int[] driverToPit = new[] { 1 };
        float[] cumulativeTimes = new[] { 100F, 110F, 120F };

        //Act
        Strategy[] strategies = TestPitStopSimulation(driverToPit,
cumulativeTimes);

        //Assert
        Assert.AreEqual(0, strategies[0].Driver.driverIndex);
        Assert.AreEqual(2, strategies[1].Driver.driverIndex);
        Assert.AreEqual(1, strategies[2].Driver.driverIndex);
    }

    [TestMethod]
    public void TestPitStopSimulationAllPit()
    {
        //Arrange
        int[] driverToPit = new[] { 0, 1, 2 };
        float[] cumulativeTimes = new[] { 100F, 110F, 120F };

        //Act
        Strategy[] strategies = TestPitStopSimulation(driverToPit,
cumulativeTimes);

        //Assert
        Assert.AreEqual(0, strategies[0].Driver.driverIndex);
        Assert.AreEqual(1, strategies[1].Driver.driverIndex);
        Assert.AreEqual(2, strategies[2].Driver.driverIndex);
    }

    public void TestPitStopSimulationUndercut()
    {
        //Arrange
        float[] cumulativeTimes = new[] { 100F, 110F, 140F };

        int[] driverToPit = new[] { 0 };
        Strategy[] strategies = TestPitStopSimulation(driverToPit,
cumulativeTimes);

        driverToPit = new[] { 1 };
        cumulativeTimes[0] = strategies[1].CumulativeTime;
        cumulativeTimes[1] = strategies[0].CumulativeTime;
        cumulativeTimes[2] = strategies[2].CumulativeTime;
        strategies = TestPitStopSimulation(driverToPit, cumulativeTimes);
    }
}

```

```
//Assert
Assert.AreEqual(0, strategies[0].Driver.driverIndex);
Assert.AreEqual(1, strategies[1].Driver.driverIndex);
Assert.AreEqual(2, strategies[2].Driver.driverIndex);
}

}

[TestClass]
public class QuickSortUnitTesting
{
    [TestMethod]
    public void TestQuickSortAscendingAlreadySorted()
    {
        int[] ints = { 1, 2, 3, 4, 5, 6, 7 };

        Functions.QuickSort<int>(ref ints, 0, ints.Length - 1, (a, b) => a > b);

        bool ascending = true;
        for (int i = 0; i < ints.Length - 1; i++)
        {
            if (ints[i] > ints[i + 1]) { ascending = false; }
        }

        Assert.IsTrue(ascending);
    }

    [TestMethod]
    public void TestQuickSortAscending()
    {
        int[] ints = { 25, 16, 14, 16, 99, 64, 24, 58, 67, 9, 9 };

        Functions.QuickSort<int>(ref ints, 0, ints.Length - 1, (a, b) => a > b);

        bool ascending = true;
        for (int i = 0; i < ints.Length - 1; i++)
        {
            if (ints[i] > ints[i + 1]) { ascending = false; }
        }

        Assert.IsTrue(ascending);
    }

    [TestMethod]
    public void TestQuickSortDescending()
    {
        int[] ints = { 25, 16, 14, 16, 99, 64, 24, 58, 67, 9, 9 };

        Functions.QuickSort<int>(ref ints, 0, ints.Length - 1, (a, b) => a < b);

        bool descending = true;
        for (int i = 0; i < ints.Length - 1; i++)
        {
            if (ints[i] < ints[i + 1]) { descending = false; }
        }

        Assert.IsTrue(descending);
    }
}
```

Strategy Unit Testing

```

using System;
using Microsoft.VisualStudio.TestTools.UnitTesting;

using StratSim.Model;

namespace StratSimTest
{
[DataSource(@"Provider=Microsoft.ACE.OLEDB.12.0; Data
Source=|DataDirectory|\\StratSim.accdb;", "StrategyTestData")]
    [TestMethod]
    public void TestStintLengthOptimisation()
    {
        int raceLaps = 58;
        var parameters = new[] { 320F, -0.8F, 0.1F, 0.3F, 200F, 2.5F, 0.02F, 60F };

        Driver driver = new Driver(parameters);
        Strategy strategy = new Strategy(driver);

        int primeStints = Convert.ToInt32(TestContext.DataRow["Prime Stints"]);
        int optionStints = Convert.ToInt32(TestContext.DataRow["Option Stints"]);
        int expectedPrimeLength = Convert.ToInt32(TestContext.DataRow["Act
Prime"]);
        int expectedOptionLength = Convert.ToInt32(TestContext.DataRow["Act
Option"]);

        strategy.SetStintLengths(raceLaps, primeStints, optionStints);

        Assert.AreEqual(expectedPrimeLength, strategy.PrimeLaps);
        Assert.AreEqual(expectedOptionLength, strategy.OptionLaps);
    }

    private TestContext testContextInstance;
    public TestContext TestContext
    {
        get { return testContextInstance; }
        set { testContextInstance = value; }
    }

    [TestMethod]
    public void TestStrategyOptimisation()
    {
        int primeStints = 2;
        int optionStints = 1;
        var parameters = new[] { 320F, -0.8F, 0.1F, 0.3F, 200F, 2.5F, 0.02F, 60F };

        Data data = new Data();
        Track.InitialiseTracks();
        Driver driver = new Driver(parameters);
        Strategy strategy = new Strategy(primeStints + optionStints, primeStints,
driver);

        driver.Strategy = driver.OptimiseStrategy(driver, 14);

        for (int stintNo = 0; stintNo < primeStints + optionStints; stintNo++)
        {
            Assert.AreEqual(strategy.Stints[stintNo].stintLength,
driver.Strategy.Stints[stintNo].stintLength);
            Assert.AreEqual(strategy.Stints[stintNo].tyreType,
driver.Strategy.Stints[stintNo].tyreType);
        }
    }
}

```

Validaiton Unit Testing

```

using System;
using Microsoft.VisualStudio.TestTools.UnitTesting;

using StratSim.Model;

namespace StratSimTest
{
    [TestClass]
    public class ValidationUnitTest
    {
        [TestMethod]
        public void TestTypicalValidationData()
        {
            //arrange
            float typical = 0.5F;
            bool expected = false;

            TestValidationInclusiveWithValue(typical, expected);
        }

        [TestMethod]
        public void TestErroneousTooLargeValidationData()
        {
            float erroneousTooLarge = 1.5F;
            bool expected = true;
            TestValidationInclusiveWithValue(erroneousTooLarge, expected);
        }

        [TestMethod]
        public void TestErroneousTooSmallValidationData()
        {
            float erroneousTooSmall = -2F;
            bool expected = true;
            TestValidationInclusiveWithValue(erroneousTooSmall, expected);
        }

        [TestMethod]
        public void TestLowerBoundaryValidationData()
        {
            float lowerBoundary = 0F;
            bool expected = false;
            TestValidationInclusiveWithValue(lowerBoundary, expected);
        }

        [TestMethod]
        public void TestUpperBoundaryValidationData()
        {
            float upperBoundary = 1F;
            bool expected = false;
            TestValidationInclusiveWithValue(upperBoundary, expected);
        }

        //Exclusive
        [TestMethod]
        public void TestTypicalValidationDataExclusive()
        {
            //arrange
            float typical = 0.5F;
            bool expected = false;

            TestValidationExclusiveWithValue(typical, expected);
        }

        [TestMethod]
        public void TestErroneousTooLargeValidationDataExclusive()
        {
            float erroneousTooLarge = 1.5F;
            bool expected = true;
            TestValidationExclusiveWithValue(erroneousTooLarge, expected);
        }
    }
}

```

```

[TestMethod]
public void TestErroneousTooSmallValidationDataExclusive()
{
    float erroneousTooSmall = -2F;
    bool expected = true;
    TestValidationExclusiveWithValue(erroneousTooSmall, expected);
}
[TestMethod]
public void TestLowerBoundaryValidationDataExclusive()
{
    float lowerBoundary = 0F;
    bool expected = true;
    TestValidationExclusiveWithValue(lowerBoundary, expected);
}
[TestMethod]
public void TestUpperBoundaryValidationDataExclusive()
{
    float upperBoundary = 1F;
    bool expected = true;
    TestValidationExclusiveWithValue(upperBoundary, expected);
}

/// <summary>
/// Tests the validation command for a given value and compares with the
expected result
/// </summary>
/// <param name="value">The value to validate</param>
/// <param name="expectedResult">The expected result of the validation</param>
void TestValidationInclusiveWithValue(float value, bool expectedResult)
{
    bool actual;

    actual = RunValidationInclusiveRoutineToTest(value);

    Assert.AreEqual(expectedResult, actual);
}
void TestValidationExclusiveWithValue(float value, bool expectedResult)
{
    bool actual;

    actual = RunValidationExclusiveRoutineToTest(value);

    Assert.AreEqual(expectedResult, actual);
}

/// <summary>
/// Tests the validation of a value inclusively between 0 and 1.
/// </summary>
/// <param name="value">The floating point value to test</param>
/// <returns>True if the value is incorrect</returns>
bool RunValidationInclusiveRoutineToTest(float value)
{
    float lowerBound = 0F;
    float upperBound = 1F;
    string field = "";
    bool incorrectValue = false;

    Functions.validateBetweenInclusive(value, lowerBound, upperBound, field,
ref incorrectValue, false);

    return incorrectValue;
}

bool RunValidationExclusiveRoutineToTest(float value)
{
    float lowerBound = 0F;

```

```
    float upperBound = 1F;
    string field = "";
    bool incorrectValue = false;

    Functions.validateBetweenExclusive(value, lowerBound, upperBound, field,
ref incorrectValue, false);

    return incorrectValue;
}
}
}
```

## Screen shots of working program

### Introduction

The screenshots below depict the state of the system during normal operation. The screenshots are taken from a fully released version of the system, and all screenshots are taken in the same sequence – sequences are shortened but I did not need to restart the program at any stage, and nor did it crash.

The screenshots are annotated to describe the relationship between the GUI and the code objects, and to describe the processes occurring at each stage. There are also brief comments on what the system is doing at each stage, but all instructions for how to use the system are left to the user manual.

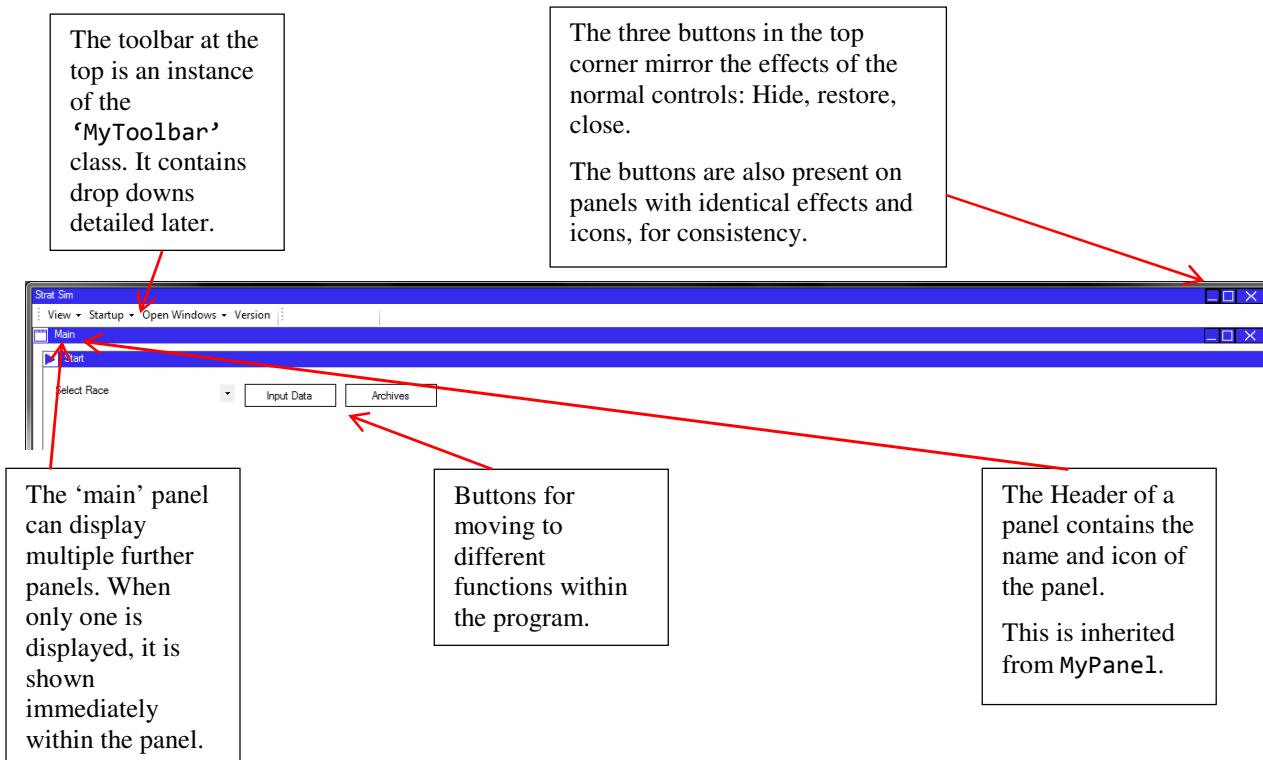
Where code features are shown their names are given in **Consolas** font in order to be linked back to the program listing where required.

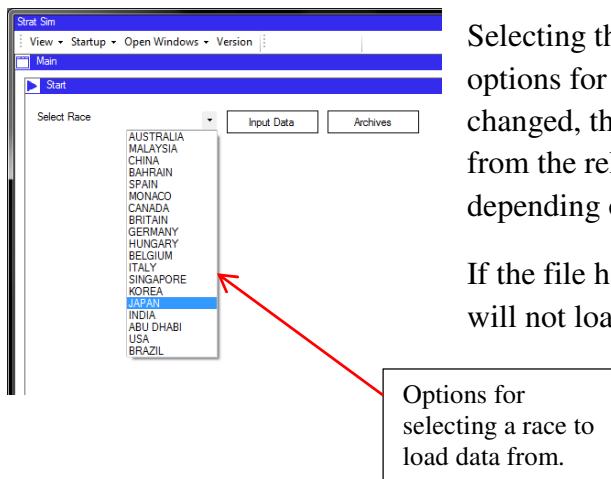
### Setting Up and Running a Race Simulation

This is the window as it is displayed when the system is first started. It appears at the full available size of the screen, so the bottom is cropped here.

There are three options, a combo box for selecting a race, a button for inputting data from the PDF files, and a button for going to the timing archives page.

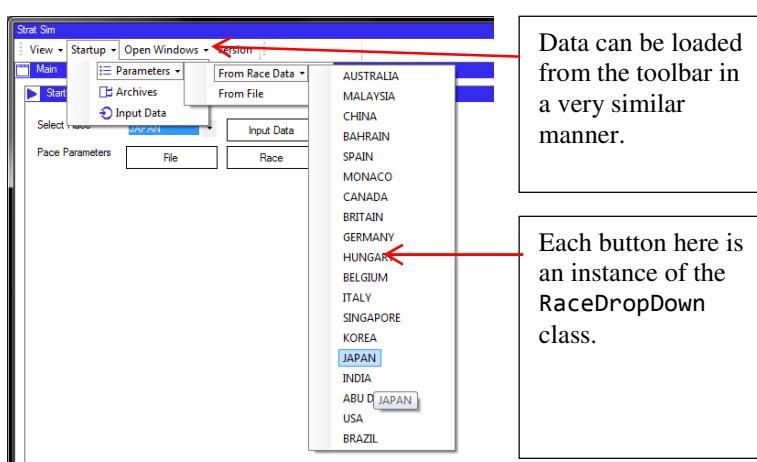
Annotated are some of the features of the system that are displayed at this stage.



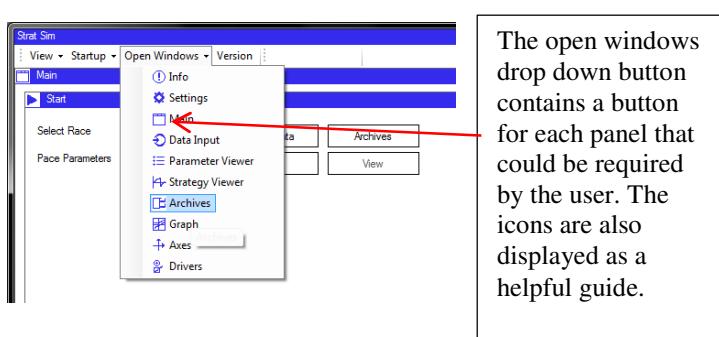


Selecting the race drop down combo box provides options for selecting a race. When the selected race is changed, the system automatically loads timing data from the relevant text file. This is ready for analysis depending on the user's next steps.

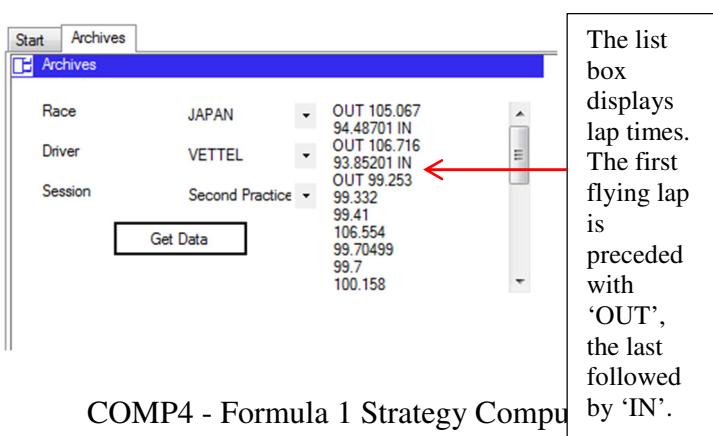
If the file has not been previously loaded, the system will not load the data but will not fail.



The toolbar's 'Startup' button contains options for starting data loading. Here, there are options for loading the pace parameters from file and from race data, and the buttons seen on the start menu are mirrored here. If the start panel is ever hidden, the functions it provides can be accessed through the toolbar.

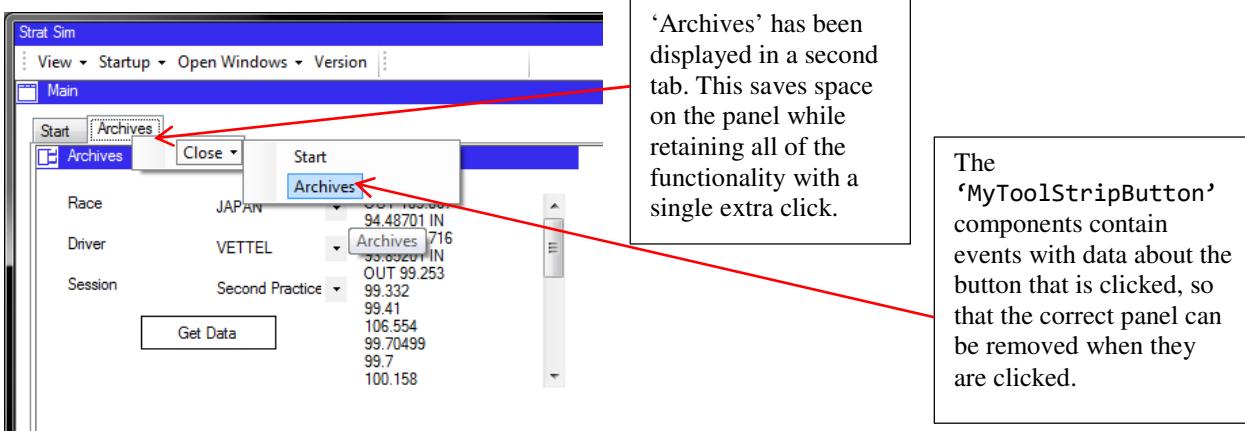


The toolbar also provides an 'Open Windows' button. The user can choose to open panels from here if they have been previously closed, or if the user wants to modify settings through panels which are not normally made visible in normal operation.



This is an instance of the TimingArchives panel. Once the race, driver, and session have been selected, and the button is clicked, the list box is populated with data from that session. This details only flying laps, with 'OUT' and

'IN' showing where pit stops have occurred.



The Main Panel is an instance of a `ContentTabControl`, which is a subclass of `MyPanel`. It can display multiple `MyPanel` content panels in a tab control for quick access while saving space. Here, two panels are displayed, so the tabs are activated. Each tab page contains a single `MyPanel`.

When added to the tab control, panels' hide, restore and close buttons are removed. To close the panel, right-clicking presents this context menu, which is an instance of `MyContextMenu`, containing `MyToolStripButton` components.

This has also been added to a new tab. The archives tab was closed.

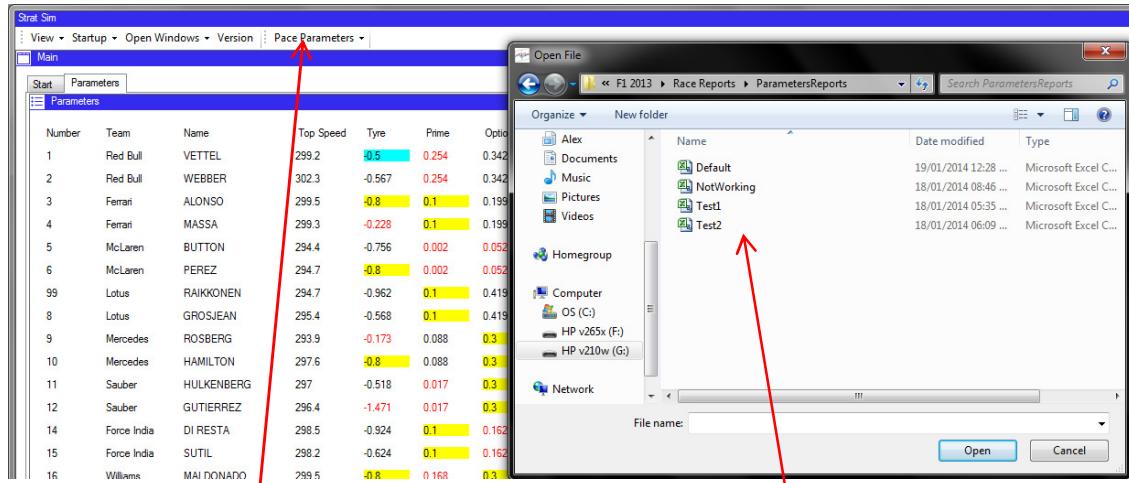
Default values are highlighted in yellow, anomalies in red text, and values that are user-modified in blue. This makes it clear what has changed and which may need changing.

To change a parameter, simply overtype the value in the text box. This triggers an update in the `PaceParameterData` class associated with the panel.

Number	Team	Name	Top Speed	Tyre	Prime	Option	Pace	Fuel Effect	Fuel Consumption	P2fuel
1	Red Bull	VETTEL	299.2	0.5	0.254	0.342	91.089	0.01	2.5	60
2	Red Bull	WEBBER	302.3	-0.567	0.254	0.342	90.915	0.02	2.5	60
3	Ferrari	ALONSO	299.5	0.8	0.1	0.199	91.665	0.031	2.5	60
4	Ferrari	MASSA	293.3	-0.228	0.1	0.199	91.378	0.031	2.5	60
5	McLaren	BUTTON	294.4	-0.756	0.002	0.052	91.227	0.035	2.5	60
6	McLaren	PEREZ	294.7	0.8	0.002	0.052	91.989	0.039	2.5	60
99	Lotus	RAIKKONEN	294.7	-0.962	0.1	0.419	91.662	0.021	2.5	60
8	Lotus	GROSJEAN	295.4	-0.568	0.1	0.419	91.365	0.021	2.5	60
9	Mercedes	ROSBERG	293.9	-0.173	0.088	0.3	91.397	0.03	2.5	60
10	Mercedes	HAMILTON	297.6	0.8	0.088	0.3	91.253	0.03	2.5	60
11	Sauber	HULKENBERG	297	-0.518	0.017	0.3	91.644	0.026	2.5	60
12	Sauber	GUTIERREZ	296.4	-1.471	0.017	0.3	92.063	0.026	2.5	60
14	Force India	DI RESTA	298.5	-0.924	0.1	0.162	91.992	0.015	2.5	60
15	Force India	SUTIL	298.2	-0.624	0.1	0.162	92.89	0.015	2.5	60
16	Williams	MALDONADO	299.5	0.8	0.168	0.3	92.093	0.033	2.5	60
17	Williams	BOTTAS	300.5	-0.004	0.168	0.3	92.013	0.033	2.5	60
18	Toro Rosso	VERGNE	295.9	-0.757	0.144	0.3	93.06	0.027	2.5	60
19	Toro Rosso	RICCIARDO	301.8	-0.962	0.144	0.3	92.485	0.027	2.5	60
20	Caterham	PIC	297.6	-0.195	0.1	0.3	94.556	0.02	2.5	60
21	Caterham	VAN DER GARDE	296.2	0.8	0.1	0.3	94.879	0.02	2.5	60
22	Marussia	BIANCHI	296.7	0.8	0.1	0.3	95	0.02	2.5	60
23	Marussia	CHILTON	301	0.8	0.1	0.3	96	0.02	2.5	60

The pace parameters panel has now been shown. This contains an array of `PaceParameterBoxes`. These boxes can fire events when the text is changed to update the values, and to validate the data entry. They also contain formatting code to display anomalies.

Each title contains a MyToolTip so that on hover, the user is provided with information about the column. The list is sorted by team.



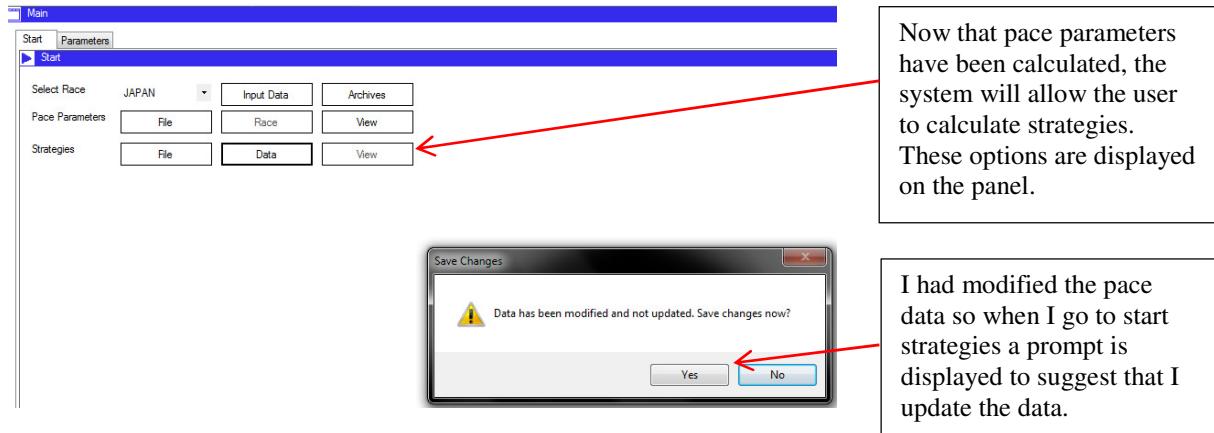
When the pace parameters are started, a toolbar button is added .This provides options for saving and loading the parameters data, and starting strategies.

Pace parameters could also be loaded from file. A file dialog has been opened here.

Pace parameter data can also be loaded from file. The system can load data from a csv file and assign this data to the drivers, and also display the data on the panel for modifications. It can also save data to a file.

Data on the panel is buffered so it is not immediately linked to the driver. This has to be done by using the 'Update' button from the toolbar.

If the user does not update the data after making changes, they are prompted to do this before the system can process strategies:

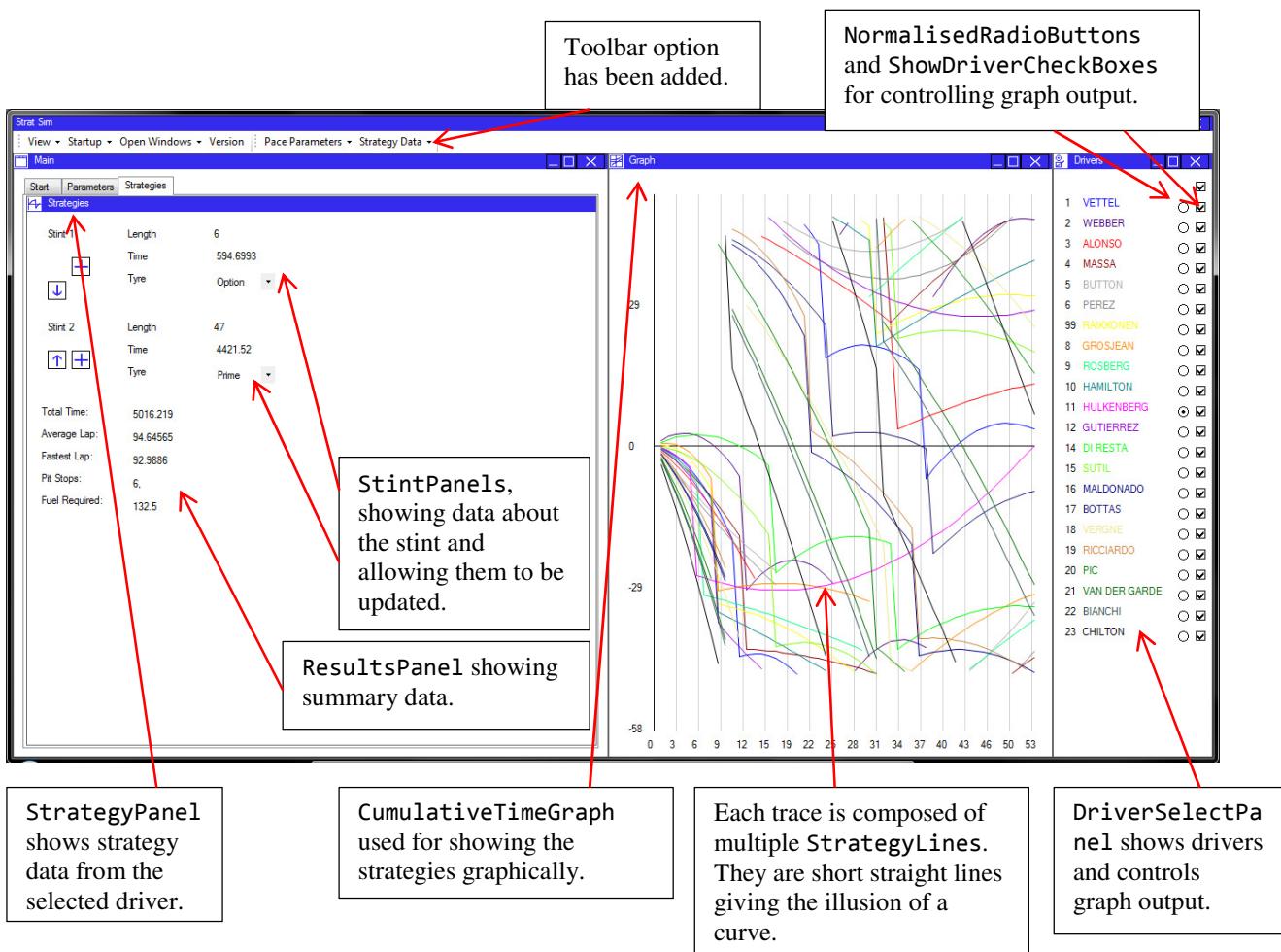


Now that pace parameters have been calculated, the system will allow the user to calculate strategies. These options are displayed on the panel.

I had modified the pace data so when I go to start strategies a prompt is displayed to suggest that I update the data.

The start panel will continue to reveal options as they become available. For example, the view button will not be enabled until the calculations have been completed. If

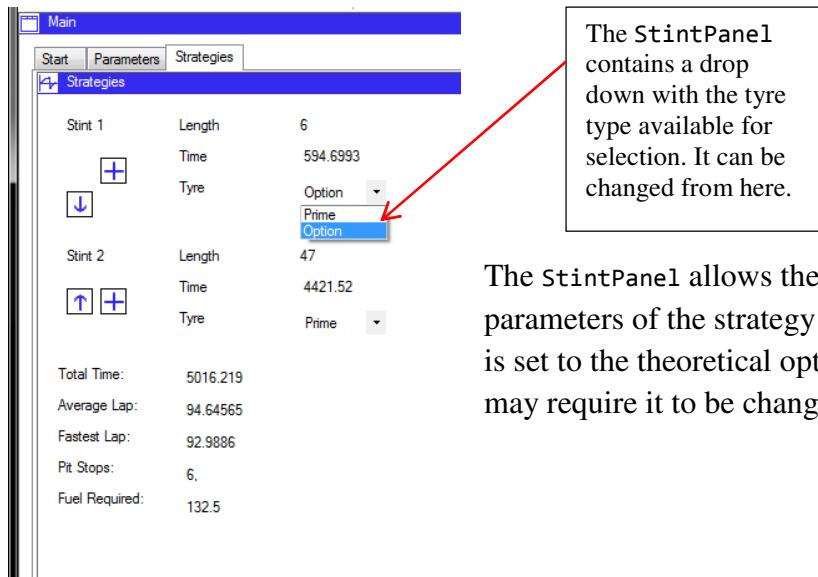
pace parameters are re-calculated, it will be hidden again until the strategies are updated.



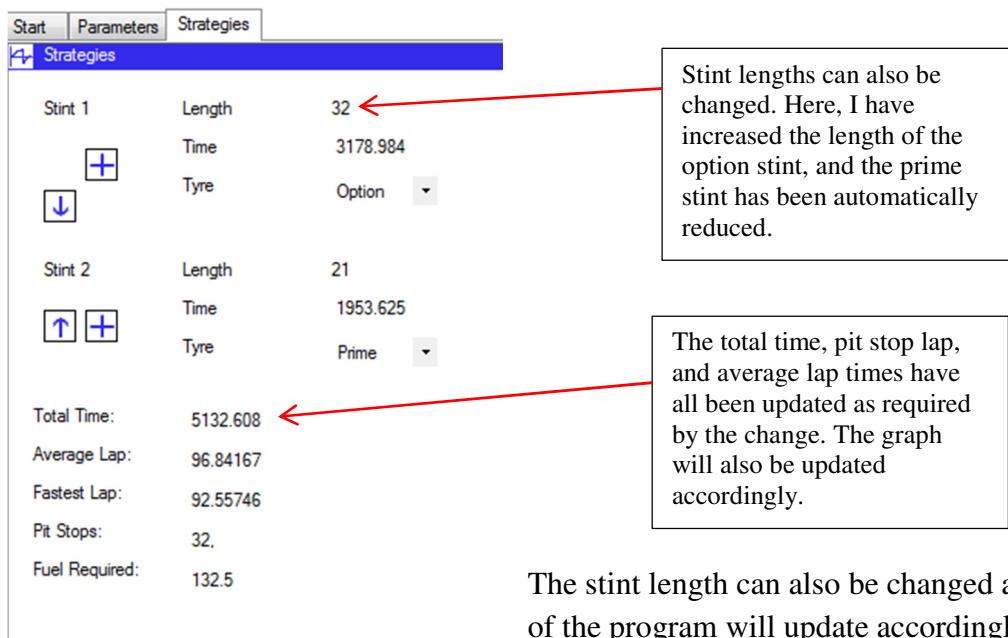
Strategies have now been calculated. The system displays a wide range of controls. The strategy data of the selected driver is shown on the left, with the recommended (optimised) stint lengths and tyre types shown. Summary data is also provided on the panel.

The graph shows the traces against each other. Right-clicking on the graph adds a pit stop to the strategy, or removes one if it is present at that location. The traces are coloured according to the driver, and are normalised on the best driver by default although this can be changed.

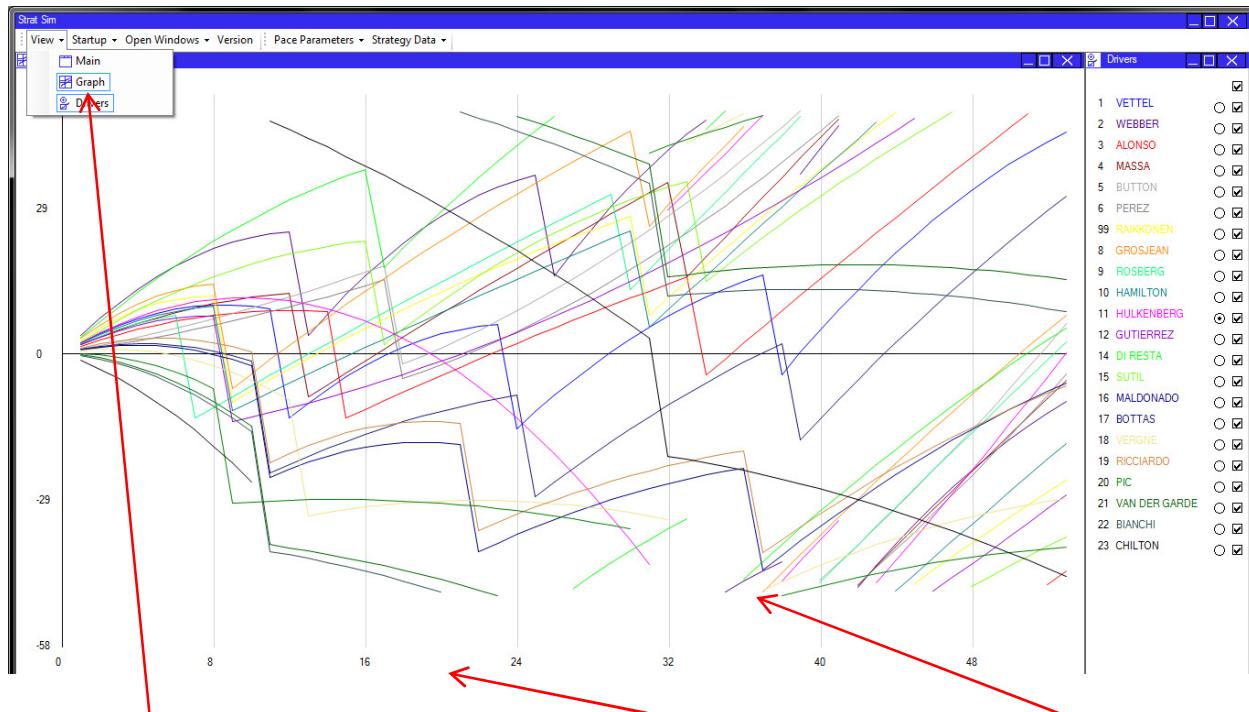
The toolbar also contains an extra option for strategy data, with similar buttons to the parameter data.



The StintPanel allows the user to change the parameters of the strategy as they wish. Although it is set to the theoretical optimum, other constraints may require it to be changed.



The stint length can also be changed and the rest of the program will update accordingly. When the strategy is updated an event is fired containing data about the strategy that has been changed. The results panel listens to this event and updates itself accordingly.

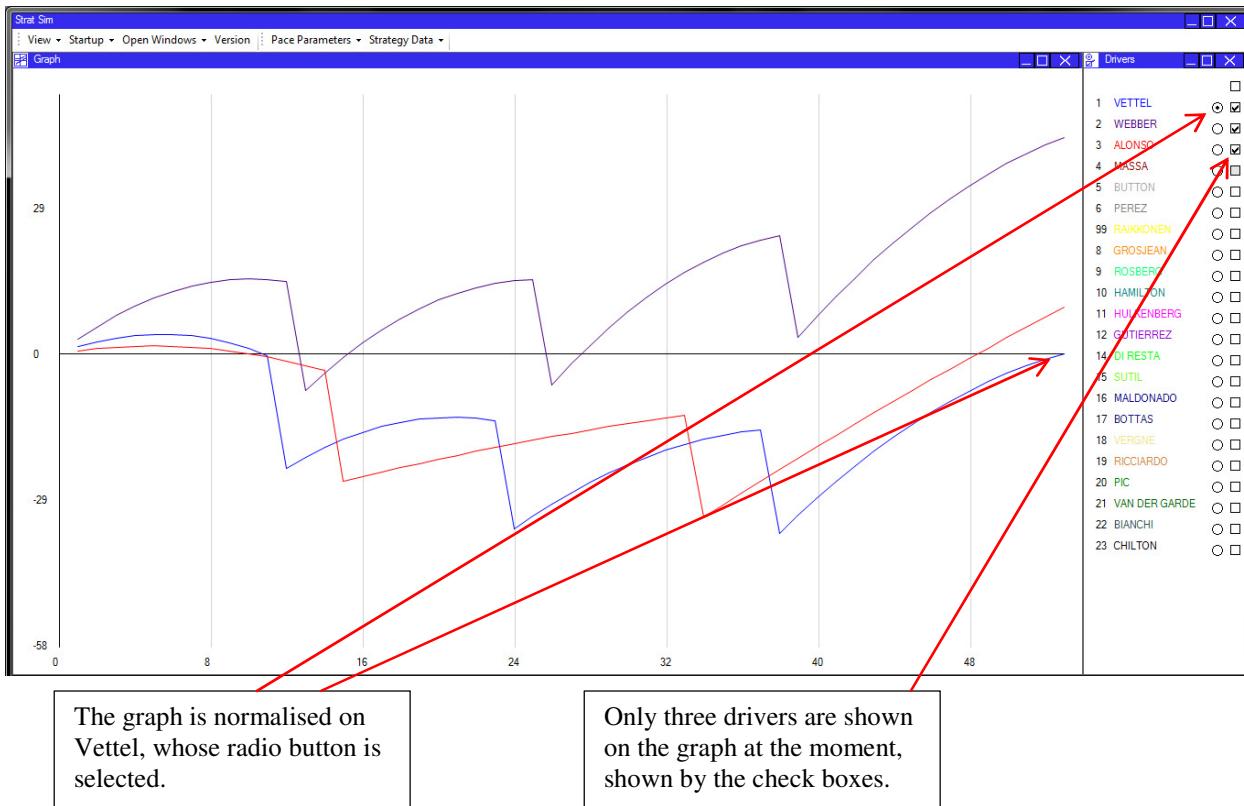


The 'View' drop down button shows a list of the panels currently on the form. Here, the main panel has been hidden so it is not selected.

The scale on the graph has updated according to the enlargement.

Note the traces cycling on the graph to represent tire track position.

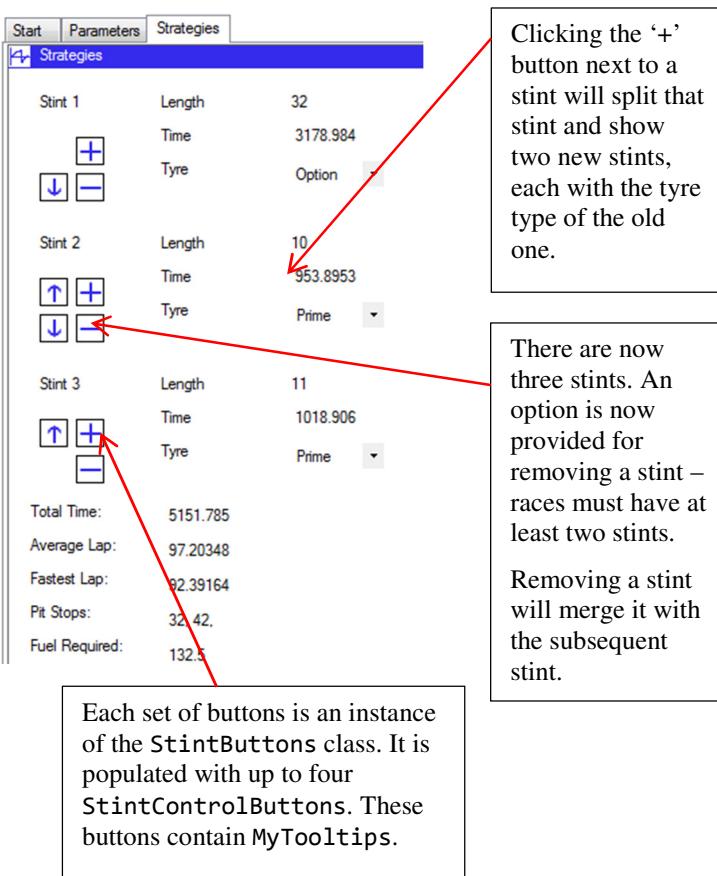
The user has the option of hiding certain panels to view other panels at a larger scale. For example, the graph here is made wider so that the same amount is viewed, just at a larger scale, after the main panel is hidden.



The traces on the graph are normalised so that they start and finish on the axis. This keeps them within the range of the graph, and gives a better idea of how track positions will change throughout the race.

It is possible to select the driver that is normalised using the radio buttons. Changing the checked radio button will change the normalised trace, and selecting the check boxes will show and hide the traces for each driver, as shown.



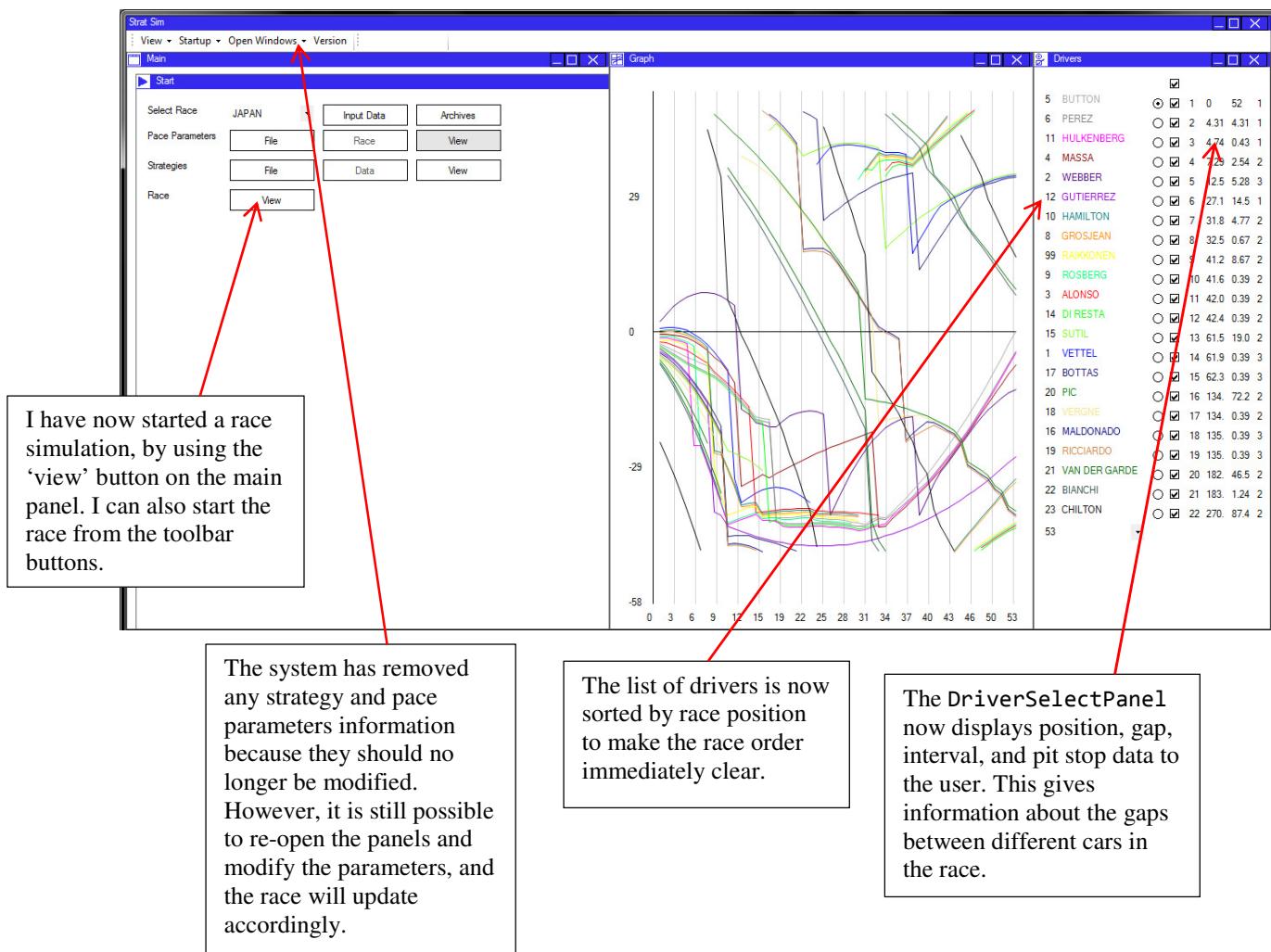


The user has the option to modify the stints on the panel. Stints can be added or removed as required to come up with the intended strategy.

This functionality is provided using `StintControlButtons` in the `StintPanels` displayed on the `StrategyPanel`.

There are now three stints. An option is now provided for removing a stint – races must have at least two stints.

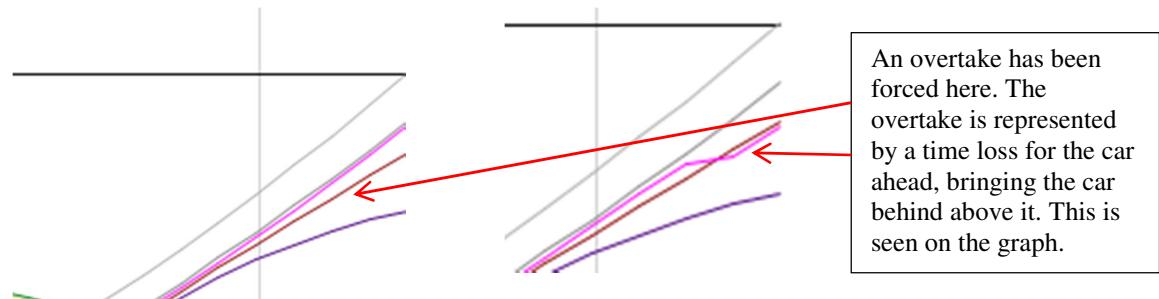
Removing a stint will merge it with the subsequent stint.



This is the window that is displayed to the user when a race simulation is run. The system displays gap and interval information and sorts the list of drivers based on race positions. The graph still behaves as before using the radio buttons and the check boxes.

The processing for the race is done in the `Race` class, which creates the traces for the graph. These are then displayed on a new instance of a `CumulativeTimeGraph`.

The system implements an overtaking model which is close to, but not always exactly equal to, reality. In this case, an overtake can be forced by the user by right-clicking on the graph.



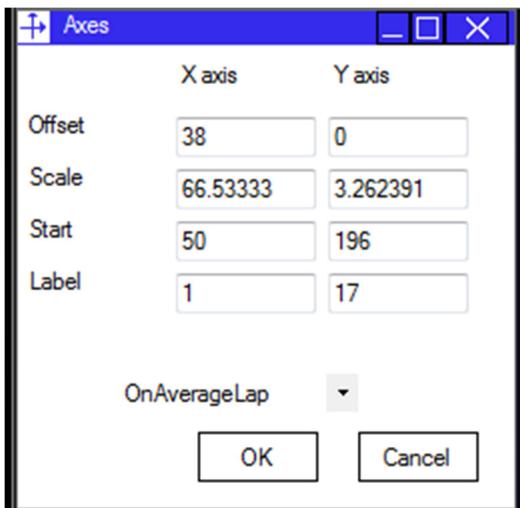
A more detailed view of the race can be achieved by using the combo box displayed in the `DriverSelectPanel`. The combo box selects a lap which is displayed in detail with an increased scale, making it very clear what is happening where on the graph. The straight nature of the `StrategyLines` can be seen clearly here.

The combo box will also set the situation displayed in the `DriverSelectPanel` to the situation as it was on that lap, giving the user up-to-date and accurate simulation information when it is required.



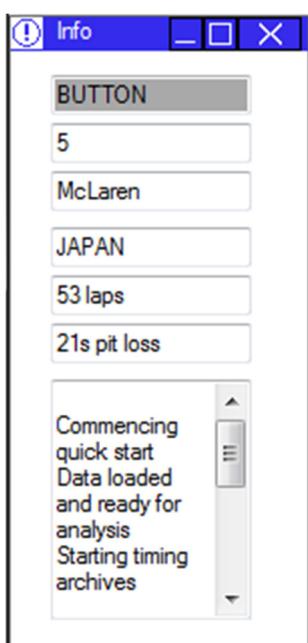
### Other Panels

The AxisPanel shows information about each axis, such as the scale factor, the start offset, and its location on the panel. The label spacing is also customisable if the user wishes to display these labels.



The normalisation type of the graph can be changed – for example it is possible to display the normalised line completely flat with all other traces shown relative to it. This is useful for working out time gaps in the race and therefore optimising a pit stop location.

Clicking OK will accept the changes and cancel will revert the changes. This panel will update when the scale is changed by resizing the panel too.

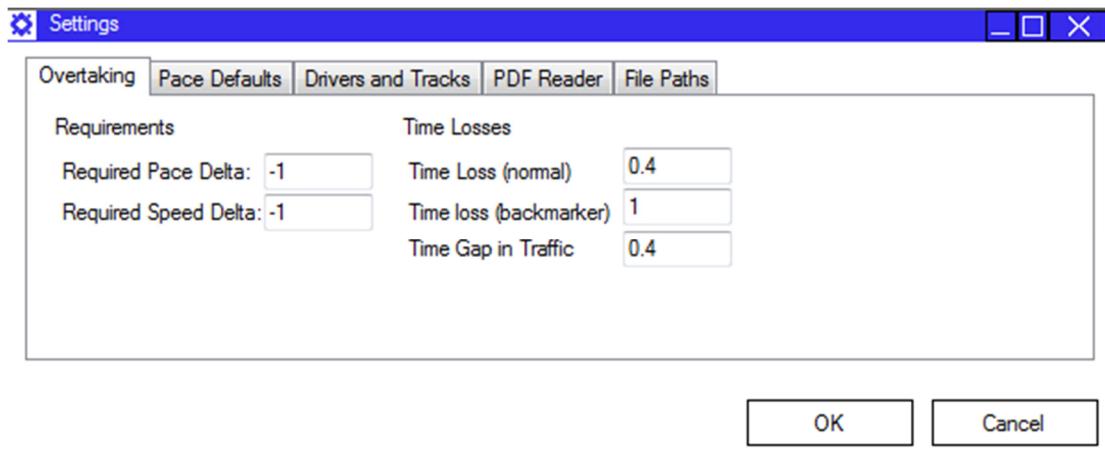


The InfoPanel displays information about the current system state. The current selected driver is displayed, along with his line colour so that this can be quickly identified on the graph. His race number and team are also displayed in case these are not immediately obvious to the user.

Track information is also displayed – here it is the number of laps in the race and the pit stop loss in seconds that are displayed. This is useful, quick access summary information for the user.

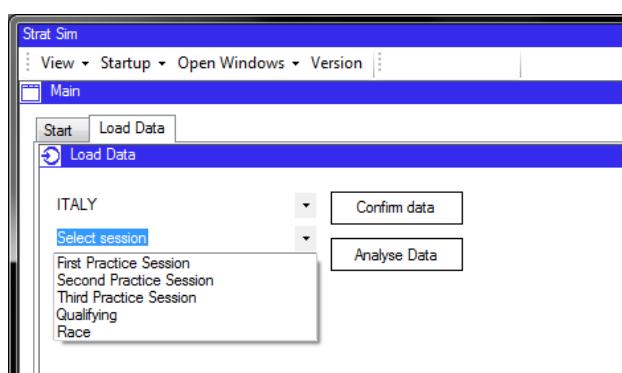
The text boxes are only displayed when a race and track are selected to save space.

The text box below contains information about what the system is processing, so that the user can track the path of previous events if they are unsure about what they have done.



The `SettingsPanel` is a large tab control. Each page contains a variety of different settings which can be modified by the user if required. The input is validated so that any incorrect values are caught when the user tries to update data.

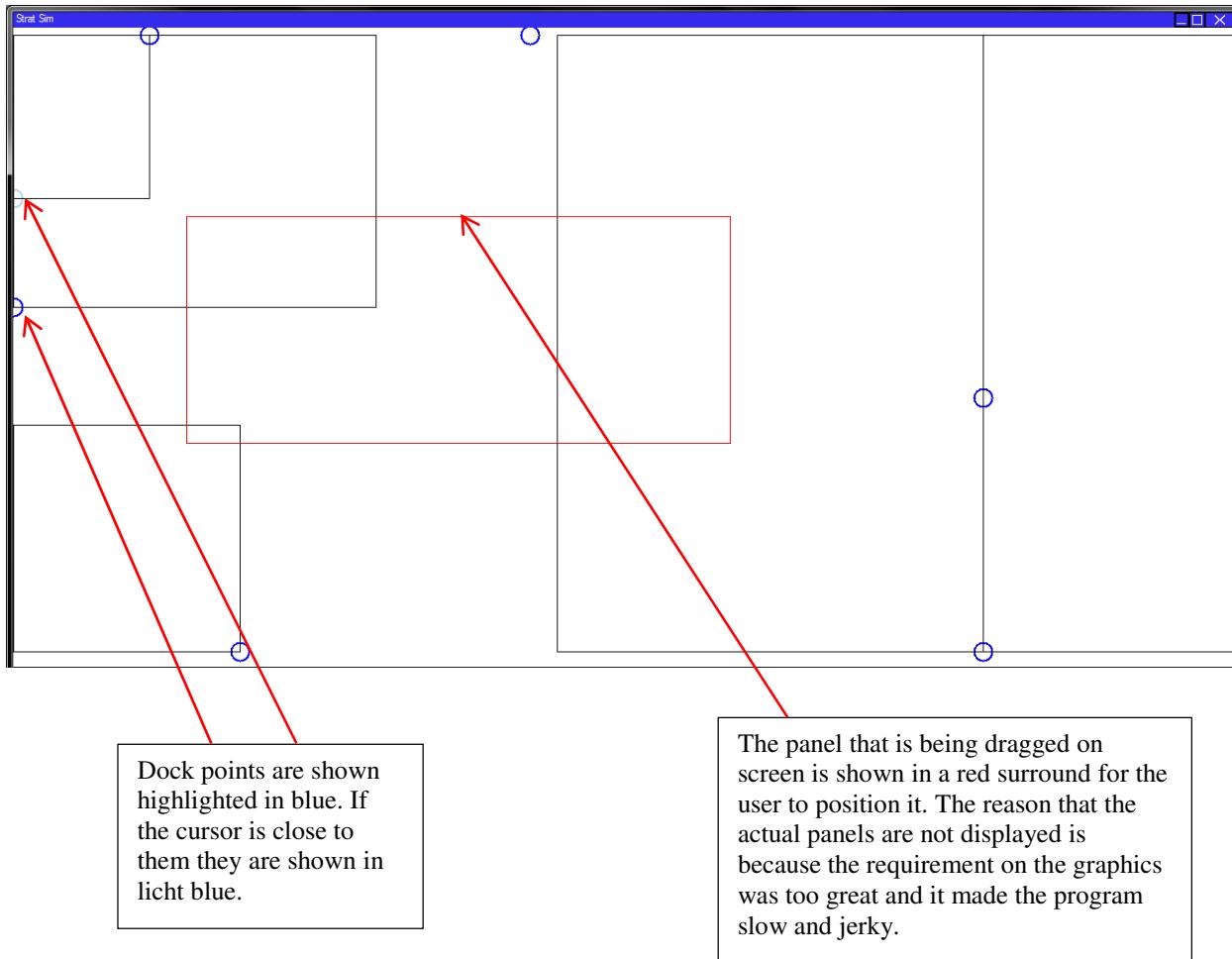
Clicking OK accepts the changes, and cancel reverts them to their original values from the settings text file. The settings data is loaded from the `Settings` class, and saved back to the `Settings` class when updated.



This is the `LoadData` content panel. It allows the user to select a race and session and load the PDF data from it, which can then be processed by the system to start strategy and race simulations.

When a race and session are loaded, the PDF will be automatically loaded using command line commands.

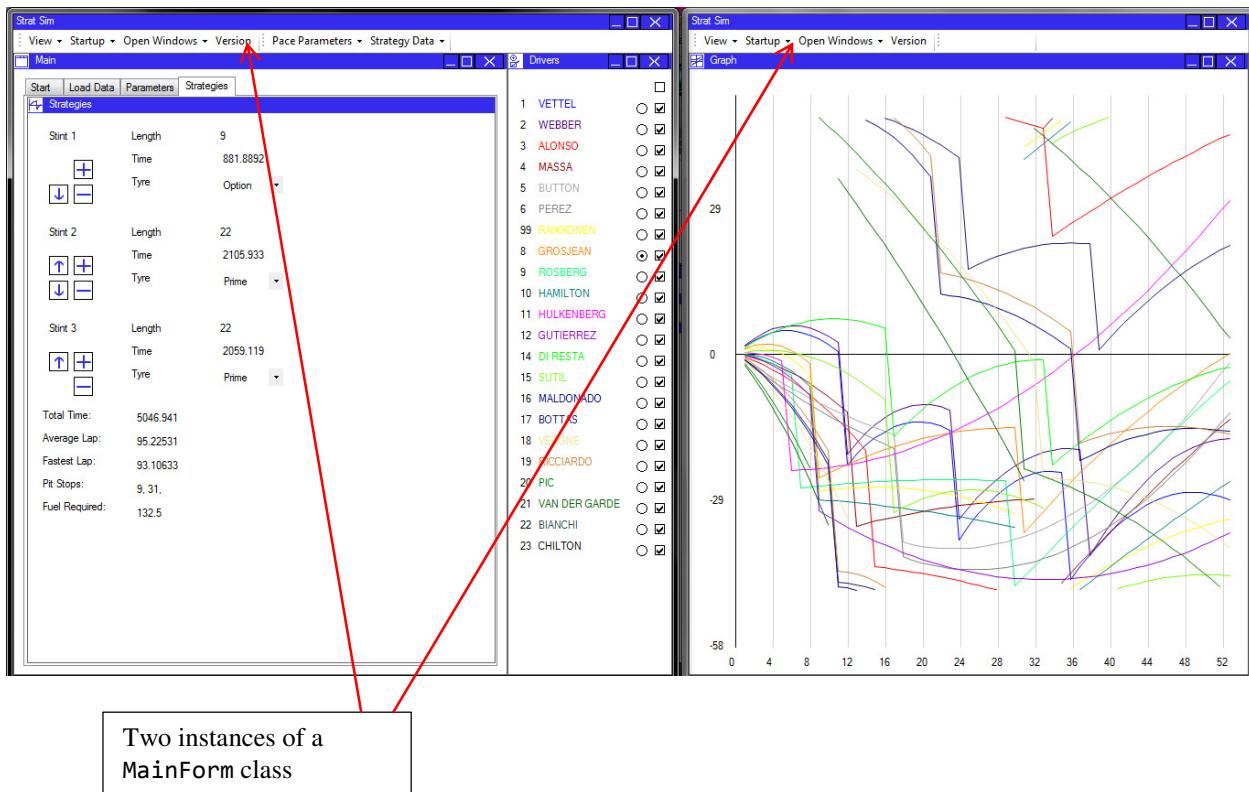
### Flow Layout



Clicking on a panel header will show this panel. This is the panel which allows the user to layout the panel dynamically.

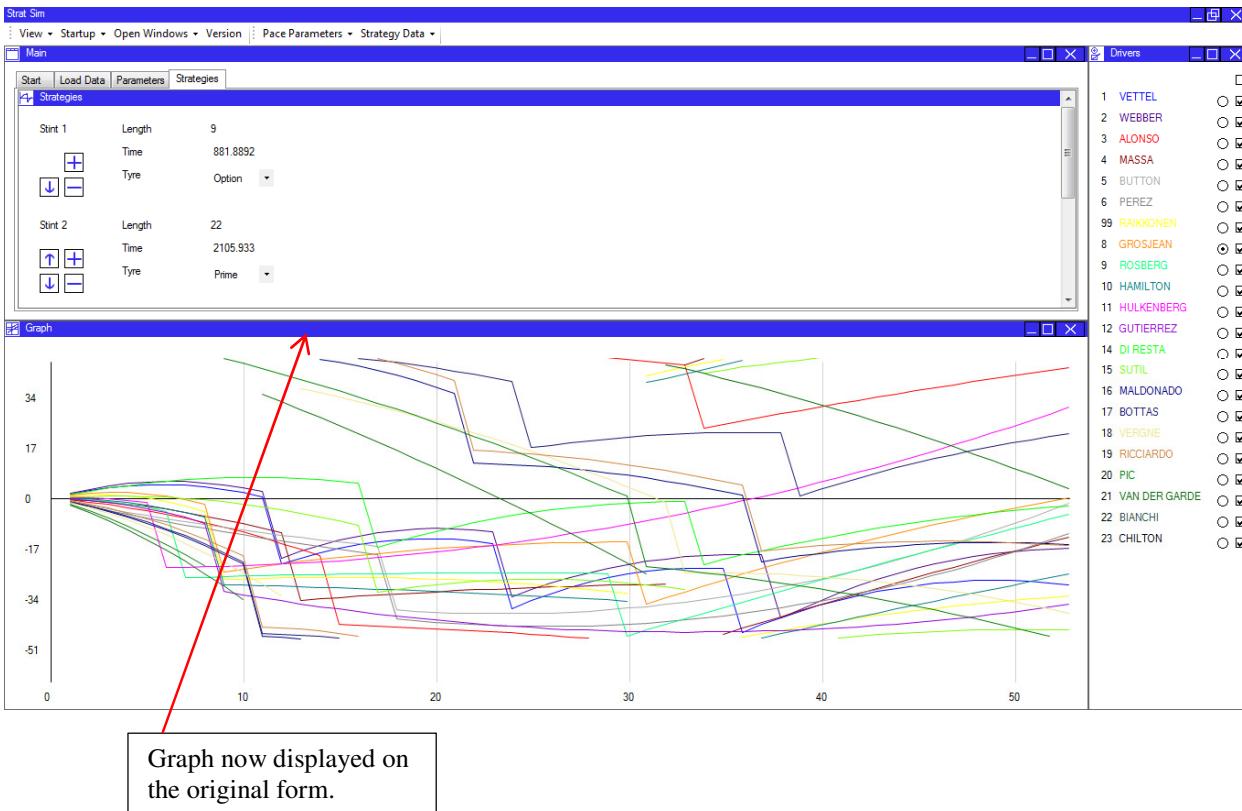
The current location of all of the panels on the form is shown by the black lines, while the panel that is being dragged is shown in red. The user can position their cursor near to one of the blue dock points to dock the panel to this location. Alternatively, if they click away from any of the dock points, the panel is opened in a new form.

The panel that is shown is an instance of the `DragDropController`.



When the panel is dropped away from any dock points, it is opened in a new form. The form is automatically sized on the form and everything remains laid out appropriately. The forms can be positioned on the screen using the standard windows OS controls.

Both forms can still communicate with each other – the graph still reacts to changes in the `DriverSelectPanel`, despite being in a different form. The new form has its own list of panels that can be loaded, hidden, shown, and moved around independently of the old form.



Panels can be moved back to the original form if desired. The system continues to function as normal – the panel is located and resized on the form and all of the controls remain linked. If the form that controls are being moved from has no controls left on it, it is automatically closed.

The graph has also been resized appropriately based on its new position and size.



The flow layout also allows the user to select the properties of the panel manually. Panels can be set to a constant size, or to autosize in any or all directions depending on the available space. They can also be set to always fill the full width or height of the screen, or be displayed at the mormal height.

The dock location of the panels can be changed through the context menu, as well as through drag-drop.

Each of the buttons displayed in the context menu is a `MyToolStripButton`, and the menu itself is a `MyContextMenu`. The menu is displayed after right-clicking the header of the panel. This functionality is removed for panels in the main tab control as it has no effect there.

## Testing

Due to the size and complexity of the program, the extensive testing is required to make sure that it meets all of the user requirements. There will also inevitably be bugs in the system which will need to be found and corrected.

By designing the test plan effectively, all aspects of the program can be checked to avoid the system crashing, breaking, or otherwise producing invalid data. If the testing exposes serious flaws in the program these will need to be corrected and any relevant tests re-run; depending on the problems this may take some time to correct the issues in the code.

A variety of different testing methods will be used, so that all sections of the program can be properly tested. These methods will be explained and a test plan for each section of tests included.

## Test plan

### 1. Testing of Data Input

#### **Start Menu Buttons**

The start menu buttons will be tested by clicking on every available one at every possible stage of the program, and making sure that the intended result occurs. This may be no result at all, because some buttons are deliberately disabled as it is known that they will generate errors.

Test No.	Program Stage	Control	Intended Result	Actual Result	Pass (y/n), Action taken	Screen shot
1.1.1	Startup	Race Select combo box	Race data is loaded and pace parameters become available	Race data is loaded and pace parameters become available	Y	
1.1.2		Data Input	The data input window is loaded and activated. Pace parameters are not displayed	Data input window is loaded and activated	Y	Figure 1.1a, 1.1b
1.1.3		Archives	The data archives window is loaded and displayed	Archives window opened	Y	
1.1.4	Race data	Race	Data from the	Data from the	Y	

	loaded	select combo box	new race is re-loaded and pace parameters redisplayed	new race is loaded		
1.1.5		Data Input	Data input window displayed but pace parameters buttons remain enabled	Data input window loaded and displayed and buttons remain available	Y	Figure 1.2
1.1.6		Archives	As in 1.1.3	Archives window is loaded	Y	
1.1.7		Pace parameters from file	File open dialog is displayed. No action taken on any controls in the start menu	File open dialog is displayed. The file button is disabled	N See note 1.1	Figure 1.3
Retest 1.1.7a		Pace parameters from file	File open dialog is displayed. No action taken on any controls in the start menu	File open dialog is displayed	Y	
1.1.8		Pace parameters from race	File open dialog is displayed. Data is loaded automatically so pace parameters button disabled and parameter view button enabled. Strategy panel also enabled.	As expected	Y	Figure 1.4
1.1.9	Pace parameter data loaded	Race select combo box	Race data reloaded and pace parameter and strategy view options disabled.	Race data re-loaded and start panel returned to initial state with pace parameters enabled.	Y	
1.1.10		Data Input	As for 1.1.5	As expected	Y	
1.1.11		Archives	As for 1.1.6	As expected	Y	
1.1.12		Pace	As for 1.1.7 or	As expected	Y	

		parameters from file/race	1.1.8 as appropriate			
1.1.13		View	Pace parameter viewer enabled and displayed. No change to buttons in start panel	Pace parameter viewer enabled and displayed.	Y	Figure 1.5
1.1.14		Strategy from file	Open file dialog displayed.	Open file dialog displayed	Y	
1.1.15		Strategy from data	Strategies loaded and strategy from data button disabled. View strategy button enabled	As expected	Y	
1.1.16	Strategies Loaded from data	Strategy view	Strategy viewer loaded and displayed. Strategy graph opened and driver select panel opened. Window is rearranged to accommodate. Race view button is shown.	Strategy viewer is loaded, graph and driver window displayed. Race button is shown.	Y	Figure 1.6
1.1.17	Strategies Loaded from file	Strategy view	As for 1.1.16.	Program crashes	N See note 1.2	
1.1.17 a	Strategies loaded from file	Strategy view	As for 1.1.16	As expected	Y	
1.1.18	Strategies loaded	View race	Graph of race is displayed, along with driver select panel, which now includes time gaps. Strategy viewer and strategy graph are removed from window.	As expected	Y	Figure 1.7

### Toolbar Buttons

The toolbar buttons are designed to have exactly the same effect as the start menu buttons so that the system can work without the start menu if this is preferred by the user. The buttons fire the same events as the start menu, so as long as some of them work, the linking to events can be proved.

As a result, I will test only a selection of buttons. They will be tested twice – once after start up, and once during the race.

Startup:

Test No.	Button	Expected Result	Actual Result	Accepted	Screen shot
1.2.1	View: Main	The main panel is hidden, revealing just the background	As expected	Y	
1.2.2	View: Main	The main panel is shown in the exact state it was in before	As expected	Y	
1.2.3	Startup: Parameters: From Race Data: Italy	Data is loaded from the race and the pace parameters window displayed	Data is loaded and the window is shown.	Y	
1.2.4	Startup: Parameters: From File	Open file dialog is shown	Open file dialog is shown	Y	Figure 1.8
1.2.5	Startup: Archives	Data archives panel is opened	Data archives panel is opened and displayed	Y	
1.2.6	Startup: Data input	The load data panel is opened and displayed	As expected	Y	
1.2.7	Open Windows: Info	Info panel is opened and displayed. The window is rearranged.	As expected	Y	Figure 1.9
1.2.8	Open Windows: Graph	No effect. Graph is not yet ready to display	As expected	Y	
1.2.9	Open Windows: Main	No effect. Main window is already displayed	As expected	Y	
1.2.10	Version	Version window	As expected	Y	

		opens in a separate form			
--	--	--------------------------	--	--	--

During Race:

Test No.	Button	Expected result	Actual Result	Accepted	Screen shot
1.2.11	View: Graph	Graph is hidden	Graph is hidden and window re-layouts accordingly	Y	
1.2.12	Startup: Race	Race is re-run and the graph updates	Graph updates showing results of new race	Y	
1.2.13	Startup: Archives	Starts the archives window and displays it. No change to the graph or driver select panel	As expected	Y	Figure 1.10
1.2.14	Open Windows: Axes	Starts the axes panel and links it to the graph	The axes panel is opened and displayed, and changes made here affect the graph	Y	Figure 1.11
1.2.15	Open Windows: Drivers	No change – the driver select panel is already shown	As expected	Y	

### Data Input

One important section of the program is to actually input data from the PDF files.

This requires several steps, each of which needs to be tested to ensure that the program works as expected. The following tests will demonstrate each stage of the data loading process to make sure it is as effective as possible.

Test No.	Test	Expected Result	Actual Result	Accepted	Screenshot
1.3.1	Selecting a race and session	User can select a race and session from combo boxes to define a PDF File	See screenshot – user can select the relevant data	Y	Figure 1.12
1.3.2	Selecting ‘BELGIUM’ and ‘First Practice Session’ from the combo boxes	The program should open the correct PDF file	The system opens the PDF file from the first practice session in Belgium	Y	
1.3.3	Copying data in the PDF and selecting ‘confirm data’	The program should load the data and then start the next PDF file.	The system loads the data and then starts the PDF file from the second practice session. See note 1.3.	Y	
1.3.4	Repeating previous tests for all PDFs from the race	The program will load all data from the PDFs.	As expected	Y	
1.3.5	Pressing ‘Analyse Data’	The program processes the data and starts the pace parameters window.	Program crashes. See note 1.4	N	
1.3.5a	Pressing ‘Analyse Data’	As for 1.3.5	As expected	Y	Figure 1.13

### Pace Parameters

This is still the data input section so testing of the calculations will come later. For now, I need to test the functionality provided to the user that allows them to change the pace parameters manually. This is done by overtyping into the text box; the value is updated when the text box is left.

The text boxes are validated with a format check and a range check, which will be checked here on two random cells. The type of testing implemented will be black box ‘BET’ testing. This uses boundary data, which is on the edge of acceptability, erroneous data, which should be rejected, and typical data, which should be accepted.

Driver number 3; Prime tyre degradation:

Prime tyre degradation is validated to be less than option tyre degradation, and greater than or equal to zero. It must also be a valid real number. In this example, option tyre degradation is 0.3.

Test No.	Type	Test Data	Expected Result	Actual Result	Accepted	Screenshot
1.4.1	B	0.3 (option degradation figure)	Rejected	Accepted	N See note 1.5	
1.4.1a	B	0.3	Rejected	Rejected	Y	
1.4.2	B	0	Rejected	Rejected	Y	Figure 1.14
1.4.3	E	Two	Rejected	Rejected	Y	
1.4.4	E	0.3	Rejected	Rejected	Y	Figure 1.15
1.4.5	E	7	Rejected	Rejected	Y	
1.4.6	E	-0.1	Rejected	Rejected	Y	
1.4.7	T	0.1	Accepted	Accepted	Y	
1.4.8	T	0.2578	Accepted	Accepted	Y	
1.4.9	T	0.005	Accepted	Accepted	Y	Figure 1.16

Driver 9; tyre delta:

The tyre delta is validated to be less than zero.

Test No.	Type	Test Data	Expected Result	Actual Result	Accepted	Screenshot
1.4.10	B	0	Rejected	Rejected	Y	
1.4.11	B	-0	Rejected	Rejected	Y	
1.4.12	E	0.5	Rejected	Rejected	Y	Figure 1.17
1.4.13	E	Hello World	Rejected	Rejected	Y	
1.4.14	T	-0.5	Accepted	Accepted	Y	
1.4.15	T	-1	Accepted	Accepted	Y	

The pace parameters panel also has an associated toolbar item, with functions for updating and resetting data, and also writing and reading to files. This function needs to be tested.

Test	Function	Expected Result	Actual Result	Accepted	Screen shot
1.5.1	Save	Open file dialog is opened. When save is clicked, the data is saved to a CSV file.	File dialog opened and data saved to file	Y	
1.5.2	Open	Open file dialog is opened and file can be selected. When file is selected, the data is loaded from file and the screen updated	As expected	Y	Figure 1.18

### Strategies

The strategies window has a number of controls, all of which can be altered by the user to change the parameters of the stint. These controls need to be tested to make sure that they all have the intended effect.

There are also, as in the parameters window, save and load functions. However, because they are programmed in the same way as for the parameters, I am satisfied that they do not require testing.

There are further ‘reset’ and ‘update’ functions, which allow the user to undo changes and to update the document with the altered parameters. These will be tested at this stage, and because they are also implemented in the pace parameters window a successful test here will be taken as proof that the pace parameters window functions also work.

For these tests, the system was loaded with the default data. A screenshot of the initial program situation and data is shown in figure 1.19. Unless otherwise specified, the data is set to the initial situation before a new test is begun, to avoid complicating the results.

Test No.	Test Details	Expected Result	Actual Result	Accepted	Screen shot
1.6.1	Overtyping 24 in the stint 1 length text box (Typical)	The first stint length is set to 24, and the second stint length is reduced to 22. The graph and other values update	As expected	Y	
1.6.2	Overtyping 10 in the stint 3 length text box (Typical)	The final stint length is set to 10, and the second stint length is increased to 25. The graph and other values update	As expected	Y	Figure 1.20
1.6.3	Overtyping 0 in the stint 2 length text box (Boundary)	The value is rejected as zero-length stints cannot be implemented.	Rejected and error message displayed. Value returns to original value	Y	
1.6.4	Overtyping 56 in the stint 1 length text box (Boundary)	The value is accepted because this still allows each of the other stints to be one lap each.	Program crashes	N See note 1.6	
1.6.4a	Overtyping 45 in the stint 1 length text box (Boundary)	The value is accepted because it still allows stint 2 to be one lap long	Accepted as expected	Y	Figure 1.21
1.6.5	Overtyping 46 in the stint 1 length text box (Boundary)	The value is rejected because this does not allow space for the other stints	Rejected and error message displayed	Y	
1.6.6	Typing 15.4 into the stint 3 length text box. (Erroneous)	The value is rejected and returned to its original value	As expected	Y	
1.6.7	Typing “Hello World” in to the stint 2 length text box (Erroneous)	The value is rejected and returned to its original value	As expected	Y	
1.6.8	Pressing the ‘stint up’ button for stint 3	Stint 2 and 3 are swapped over, with stint 2 becoming length 12 and option tyre. The graph updates	As expected	Y	

1.6.9	After 1.6.8, pressing the ‘stint down’ button for stint 2	Stint 2 and 3 are re-swapped and stint 3 becomes length 12 and option tyre. The graph updates.	As expected	Y	
1.6.10	Pressing the ‘remove stint’ button for stint 1	Stint 2 takes on the length of stint 1 and the strategy is displayed with only two stints.	As expected	Y	
1.6.11	Immediately after 1.7.0, trying to remove stint 1 in the same way	There is no button to remove the stint so the action cannot be completed. This would lead to an illegal strategy.	As expected	Y	Figure 1.21
1.6.12	Pressing the ‘add stint’ button for stint 3	Stint 3 is split into two six-lap stints which are both done on the option tyre	As expected	Y	
1.6.13	Right-clicking on the graph away from a pit stop	A pit stop is added to the driver’s strategy at this location; the graph and panels update	As expected	Y	
1.6.14	Right-clicking on the graph at the location of a pit stop	A pit stop is removed from the driver’s strategy at this location and the stints are merged	As expected	Y	
1.6.15	Using the ‘reset’ toolbar option after making a change to stint 3	Stint 3 is reset to its original status and the graph updates accordingly	As expected	Y	
1.6.16	Using the ‘update’ toolbar option after making a change to stint 3, and then repeating 1.6.15	Stint 3 is modified and remains modified when ‘update’ is clicked. When ‘reset’ is clicked, it returns to the state it was in when ‘update’ was pressed	As expected	Y	
1.6.17	Using the ‘Start Race’ option after making a change but not clicking ‘update’.	An error message appears suggesting to the user that they should update the data now before beginning the race simulation	An error message appears	Y	Figure 1.2.2

### Driver select panel

The driver select panel is started during strategy processing and expanded during race processing. It allows the user to select and control the display of traces on the graph, and also allows for fast navigation to an enlarged point in the race.

I will test the functionality of the controls in the window so that I can be sure the user can operate the graph as intended without any unintended consequences for the program.

For these tests, a race was run on Japan so that the results can be more clearly seen.

The tests all run sequentially, following on from the system's state after the last test, unless otherwise stated.

Test No.	Test	Expected Result	Actual Result	Accepted	Screen shot
1.7.1	Clicking the radio button next to Hulkemberg's name on the panel	The graph is normalised on Hulkemberg's trace, which becomes straight and horizontal	As expected	Y	
1.7.1	Clicking the 'show all' check box at the top of the panel	All of the traces on the graph are hidden	As expected	Y	
1.7.2	Clicking the check box next to Vettel on the driver select panel	Only Vettel's trace is shown, and it becomes the normalised trace.	As expected. Vettel's trace is displayed and normalised	Y	
1.7.3	Clicking check boxes next to further drivers: Webber, Alonso and Massa	The extra traces are shown but with no changes to the normalisation	As expected	Y	
1.7.4	Clicking the radio button next to Button, whose trace is not currently shown	No effect. The system should not normalise on a hidden driver	There is a small flicker but there is no change to the display of the program	Y	
1.7.5	Clicking the check box next to Vettel's name, who is currently the normalised driver	Vettel's trace should be hidden and the next fastest driver selected for normalisation	The program hides Vettel's trace and normalises on Massa, who was the next fastest driver.	Y	Figure 1.24

1.7.6	Clicking the ‘show all’ check box again.	All traces should be shown but the normalised driver should remain the same: Massa.	All traces are shown and Massa remains the normalised driver	Y	
1.7.7	Selecting lap 40 from the lap select combo box	The program zooms in to show lap 40 on the screen, with a small buffer around the lap to view what is going on around. The gaps and intervals update to this lap	The graph updates and the gaps and intervals are sorted and updated.	Y	Figure 1.25
1.7.8	Selecting lap 1 from the lap select combo box	The program zooms in on lap 1, but because this is close to the start of the race a range of 10 laps is shown instead of 15.	The graph updates and the gaps and intervals are changed	Y	
1.7.9	Right clicking on the race graph during the race simulation	No effect as the user can no longer edit strategies	As expected	Y	

### Axes Panel

The axes for the graph can be modified through an external panel. The axes have to be validated in case an incorrect value is entered, so I need to adequately test data input into the axes text boxes. The combo box is ‘select from list only’ so it does not require validation.

For this test, I will use the horizontal axis scale text box. If this is correct, I will be satisfied that the rest of the boxes work effectively. BET testing is used again

Test No	Test Data	Expected Result	Actual Result	Accepted	Screen shot
1.8.1	-5 (E)	Rejected	Rejected	Y	
1.8.2	0 (E)	Rejected	Rejected	Y	
1.8.3	“Two” (E)	Rejected	Rejected	Y	
1.8.4	0.01 (B)	Accepted	Accepted	Y	
1.8.5	8 (T)	Accepted	Accepted	Y	
1.8.6	100(T)	Accepted	Accepted	Y	

### Data Archives

The data archives works directly with files that are accessible and can be edited outside of the scope of the program. As a result there is the potential for files to be missing, corrupted, or incorrectly named. There needs to be significant validation of these files and file inputs to protect the program.

The archives use a similar method to the data input for file handling, so I will trust that if the system works in this situation it will be robust for all other situations.

Test No	Test	Expected Result	Actual Result	Accepted	Screen shot
1.9.1	Search for Japan, Vettel, First Practice session	Lap times displayed in list box. The lap times should match those found in the corresponding PDF	Lap times displayed. See screenshot for comparison with PDF.	Y	Figure 1.26
1.9.2	Search for Japan, Webber, First Practice Session	The lap times box is cleared and replaced with times from Webber's session	Lap times cleared and displayed as expected	Y	
1.9.3	Search for Japan, Chilton, First Practice Session	Chilton did not take part in this session so no lap times should be displayed.	No lap times are displayed. The continues to run properly	Y	
1.9.4	Search for Malaysia, Vettel, Qualifying	No data or folder exists for these files, so the system should not display any lap times	Lap times are displayed for Vettel's qualifying at Japan; the system did not register the change of race.	N See note 1.7	
1.9.4a	Complete 1.9.1 and then repeat 1.9.4	No data exists so the system should not display any lap times	As expected	Y	Figure 1.27
1.9.5	Loading intentionally corrupted data	The data has been modified to change the format. The system may fail to load it but should not crash.	No lap times are displayed but the system does not crash	Y	

I am now satisfied that I have tested as many of the inputs as necessary to prove that validation is in place and works as expected. All of my protocols and procedures are working as expected and the code has been updated where necessary to fix issues and remove bugs.

The next section of the program to be tested is the testing against the specification, which is important to ensure that all of the intended features are present and that it will meet the user's requirements.

## 2. Specification Testing

This section can be largely subjective. The tests will sometimes ask if a feature is present or not, sometimes specify a test that it must pass, or sometimes describe a calculation which would itself need to be tested.

The requirements for these tests will be listed and tests designed to ensure that the requirements are fulfilled. I will then execute the tests and make note of anything that is not present. This will be checked with the client, as will the results of the tests to make sure he is satisfied with the way that the tests have been conducted and that the features are all present.

### **Requirements and tests**

Test No.	Section	Requirement	Test
2.1.1	Format	The solution will be automated in program code It will be programmed in C# It will be a forms application	See note 2.1
2.1.2		The solution will be bespoke It will be specific to Formula 1 It will be designed to work specifically for my end user's requirements	See note 2.1
2.1.3		The solution will use object oriented programming	See note 2.1
2.1.4		The user will be able to work interactively with the system.	See note 2.1
2.2.1	Data Input	The system will allow the user to select an FIA PDF file that is saved locally based on the selection of a race weekend, session details, and data type through combo boxes.	In the data input window, select a race and session from the combo boxes.
2.2.2		The system will then open the selected timing data.	Selecting race and session data from the combo boxes.
2.2.3		The user can copy data in the PDF.	Copy data from the PDF that is opened.
2.2.4		The system will read the copied text from the clipboard	Click 'confirm data' in the data input window to

Test No.	Section	Requirement	Test
		To file, to save data that is already To a routine to process the data loaded.	start data processing.
2.2.5		The system will read settings data from file to establish Default values for parameters Infrequently changed parameters Track names and data Driver names	Change the default tyre delta in the file to -0.6 and then start the program and the settings panel.
2.2.6		The system will take user inputs to fine-tune or define parameters used for the simulation.	Overtype -0.8 into the settings panel and click ok.
2.2.7		The user will be able to right-click on certain objects to insert a pit stop	See tests 1.6.13 – 1.6.14
2.3.1	Archives	The system will save data from the input sessions to CSV files, which can be accessed at any time to view past lap time data.	Load data from PDFs. Then load the CSV files from the folder structure.
2.3.2		This data will be processed to remove the unnecessary data found in the PDF files.	Data from the CSV files should be in the format <time (float, 3dp)>, <lap status (int)>
2.3.3		The CSV files should be in a format that can be read by Microsoft Excel, for analysis.	See tests 2.3.1-2.3.2
2.3.4		The output of data will be tabulated for ease of viewing.	Start data archives and view the archives for Vettel, from Belgium FP2
2.3.5		Data should not be able to be corrupted during viewing.	Overtype '100.1' into the box containing lap times.
2.4.1	Pace Parameters	The system will analyse the timing data to define parameters about a driver's pace. These include: Top Speed Base lap pace Option tyre delta Tyre degradation (prime and option) Fuel effect	See Data Input Unit Testing.
2.4.2		The system will save the pace parameters data in race and session specific files, which can be used for data analysis.	Use the 'save data' option in the pace parameters window.
2.4.3		The system will set to default values any data which cannot be loaded from the timing data.	Run the pace parameters window and check for default values.

Test No.	Section	Requirement	Test
2.4.4		The system will allow the user to edit these values in case they have more accurate information.	See test series 1.4
2.5.1	Strategies	<p>The system will choose the optimum number of stops, the tyre choice, and the laps on which to pit, based on:</p> <ul style="list-style-type: none"> <li>i. Automated algorithms for calculating the optimum</li> <li>ii. The number of laps in a race</li> <li>iii. The pace parameters</li> <li>iv. The pit stop loss</li> <li>v. The regulations, which are taken into account in the code</li> </ul>	See Strategy Unit Testing
2.5.2		The system will set this strategy as the default for this driver	See Strategy Unit Testing
2.5.3		The user will be able to change the strategy that has been selected as the default.	See test series 1.6
2.5.4		<p>Once a race has been simulated, the strategy for one driver can be optimised taking into account other drivers</p> <p>This will use iterative methods.</p>	See note 2.1
2.5.5		The user will also be able to manually adjust strategies	See test series 1.6
2.6.1	Race Simulation	The system will put all of the drivers' strategies together in a race.	Click the race simulation button
2.6.2a		The system will run a lap-by-lap simulation of the race based on the defined pace parameters for each car.	Inspect code to make sure the simulation is lap-by-lap
2.6.2b		If pit stops or traffic are encountered, the event will be logged.	Run a race simulation and check that the pit stops are inserted in the correct location
2.6.2c		At the end of the lap, all events will be processed, causing a time loss for the car as well as (potentially) other effects.	See Race Unit Testing
2.6.3		The simulation will include:	
2.6.3a		Pit stops	See Race Unit Testing
2.6.3b		Overtaking (and the effects of 'traffic')	Inspect graph for overtaking and traffic
2.6.3c		Tyre degradation and delta effects	Inspect graph for effects of degradation and delta
2.6.3c		Passing lapped cars	Inspect graph for effects

Test No.	Section	Requirement	Test
			of lapped cars
2.6.3e		The tyre 'cliff'	See note 2.2
2.6.4		The simulation will not include The effects of weather Safety cars The effect of DRS Track evolution through a race	See note 2.1
2.6.5		The results of the race simulation will be displayed graphically, with the cumulative time (normalised on any selected driver's average lap time) plotted against number of laps for each driver.	Click the race simulation button and show the graph
2.6.6		Overtaking will be simulated using a probabilistic approach	See Race Unit Testing
2.6.7		Each driver will have a different coloured trace on the graph.	Show the graph to check for different colours
2.6.8		The graph will have adjustable axes to allow the user to change the detail of the view.	See test series 1.8
2.6.9		The strategy can be changed from the same window as the graph to see the effects visually.	See test series 1.7
2.7.1	Settings	Settings need to be alterable but do not need to be altered within the running program.	See tests 2.2.4 – 2.2.5
2.7.1		The units should be in line with FIA standards: i. Speed: kmh <sup>-1</sup> ii. Time: s iii. Fuel weight: kg iv. Fuel consumption: kg.lap <sup>-1</sup>	Start the pace parameters and PDF files and check that the units and values are consistent.
2.7.3		Time is increasing in the positive direction	Start the pace parameters and make sure that the degradation is greater than zero, and the tyre delta is less than zero.
2.8.1	Processing	The system will implement interfaces to link together different classes	Visual check of code to indicate the presence of interfaces
2.8.2		The system will be able to adapt to new regulation changes in future years.	See note 2.2
2.8.3		The solution will run on Windows 7 PCs	Run the system on a windows 7 PC
2.8.4		The solution will take up less than the 8GB of RAM available on the PCs used by the race team.	Start race simulation and then start windows task manager

Test No.	Section	Requirement	Test
2.8.5		There is no need for encryption or data security.	See note 2.1
2.9.1	Output	The system will be displayed in one complete window.	Start the race simulation and check that only one window has been opened.
2.9.2		This window will scale with the size of the screen	Start the program on two different sized screens and show that the window re-sizes appropriately.
2.9.3		The window will contain tab controls to viewing different elements	Start the archives panel and check that the tab control displays the items
2.9.4		The system will use simple graphics to prevent the display from becoming jerky.	Start a race simulation and move the mouse quickly across the screen.
2.9.5		Additional windows will be used for the timing archives and version information.	Click the version information button and check that it opens in a new window.

**Tests and results**

Test No.	Test Description	Expected Result	Actual Result	Accepted	Screen shot
2.2.1	In the data input window, select a race and session from the combo boxes.	The system automatically finds the available data types and selects one	As expected	Y	
2.2.2	Selecting race and session data from the combo boxes.	The system will open a PDF reader and the selected file	As expected	Y	Figure 2.1
2.2.3	Copy data from the PDF that is opened.	The text is copied to the clipboard	As expected	Y	
2.2.4	Click 'confirm data' in the data input window to start data processing.	The text is processed by the program and the driver's times are updated.	As expected	Y	
2.2.5	Change the default tyre delta in the file to -0.6 and then start the program and the settings panel.	The settings panel opens and displays -0.6 in the tyre delta section.	The panel opens with -0.6 loaded.	Y	Figure 2.2
2.2.6	Overtype -0.8 into the settings panel and click ok.	The settings panel updates and all references in the program update. The text file is also updated.	As expected	Y	
2.3.1	Load data from PDFs. Then load the CSV files from the folder structure.	The CSV files open in the correct format in Microsoft Excel	As expected	Y	Figure 2.3
2.3.2	Data from the CSV files should be in the format <time (float, 3dp)>, <lap status (int)>	The first column is the time to three decimal places and the second column is an integer	As expected	Y	Figure 2.3
2.3.4	Start data archives and view the archives for Vettel, from Belgium FP2	The data appears in a straightforward, chronological list for easy viewing	As expected	Y	
2.3.5	Overtype '100.1'	The data is read	The data is in a	Y	

Test No.	Test Description	Expected Result	Actual Result	Accepted	Screen shot
	into the box containing lap times.	only to prevent it from being corrupted	list box which is read only.		
2.4.2	Use the ‘save data’ option in the pace parameters window.	A save file dialog is opened and the file can be saved as CSV anywhere on the PC or network	As expected	Y	
2.4.3	Run the pace parameters window and check for default values.	There are several default values shown where data cannot be processed.	As expected, default values are shown in several locations.	Y	Figure 2.4
2.6.1	Click the race simulation button	A graph is shown with multiple traces run simultaneously and interacting with each other	As expected	Y	
2.6.2a	Inspect code to make sure the simulation is lap-by-lap	The code for ‘simulate race’ includes a for-loop to iterate through each lap	As expected	Y	
2.6.2b	Run a race simulation and check that the pit stops are inserted in the correct location	Vettel, who pits on lap 21, should have a much longer 21 <sup>st</sup> lap	As expected. His graph trace drops when this occurs	Y	Figure 2.5
2.6.3b	Inspect graph for overtaking and traffic	The graph shows evidence of one driver being stuck behind another, and of overtakes occurring.	As expected. It is difficult to find a good example but the screenshot shows all of the required information	Y	Figure 2.6
2.6.3c	Inspect graph for effects of degradation and delta	The graph will show curves of different initial and rate of change of gradient, demonstrating different pace and tyre deltas.	The graph begins with a shallow gradient and gradual curve. After a pit stop, the line becomes steep, because the option tyre is faster.	Y	
2.6.3c	Inspect graph for effects of lapped cars	The graph will show evidence of a slower car losing	As expected.	Y	Figure 2.6

Test No.	Test Description	Expected Result	Actual Result	Accepted	Screen shot
		time as it is passed by a faster car.			
2.6.5	Click the race simulation button and show the graph	A graph of cumulative time is shown, which is increasing downwards. Pit stops go downwards on the graph. The trace starts and finishes on the axis.	As expected	Y	
2.6.7	Show the graph to check for different colours	Each trace on the graph is shown in a different colour to allow them to be distinguished from each other.	The colours are also displayed in the info panel and the driver select panel.	Y	Figure 2.6
2.7.1	Start the pace parameters and PDF files and check that the units and values are consistent.	Speed should be in the 300s, in kph, and times should be in the 80s-90s, in seconds.	As expected. Degradations are also reasonable values in seconds.	Y	
2.7.3	Start the pace parameters and make sure that the degradation is greater than zero, and the tyre delta is less than zero.	Degradation should be positive as it will cause lap time to increase. Tyre delta should be negative as it will cause lap time to decrease.	As expected. The values are also validated as such on user input.	Y	Figure 2.4
2.8.1	Visual check of code to indicate the presence of interfaces	The code will include interfaces specifying required fields and methods.	The code includes two interfaces for file handling classes.	Y	
2.8.3	Run the system on a windows 7 PC	The system will function correctly	The system was developed on windows 7 and runs perfectly.		
2.8.4	Start race simulation and then start windows task manager	The system will take up only a small amount of main memory	The system runs on at most 20,000k, which is well within required bounds.	Y	
2.9.1	Start the race simulation and check that only	Only one window is displayed, although the user	As expected	Y	

Test No.	Test Description	Expected Result	Actual Result	Accepted	Screen shot
	one window has been opened.	can elect to display two			
2.9.2	Start the program on two different sized screens and show that the window re-sizes appropriately.	The window and its contents re-size appropriately	The window is displayed effectively on a small screen and does not crash or cause issues.	Y	Figure 2.7
2.9.3	Start the archives panel.	The archives panel should open in a tab control next to the start panel.	The archives panel opens and is displayed in a tab control.	Y	
2.9.4	Start a race simulation and move the mouse quickly across the screen.	The mouse moves smoothly and does not hold or pause	As expected. The graphics are simple and quick to process and draw	Y	
2.9.5	Click the version information button and check that it opens in a new window.	A new window is opened so that the version information can be displayed.	As expected	Y	

### 3. Window 'Flow Layout' Testing

In order to display all of the controls dynamically on the form, I have developed my own flow layout logic for the panels. The layout has several features which require testing to ensure that the program works as expected.

Test No.	Feature	Test	Expected Result	Actual Result	Accepted	Screen shot
3.1.1	Layout of panels	Load a race simulation	All panels are displayed with no overlap and there is no free space on the screen.	As expected; see screenshot	Y	Figure 3.1
3.1.2		Start the axes and settings panels	The panels re-size to the maximum available size with no overlaps.	As expected	Y	
3.1.3		Re-size the main window	The panels are scaled and re-laid appropriately without overlapping	As expected	Y	
3.1.4		Minimise and then restore the main window	The panels return to the locations they were in before the restore	As expected	Y	
3.2.1	Hiding and showing of the panels	Press the minimise button on the graph window	The graph window is hidden and all other panels re-size around it	Windows re-size appropriately	Y	
3.2.2		Press the view: graph button in the toolbar	The graph is re-shown and the window returns to the layout from before the action	The graph is shown and the panels re-size accordingly.	Y	Figure 3.2
3.2.3		Press the view: graph button again	The graph is hidden and panels re-size	As expected	Y	

Test No.	Feature	Test	Expected Result	Actual Result	Accepted	Screen shot
3.3.1	Full screen toggle of the panels	Press the full screen toggle button on the graph panel	The graph goes to the full size of the screen. The button also changes to show the restore down icon	As expected	Y	
3.3.2		Press the full screen toggle button on the maximised panel	The graph returns to its original size and the button changes back to its original icon.	As expected	Y	
3.4.1	Dynamic panel reordering	Click the header of the graph panel	The screen is blanked with outlines of the panels shown. The selected panel moves with the mouse	As expected	Y	Figure 3.3
3.4.2		Click near the 'top left' dock point	The panel is docked to the top left of the window and the panes are displayed	As expected	Y	
3.4.3		Click away from any of the dock points	A new window is opened with the panel selected displayed within it.	As expected	Y	
3.4.4		With two windows open, activate the re-layout function and click the background of the original window.	The selected panel is moved to the original window. The second window will close if it has no further panels to display.	As expected	Y	
3.5.1	Panel dock locations	Right-click the header of the driver panel and	The panel is docked to the top left of the screen and all	As expected	Y	Figure 3.4

Test No.	Feature	Test	Expected Result	Actual Result	Accepted	Screen shot
		select the 'top left' dock type	other panels re-size around it. The new dock type is highlighted in the menu.			
3.5.2		Right-click the header of the driver panel and re-select the top right dock type.	The panel is returned to its original layout and the top left option is selected in the context menu.	The panel is returned and re-sizes appropriately.	Y	
3.6.1	Panel autosize types	Change the autosize type of the graph to constant	The graph reduces to a fixed size and the window rearranges around it. No changes in the window layout should occur.	The panel is re-sized.	Y	
3.6.2		Change the autosize type of the settings panel to 'free'	The settings panel becomes wider to fill the available space. There should be no panel gaps at the bottom of the window.	The settings panel is resized as expected.	Y	Figure 3.5
3.7.1	Panel fill types	Change the fill style of the main panel to 'full width'	The main panel stretches to fill the width of the screen. All other panels fit around it.	As expected	Y	

#### 4. Unit Testing

Microsoft Visual Studio provides functionality for creating ‘test methods’. This uses a framework ‘MSTest’, which is used in industry to run large numbers of tests on a system.

Test methods can be run without requiring the program to be debugged. They can also be stored in a different project, with the initial project being stored as a reference. This means that when the program is compiled and distributed, it does not need to include the test methods.

The test methods work on a simple and repeatable structure:

- Arrange: Set up all required variables and define data sets using normal variable initialisations. The data sets can be linked from databases or read from files. This includes defining the expected result of the test.
- Act: Call one or more functions within the program using the parameters that were arranged. Save the output of the function or check the variables for evidence of a successful function output.
- Assert: Use the Assert class to check if the intended result matches the expected result. This will allow MSTest to output a ‘pass’ or ‘fail’ message.
- 

Multiple tests can be written and executed at the same time so that the system can be continually tested after any updates to make sure they have not changed the behaviour of the system.

I have written a series of unit tests designed to confirm the behaviour of some of the important routines in the system. These were used in some of the specification tests seen above. The specific details of the tests in this section are shown below.

Due to the repeatability of these tests within code, i.e. they can be automated to test a large variety of input parameters, it is possible to execute white-box testing. This is where every possible path through the code is tested. I have used it in some situations here. It requires testing every limit in loops, every possible outcome in if-statements, and every possible range of input parameters. This requires knowledge of the code but is a good way of ensuring 100% reliability on important routines.

#### **Data Input Unit Testing**

Test No.	Method Tested	Intended Function	Test Description	Assertion
4.1.1	<code>void SetDriverSpeed(     string     lineContainingData)</code>	Sets the top speed of a driver to a value specified within the string passed to the function.	Hard codes a sample line of text to pass to the file and calls the function. Also hard codes a value for the top speed based on the line of text.	Checks if the intended driver’s top speed has been set to the value defined in the string.
4.1.2	<code>float GetLapTime(string</code>	Converts a string	Hard codes a sample	Checks if the

Test No.	Method Tested	Intended Function	Test Description	Assertion
	<code>ng time)</code>	containing a time to a floating point value for time.	string containing a lap time and a value for the time in seconds that is represented by the string.	returned value converted from the string is the same as the hard coded expected result.
4.1.3	<code>void CalculateStrategyParameters(in t raceIndex)</code>	Calculates and sets the pace parameters for each driver from the timing data that has been loaded from files.	PDF files were analysed by hand to find specific data values. These were hard coded as expected results. The test loads the files, processes the files, and calls the function to process the data.	Checks that the hard coded data matches the data calculated from the files. A sample is tested.

### Pit Stop Unit Testing

Test No.	Method Tested	Intended Function	Test Description	Assertion
4.2.1	<code>void setupGrid()</code>	Sorts the strategies used for the race in ascending order of pace, simulating qualifying	Loads a test set of strategies from files and calls the method on these strategies.	Tests that the pace for each driver is less than that of the subsequent driver, proving that the array is sorted.
4.2.2	<code>void updatePositions (ref Strategy[] strategies, List&lt;PitStop&gt; pitStops, int trackIndex)</code>	Updates the order of an array of strategies when a pit stop occurs in one or more of them.	This is an example of a white box test. There are several test methods for the one function. Mock strategies are started with defined times, and then the drivers that are supposed to pit are hard coded. The method is called to update the positions due to the pit stops.	Tests that the pit stops are in the correct order (user defined within the test, depending on the input parameters).

### Strategy Unit Testing

The strategy optimisation algorithm forms the core of the program. It is vital therefore that it works perfectly and efficiently in every situation.

In order to test the algorithm, I have developed an Excel spreadsheet that will complete the same calculation. The spreadsheet contains a collection of data sets, each of which will be tested in the system to make sure that the output is correct. The data sets are created by defining constant pace parameters and race data, and by changing the number of stints and stints on each tyre. Each of the strategies will be processed by the system and the one which takes the least total time should be selected.

The excel system was created and the data sets run to produce the expected results. The expected results are integer stint lengths, which are the optimal lengths for the prime and option tyre. These data sets were then linked into a database referenced within the test project. When the tests are called, the data is pulled from the database.

Test data is shown in the tables below. This has come directly from the excel spreadsheets.

Parameters – these are the same for each strategy:

Prime Deg	Option Deg	Delta	Laps	Pit Loss	Fuel Effect	Fuel Consumption	Pace
0.1	0.3	-0.8	58	21	0.02	2.5	200

Data Sets – the prime and option stints are the independent variables. All other values are calculated. The ‘total added time’ column is used to compare the strategies – the strategy with the lowest value is the one that should be used.

The ‘race time’ column displays the total time required to complete the race. This is what is displayed to the user so is a convenient value to check.

Data Set	Prime Stints	Option Stints	Ideal Prime Laps	Act Prime Laps	Act Option Laps	Deg Time	Pit Time	Fuel Time	Total Added Time	Race Time
1	1	1	41.25	41	17	109.2	21	85.55	215.75	11862.15
2	1	2	31.20	31	13.5	75.525	42	85.55	203.075	11849.48
3	2	1	23.57	23	12	60.8	42	85.55	188.35	11834.75
4	2	2	19.50	19	10	45.2	63	85.55	193.75	11840.15
5	1	3	24.50	25	11	53.1	63	85.55	201.65	11848.05
6	3	1	16.50	16	10	41.5	63	85.55	190.05	11836.45

The strategy unit test hard codes the parameters and sets up a database connection to the database containing these data sets. It is a data-driven test. The system iterates through each data set in the database and calls the strategy optimisation method for each data set. This calculates the ideal prime stint length. The prime stint length is then rounded to an integer, done by another method because it is difficult to know whether to round up or down. Finally, the option stint length is calculated. These

values are compared with the expected values, also loaded from the database, to check if the test is passed.

In summary, the first test takes every permutation of prime and option stints in a race and checks that the system produces the correct stint length for these values.

The strategy with the lowest time must also be assigned to the driver correctly, and for this a second test was written. It checks all of the strategies for the one with the lowest added time, and sets this strategy as the driver's default.

Visual inspection of the data sets shows that data set 3 is the fastest strategy, so it is checked that the tyre type and stint lengths match that of the third data set.

The table below officially documents the tests.

Test No.	Method Tested	Intended Function	Test Description	Assertion
4.3.1	<code>void SetStintLengths (int lapsInRace, int noOfPrimeStints, int noOfOptionStints)</code>	Sets the prime and option stint lengths for a strategy to the optimum possible value.	Defines input parameters from data sets and calls the method to calculate the optimum stint lengths.	Tests that the output stint lengths are equal to the predicted stint lengths from the data sets.
4.3.2	<code>Strategy OptimiseStrategy(Driver driver, int trackIndex)</code>	Returns the optimum strategy for a driver to complete a race.	Defines input parameters, for example pace parameters, for a given driver. The method is called to optimise the driver's strategy based on his pace parameters.	Checks that for every stint in the driver's selected race strategy, the stint length and tyre type are the same as the one expected from the data sets. This proves equivalency of the strategies.

## Validation Unit Testing

The validation routines that I have coded need to be tested. They are normally the first port of call on a user input, so they could have almost anything thrown at them, and therefore need to work in every situation.

The best way to guarantee this is through white box testing. To achieve this I will inspect the code and make sure that every possible route through all of the routines is covered by a test.

The way that the validation routines are structured, where only the ‘equal to’, ‘less than or equal to’, and ‘greater than or equal to’ routines are actually coded (the rest are simply combinations of these other inequalities), means that this is quite easy to do. The routine that checks if a value is between two boundaries, exclusive, will call all of the programmed validation routines except the ‘validate between inclusive’ method. This will be tested separately. By testing just these two routines I can call every method and therefore check every line of code works properly.

To test every possibility, I need to complete boundary, erroneous, and typical tests. This will ensure that every route through the code is tested – every inequality is checked and every line should be executed at one point or another.

The routine below shows an example method, annotated to show how each statement is tested.

```
public static float validateGreaterThanOrEqualTo(float value, float notLessThan,
string field, ref bool incorrectValue, bool showDialog)
{
    if (!incorrectValue) ← 1
    {
        incorrectValue = (value < notLessThan); ← 2
        if (incorrectValue && showDialog) ← 3
        {
            string warningMessage = "The field " + field + "
cannot be less than " + Convert.ToString(notLessThan);
            var warning = MessageBox.Show(warningMessage,
"Incorrect Value", MessageBoxButtons.OK, MessageBoxIcon.Warning);
        }
    }
    return value; ← 5
}
```

1. In the ‘validate between inclusive’ method, the value is checked for the upper bound before it is checked for the lower bound. If the value fails on the upper bound test, ‘incorrectValue’ will be set to true, so the structure will not execute: see test 4.4.5. If, however, the value is typical, it will pass the upper bound test and ‘incorrectValue’ will be false, so the code within the structure will execute: see test 4.4.4. This expression has been tested for every eventuality.

2. In the ‘validateBetweenInclusive’ method, this statement will be executed for any value which passes the upper bound check. To check every eventuality for this assignment requires a boundary, erroneous, and typical check on the lower bound; in other words, a test where ‘value’ is equal to, greater than, and less than, ‘notLessThan’ – see tests 4.4.9, 4.4.6, and 4.4.8 respectively.
3. To test this expression, I have to check the eventuality in which it will pass and fail. In the unit tests, ‘showDialog’ is always false, which will cause the inequality to be false. In the user input testing, ‘showDialog’ is always true, and erroneous tests on the lower bound will cause the statement to be true. Tests 1.4.2 and 1.6.3 demonstrate this occurring; figure 1.14 shows the error message being correctly displayed. This has now been tested for every eventuality.
4. See figure 1.14. This proves that the error message is displayed successfully.
5. The unit tests do not assign the value to any variable. However, the user input testing assigns the value to a variable if the value is valid. The typical user input tests show examples of the user input being assigned to the variable – see test 1.6.4a and figure 1.21 for examples.

The unit tests also double as integration testing, by proving that all of the required methods are called and work together in the intended way. For example, the ‘validateBetweenInclusive’ method will check the upper bound and subsequently the lower bound.

If the value fails either test, ‘incorrectValue’ must be returned true. I must not, therefore, have the second test overwrite the first test. Equally, if the first check is passed, the second test should still be called. By testing both the upper and lower bounds in these tests, I can show that the tests are called in the correct order.

See the table below for proper documentation of the tests. All tests are done with `upperBound = 1`, `lowerBound = 0`, and `showDialog = false`. The assertion checks `incorrectValue` for either false or true as specified, and all tests are passed.

Test No.	Method Tested	Value Tested	Test Type (BET)	Assertion
4.4.1	<code>float validateBetweenInclusive (float value, float lowerBound, float upperBound, string field, ref bool incorrectValue, bool showDialog)</code>	0.5	T	False
4.4.2		1.5	E	True
4.4.3		-2	E	True
4.4.4		0	B	False
4.4.5		1	B	False
4.4.6	<code>float validateBetweenExclusive (float value, float lowerBound, float upperBound, string field, ref bool incorrectValue, bool showDialog)</code>	0.5	T	False
4.4.7		1.5	E	True
4.4.8		-2	E	True
4.4.9		0	B	True
4.4.10		1	B	True

It is proper to note that the validation routines seen here, and copied into the implementation, are not quite identical to those developed in the Pseudocode section. This is because the testing has shown that there were some flaws in routines of that structure, so they have been updated in order to pass all of the tests.

## Notes

### 1. User Input

#### **1.1. Code changes to avoid hiding the pace parameters from file button:**

The code was modified to remove the command that hid the file button. This occurred under the OnLoad event, not the ‘Loaded’ event, as it should have been. This new code was written and implemented for the file and race options, as both behaved in similar ways. Once the code was moved, a retest proved that the system functioned properly.

It was later discovered that a similar issue occurred in the strategies section, and this was fixed in exactly the same way.

#### **1.2. Changes to prevent program crash when file input is cancelled.**

It was discovered that the program fired the ‘StrategiesLoaded’ event after displaying the dialog box, not after the data was actually loaded. The point at which the event was fired was changed.

A subsequent re-test also showed that if file input was cancelled, the load strategies from data button stayed disabled. This was traced to an error in the implementation of note 1, where the wrong variable was referenced. This was corrected and a retest proved the functionality worked correctly.

#### **1.3: Loading subsequent PDF files:**

The system will only load the next PDF file if the previous one has been closed. This is due to the limitations on accessibility of the command line. If the file is not opened automatically, closing the previous file and then re-selecting the correct session will open the file.

In the case of qualifying, where there could be two data sets to load, the system will only open a PDF when the data type has been selected from the third combo box.

#### **1.4: Changes due to program crash after files loaded.**

When the ‘analyse data’ button was pressed previously, the program crashed. This was traced to an issue in the data collection process, where the driver number, retrieved from the PDF files, was incorrectly converted to a driver index, used for labelling and recording the data. This resulted in an index out of range exception.

The routine was corrected and the program now returns the correct value in all cases. A retest showed that the program is now correct.

#### **1.5: Changes to validation procedure to ensure that system does not accept invalid data entry in pace parameters.**

The validation functionality was present but was not implemented. I added code to the constructors for the pace parameter text boxes to set the upper and lower bounds for each, and made sure that when the text was set it was validated. If an incorrect value was entered, the system will return to its original value.

Once the functionality was properly implemented, the remainder of the tests prove its effectiveness.

### **1.6: Changes to stint length validation to prevent system crashes:**

The system currently removes length from the preceding or subsequent stint when the length of a stint is updated. In the event of a boundary test here, the system failed, because the subsequent stint did not have sufficient length to accommodate the change in length.

I could either change the stint length change functionality to take length from further stints if necessary, or I could validate the stint to have a length that can be taken from the stints directly nearby.

I chose the second option because the former would require a lot of programming, which for the benefit of the feature is not worthwhile. It was implemented and re-tested and the program did not crash. While the upper bound of the stint length has now been reduced, every combination of stint lengths is still possible as long as later stints lengths are changed before early ones by the user.

### **1.7: Changes to data archives to allow for changing race tracks:**

The system currently only overwrites the data that is stored and presented to the user in archives. If there was no data to overwrite the previously recalled data, the system would re-display the data that was previously recalled.

The solution was to re-set all of the driver's data when the race index was changed. This was also a useful thing to implement so that when the race index was changed in the start panel, effectively signifying the start of a new race simulation, all of the data was reset automatically.

A retest shows that the program now works as intended.

## 2. Specification Testing

### **2.1: Tests not required:**

There are some requirements in the specification which are unnecessary to test as they are fundamental to the program. As a result they are not included in the test plan.

### **2.2: Functions not implemented:**

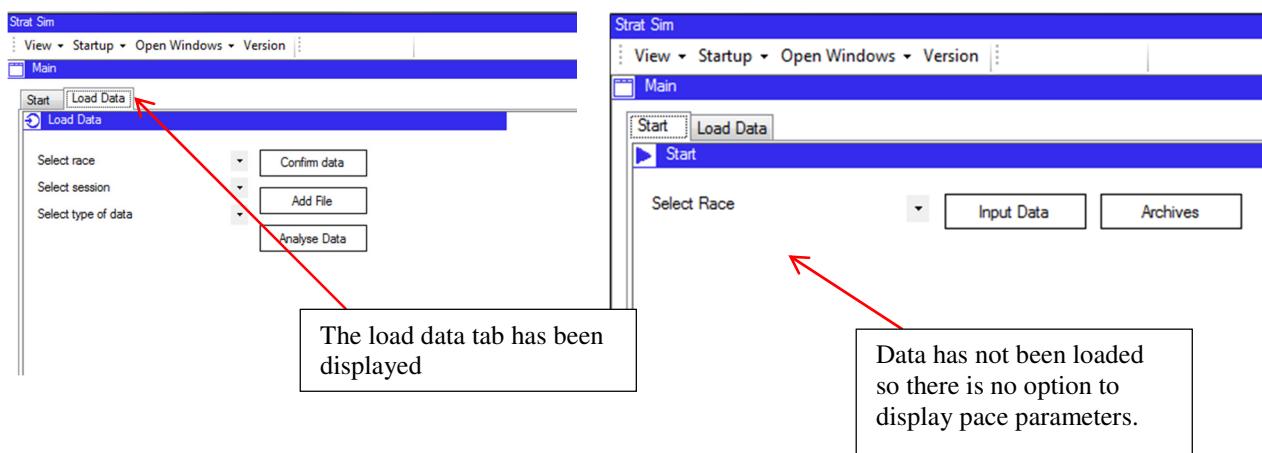
During the course of development it is known that these functions have not been implemented. This has been recorded and will be developed further in the future developments section. There is no point in testing for these features as they are not present.

## Hard copy evidence

### 1. User Input Testing

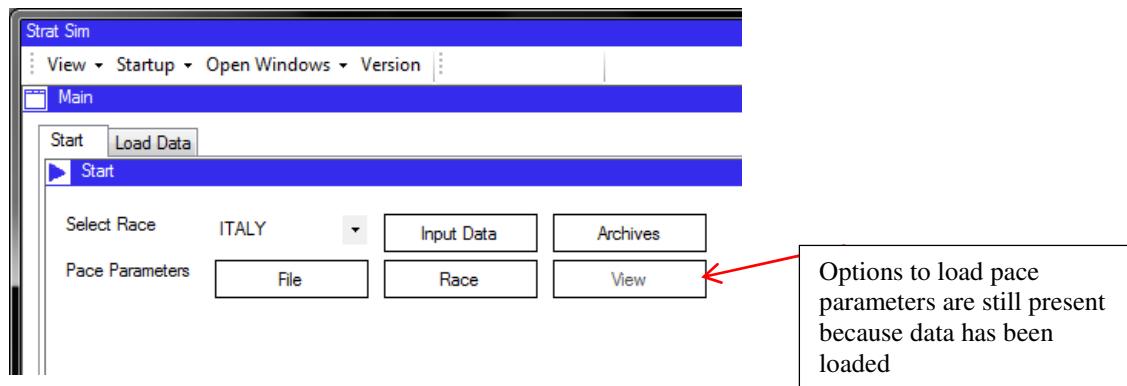
Test 1.1.2: Data Input is initiated. The data input window is shown and pace parameters remain hidden.

Figure 1.1a, 1.1b:



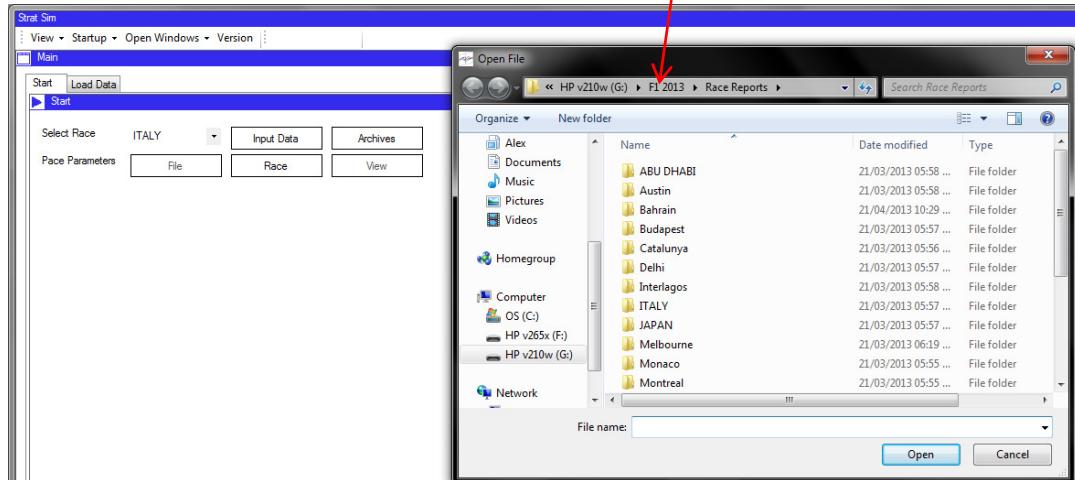
Test 1.1.5: Data input initiated after race combo box has been selected. The pace parameters controls remain enabled.

Figure 1.2:



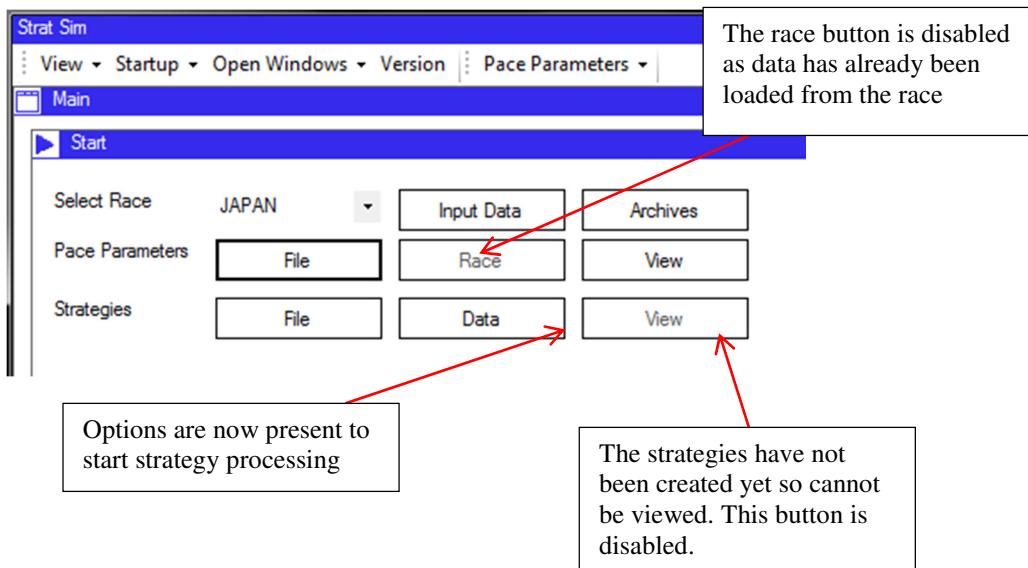
Test 1.1.7: Pace parameters load from file is selected; the program displays a file open window.

Figure 1.3:



Test 1.1.8: Pace parameters from race button is clicked. It is disabled and the view button enabled, as the data has become available for viewing.

Figure 1.4:



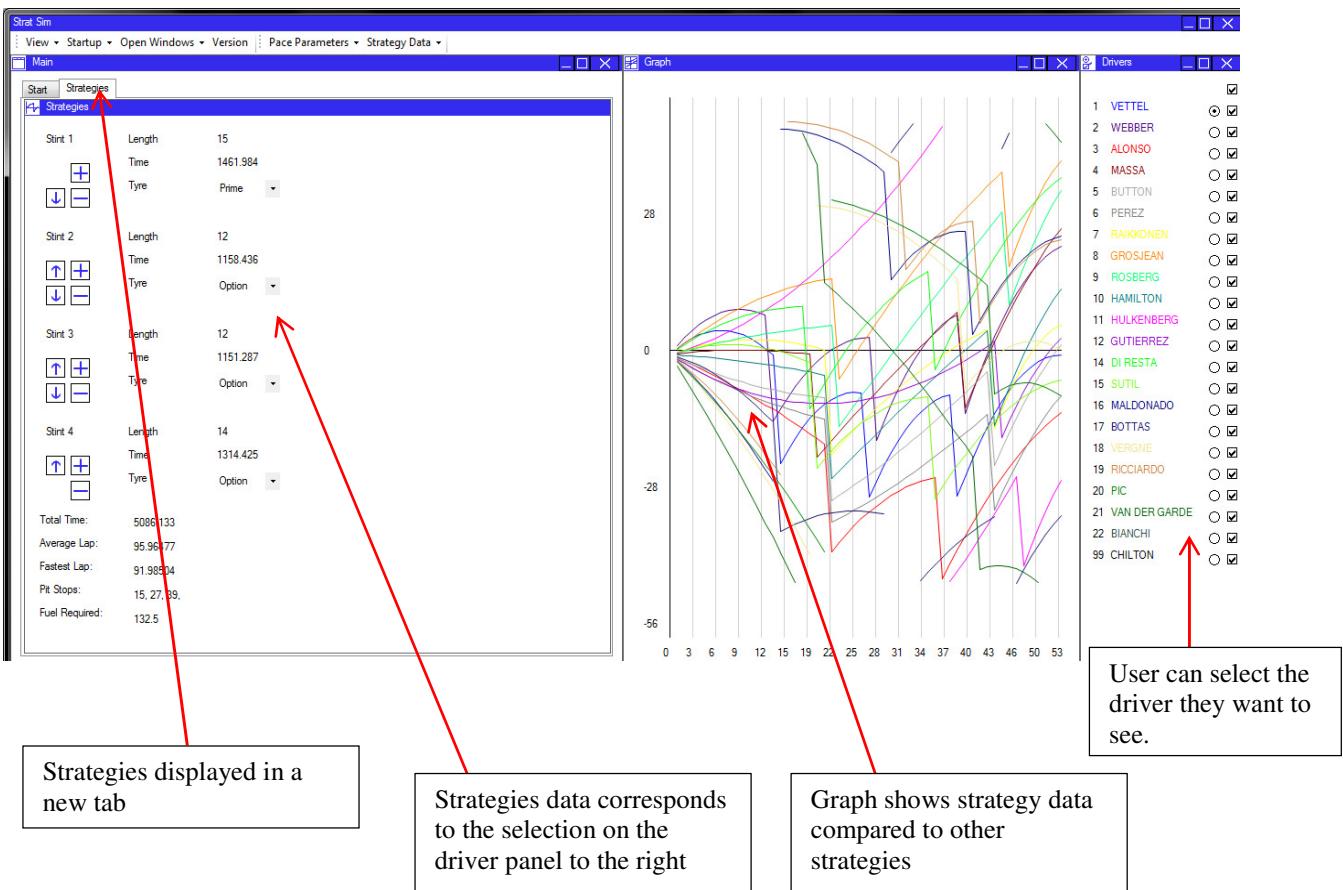
Test 1.1.13: Pace parameter view button clicked: Pace parameters displayed.

Figure 1.5:

Number	Team	Name	Top Speed	Tyre	Prime	Option	Pace	Fuel Effect	Fuel Consumption	P2fuel
1	1	VETTEL	299.2	-0.716	0.254	0.342	91.29	0.02	2.5	60
2	1	WEBBER	302.3	-0.567	0.254	0.342	90.915	0.02	2.5	60
3	2	ALONSO	299.5	0.8	0.1	0.199	91.665	0.031	2.5	60
4	2	MASSA	299.3	-0.228	0.1	0.199	91.378	0.031	2.5	60
5	3	BUTTON	294.4	-0.756	0.05	0.248	91.827	0.031	2.5	60
6	3	PEREZ	294.7	0.8	0.05	0.248	91.989	0.031	2.5	60
7	4	RAIKKONEN	294.7	-0.962	0.1	0.419	91.662	0.021	2.5	60
8	4	GROSJEAN	295.4	-0.568	0.1	0.419	91.365	0.021	2.5	60
9	5	ROSERG	293.9	-0.173	0.088	0.3	91.397	0.03	2.5	60
10	5	HAMILTON	297.6	0.8	0.088	0.3	91.253	0.03	2.5	60
11	6	HULKENBERG	297	-0.518	0.017	0.3	91.644	0.026	2.5	60
12	6	GUTIERREZ	296.4	-1.471	0.017	0.3	92.063	0.026	2.5	60
14	7	DI RESTA	298.5	-0.924	0.1	0.162	91.992	0.015	2.5	60

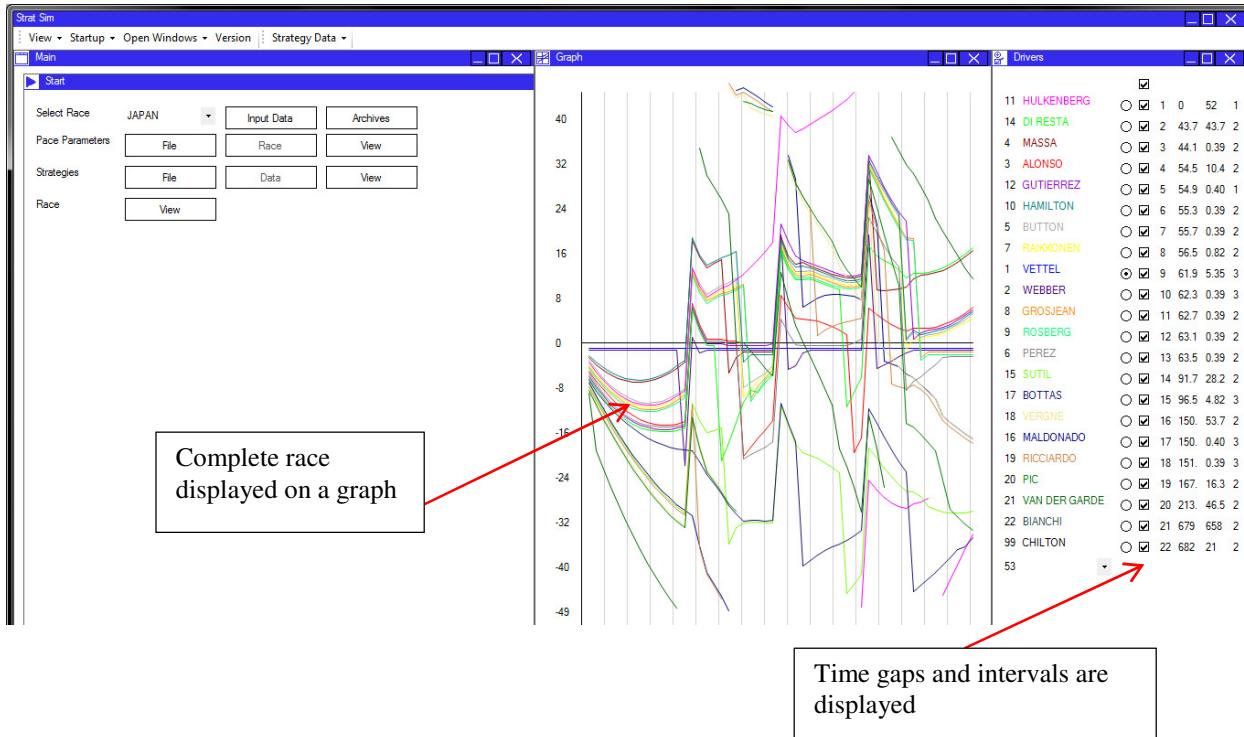
Test 1.1.16: Strategy viewer clicked. Strategy viewer is displayed along with graph.

Figure 1.6:



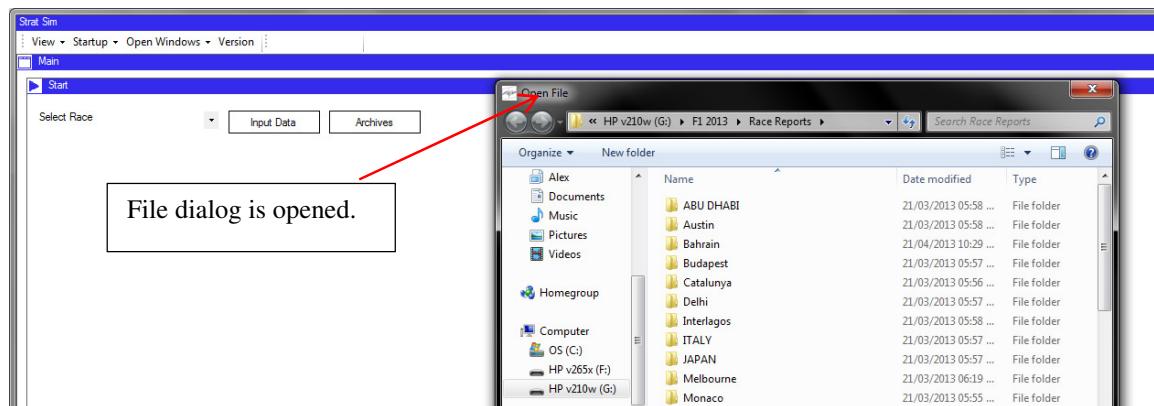
Test 1.1.18: Race viewer clicked. Race is displayed on graph and driver select panel shows time gaps.

Figure 1.7:



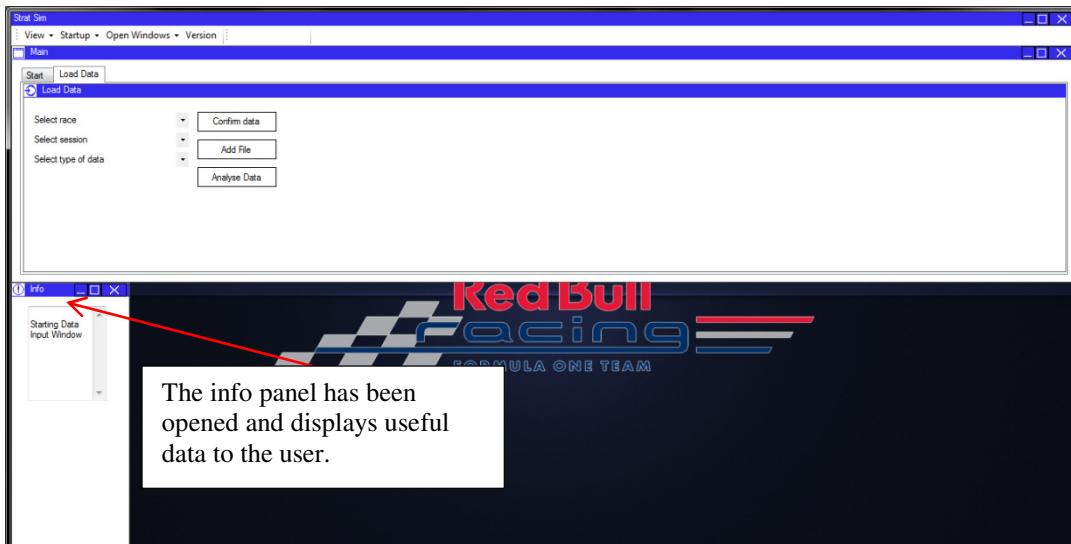
Test 1.2.4: Using the toolbar to start pace parameters from file. The program opens an open file dialog.

Figure 1.8:



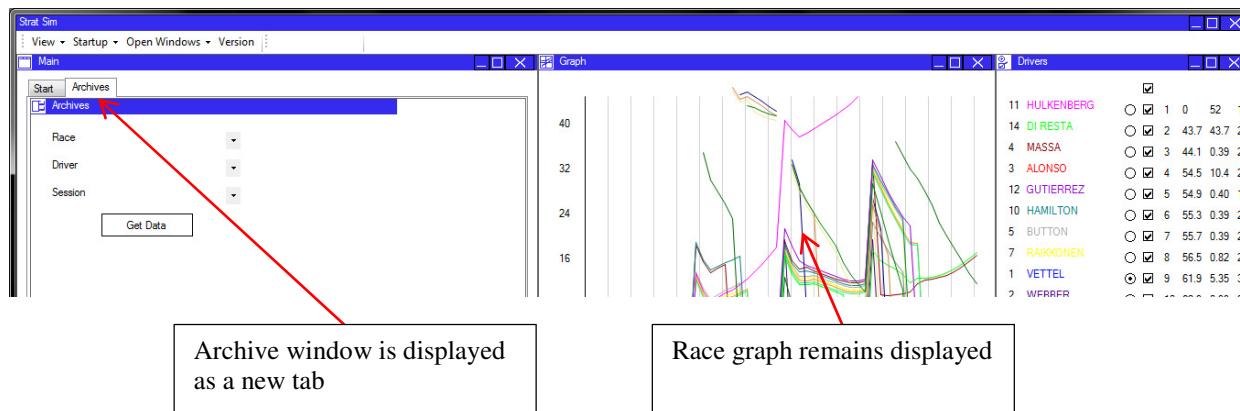
Test 1.2.7: Info panel is displayed using the toolbar.

Figure 1.9:



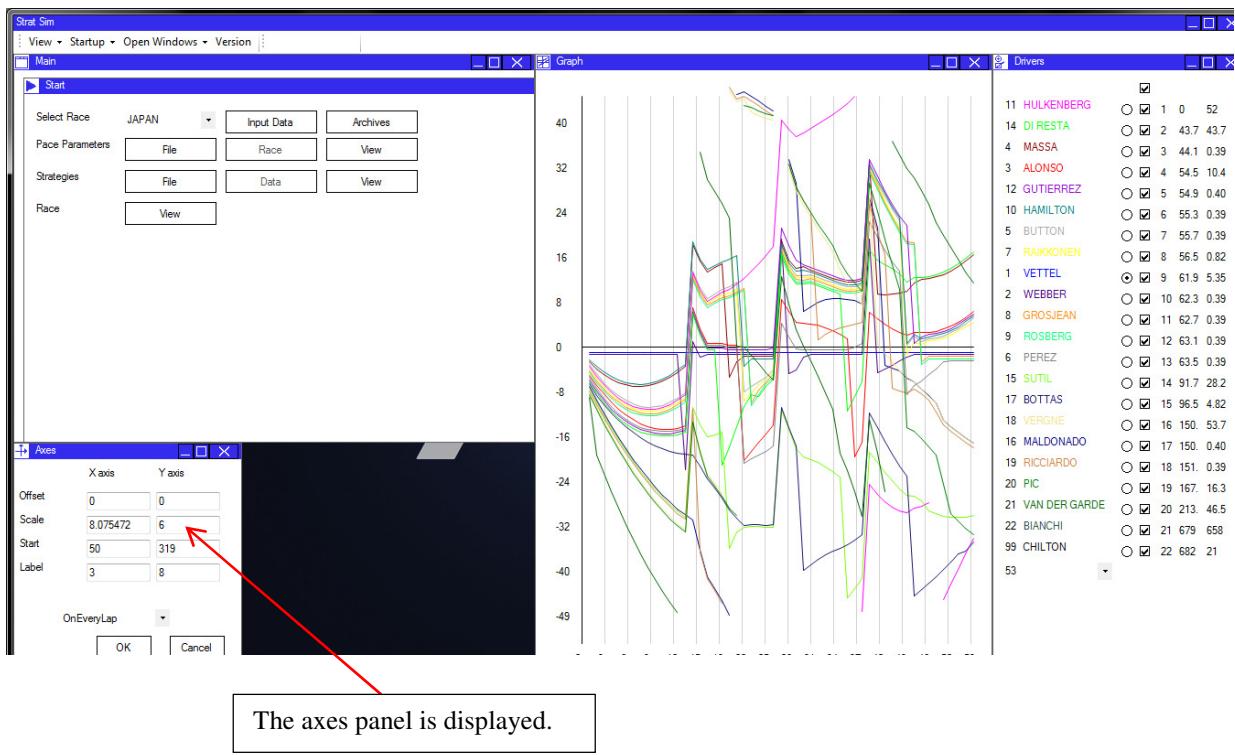
Test 1.2.13: Starting data archives during a race. The race remains shown while the data archives panel is opened.

Figure 1.10:



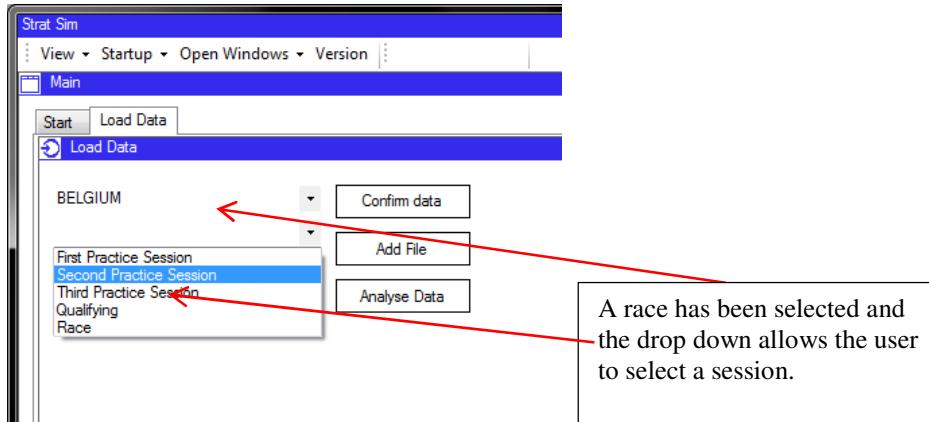
Test 1.2.14: Starting the Axes panel. The Axes panel is displayed and linked.

Figure 1.11:



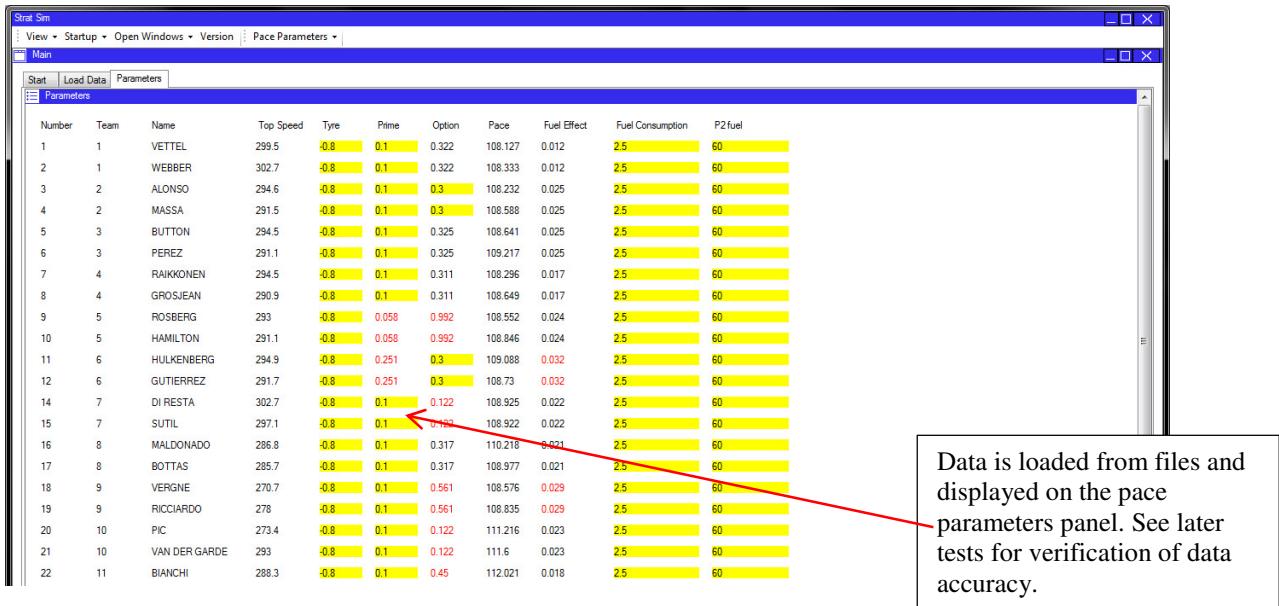
Test 1.3.1 Selecting a race and session using combo boxes

Figure 1.12:



Test 1.3.5: Finishing loading the data and starting the pace parameters page.

Figure 1.13:



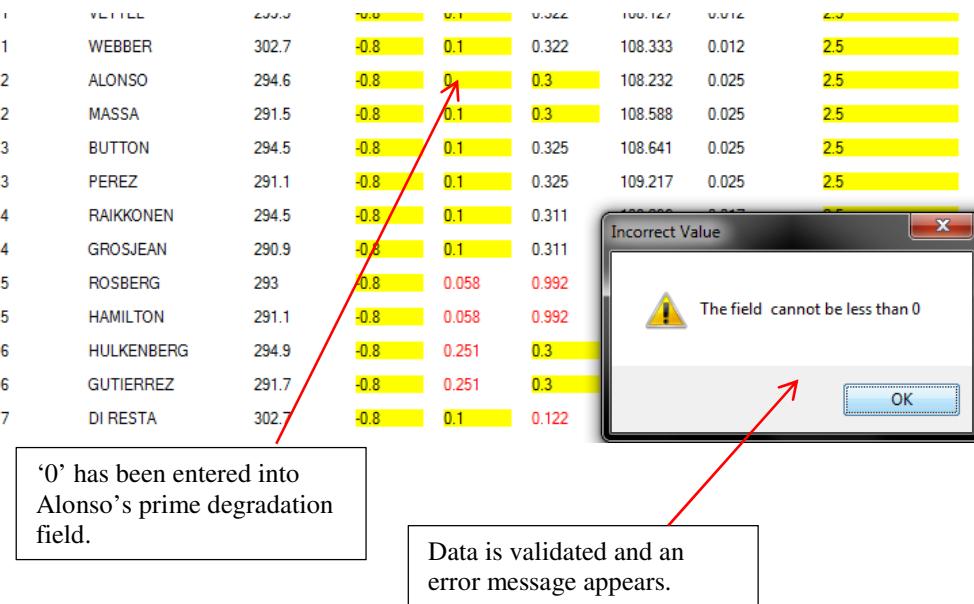
The screenshot shows the 'Pace Parameters' panel of the Strat.Sim software. The table displays the following data:

Number	Team	Name	Top Speed	Tyre	Prime	Option	Pace	Fuel Effect	Fuel Consumption	P2fuel
1	1	VETTEL	299.5	-0.8	0.1	0.322	108.127	0.012	2.5	60
2	1	WEBBER	302.7	-0.8	0.1	0.322	108.333	0.012	2.5	60
3	2	ALONSO	294.6	-0.8	0.1	0.3	108.232	0.025	2.5	60
4	2	MASSA	291.5	-0.8	0.1	0.3	108.588	0.025	2.5	60
5	3	BUTTON	294.5	-0.8	0.1	0.325	108.641	0.025	2.5	60
6	3	PEREZ	291.1	-0.8	0.1	0.325	109.217	0.025	2.5	60
7	4	RAIKKONEN	294.5	-0.8	0.1	0.311	108.296	0.017	2.5	60
8	4	GROSJEAN	290.9	-0.8	0.1	0.311	108.649	0.017	2.5	60
9	5	ROSERG	293	-0.8	0.058	0.992	108.552	0.024	2.5	60
10	5	HAMILTON	291.1	-0.8	0.058	0.992	108.846	0.024	2.5	60
11	6	HULKENBERG	294.9	-0.8	0.251	0.3	109.088	0.032	2.5	60
12	6	GUTIERREZ	291.7	-0.8	0.251	0.3	108.73	0.032	2.5	60
14	7	DI RESTA	302.7	-0.8	0.1	0.122	108.925	0.022	2.5	60
15	7	SUTIL	297.1	-0.8	0.1	0.122	108.922	0.022	2.5	60
16	8	MALDONADO	286.8	-0.8	0.1	0.317	110.218	0.021	2.5	60
17	8	BOTTAS	285.7	-0.8	0.1	0.317	108.977	0.021	2.5	60
18	9	VERGNE	270.7	-0.8	0.1	0.561	108.576	0.029	2.5	60
19	9	RICCIARDO	278	-0.8	0.1	0.561	108.835	0.029	2.5	60
20	10	PIC	273.4	-0.8	0.1	0.122	111.216	0.023	2.5	60
21	10	VAN DER GARDE	293	-0.8	0.1	0.122	111.6	0.023	2.5	60
22	11	BIANCHI	288.3	-0.8	0.1	0.45	112.021	0.018	2.5	60

Data is loaded from files and displayed on the pace parameters panel. See later tests for verification of data accuracy.

Test 1.4.2: Entering '0' into the prime degradation field for Alonso. The data is rejected and an error message is displayed.

Figure 1.14:



The screenshot shows the 'Pace Parameters' panel of the Strat.Sim software. The table displays the following data, with Alonso's prime degradation field highlighted:

Number	Team	Name	Top Speed	Tyre	Prime	Option	Pace	Fuel Effect	Fuel Consumption	P2fuel
1	WEBBER	WEBBER	302.7	-0.8	0.1	0.322	108.333	0.012	2.5	60
2	ALONSO	ALONSO	294.6	-0.8	0	0.3	108.232	0.025	2.5	60
2	MASSA	MASSA	291.5	-0.8	0.1	0.3	108.588	0.025	2.5	60
3	BUTTON	BUTTON	294.5	-0.8	0.1	0.325	108.641	0.025	2.5	60
3	PEREZ	PEREZ	291.1	-0.8	0.1	0.325	109.217	0.025	2.5	60
4	RAIKKONEN	RAIKKONEN	294.5	-0.8	0.1	0.311	108.296	0.017	2.5	60
4	GROSJEAN	GROSJEAN	290.9	-0.8	0.1	0.311	108.649	0.017	2.5	60
5	ROSERG	ROSERG	293	-0.8	0.058	0.992	108.552	0.024	2.5	60
5	HAMILTON	HAMILTON	291.1	-0.8	0.058	0.992	108.846	0.024	2.5	60
6	HULKENBERG	HULKENBERG	294.9	-0.8	0.251	0.3	109.088	0.032	2.5	60
6	GUTIERREZ	GUTIERREZ	291.7	-0.8	0.251	0.3	108.73	0.032	2.5	60
7	DI RESTA	DI RESTA	302.7	-0.8	0.1	0.122	108.925	0.022	2.5	60

'0' has been entered into Alonso's prime degradation field.

Data is validated and an error message appears.

Incorrect Value  
The field cannot be less than 0  
OK

Test 1.4.4: '0.3' is entered into the prime tyre degradation box, which is rejected and the value is set to its original value.

Figure 1.15:

m	Name	Top Speed	Tyre	Prime	Option
	VETTEL	299.5	-0.8	0.1	0.322
	WEBBER	302.7	-0.8	0.1	0.322
	ALONSO	294.6	-0.8	0.3	0.3
	MASSA	291.5	-0.8	0.1	0.3
	BUTTON	294.5	-0.8	0.1	0.325

Team	Name	Top Speed	Tyre	Prime	Op
1	VETTEL	299.5	-0.8	0.1	0.3
1	WEBBER	302.7	-0.8	0.1	0.3
2	ALONSO	294.6	-0.8	0.1	0.3
2	MASSA	291.5	-0.8	0.1	0.3
2	BUTTON	294.5	-0.8	0.1	0.3

Test 1.4.9: '0.005' is entered into the prime tyre degradation box, which is accepted as valid data. The text box turns blue as the data has been user modified, and the text itself turns red as the value is sufficiently far away from the default that it could be anomalous.

Figure 1.16:

'0.005' is accepted.

Name	Top Speed	Tyre	Prime	Option	P
VETTEL	299.5	-0.8	0.1	0.322	10
WEBBER	302.7	-0.8	0.1	0.322	10
ALONSO	294.6	-0.8	0.005	0.3	10
MASSA	291.5	-0.8	0.1	0.3	10

Test 1.4.12: '0.5' is entered into the tyre delta field for Rosberg. The data is rejected and an error message is displayed.

Figure 1.17:

Name	Top Speed	Tyre	Prime	Option	Pace	Fuel Effect
VETTEL	299.5	-0.8	0.1	0.322	108.127	0.012
WEBBER	302.7	-0.8	0.1	0.322	108.333	0.012
ALONSO	294.6	-0.8	0.005	0.3	108.232	0.025
MASSA	291.5	-0.8	0.1	0.3	108.588	0.025
BUTTON	294.5	-0.8	0.1	0.325	108.641	0.025
PEREZ	291.1	-0.8	0.1	0.325	109.217	0.025
RAIKKONEN	294.5	-0.8	0.1	0.311	108.296	0.017
GROSJEAN	290.9	-0.8	0.1	0.311	108.322	0.022
ROSERG	293	0.5	0.058	0.992	108.322	0.022
HAMILTON	291.1	-0.8	0.058	0.992	108.322	0.022
HULKENBERG	294.9	-0.8	0.251	0.3	108.322	0.022
GUTIERREZ	291.7	-0.8	0.251	0.3	108.322	0.022
DI RESTA	302.7	-0.8	0.1	0.122	108.322	0.022
SUTIL	297.1	-0.8	0.1	0.122	108.322	0.022
MALDONADO	286.8	-0.8	0.1	0.317	110.218	0.021

'0.5' is entered and rejected because it is greater than zero.

A helpful error message is displayed.

Test 1.5.2: Testing the 'open' functionality for pace parameters. A file dialog is opened and then the pace parameters are updated. Note that they have changed in the second screenshot, showing that data has been successfully loaded from the file.

Figure 1.18a:

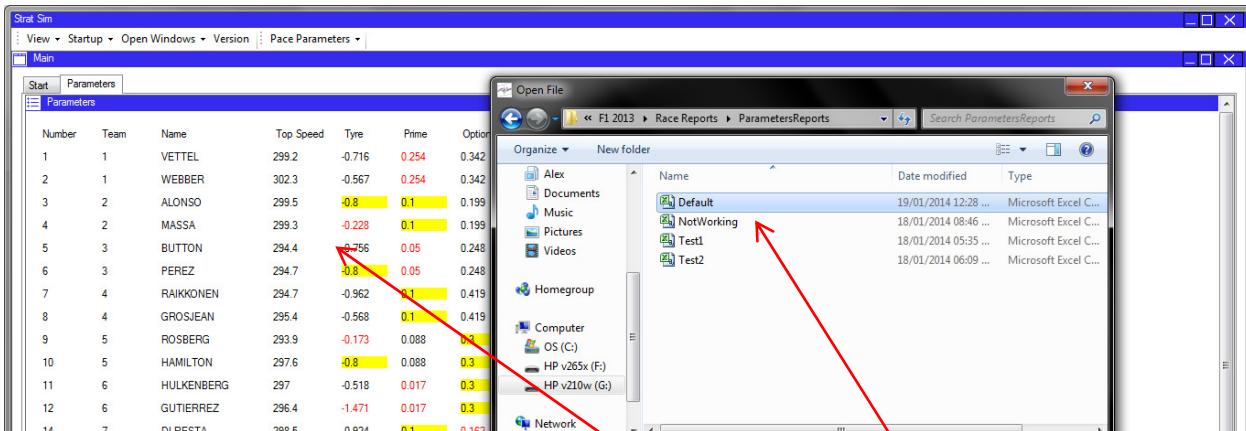
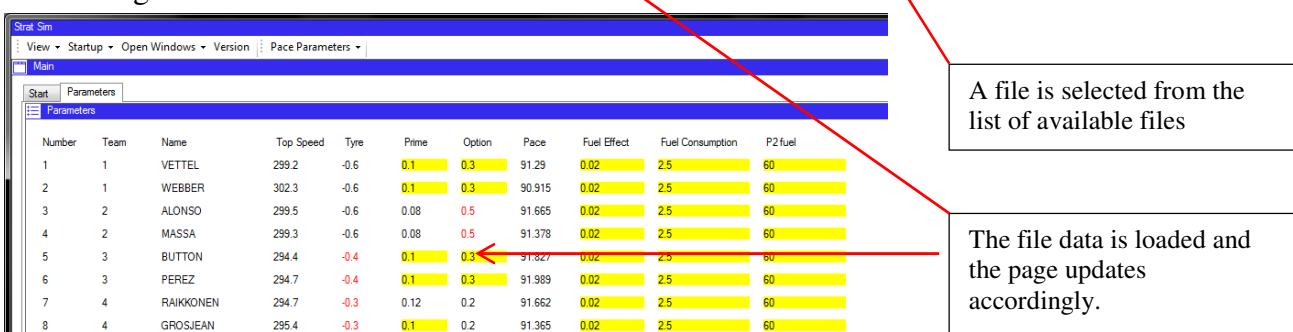


Figure 1.18b:



Testing 1.6: Testing of strategy manipulation panel. The system uses an initial test data set as shown below.

Figure 1.19a: (Calculated Pace Data)

Number	Team	Name	Top Speed	Tyre	Prime	Option	Pace	Fuel Effect	Fuel Consumption	P2fuel
1	1	VETTEL	315	-0.8	0.1	0.3	200	0.02	2.5	60
2	1	WEBBER	315	-0.8	0.1	0.3	200	0.02	2.5	60
3	2	ALONSO	315	-0.8	0.1	0.3	200	0.02	2.5	60

Figure 1.19b: (Race Data)

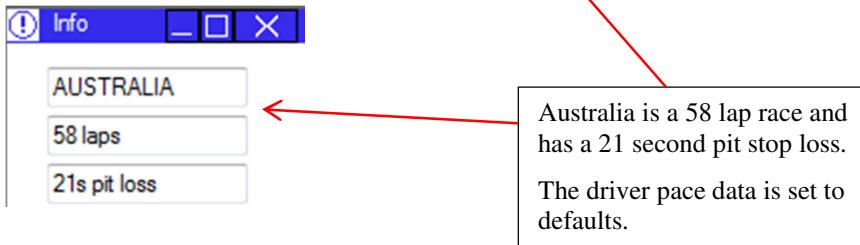
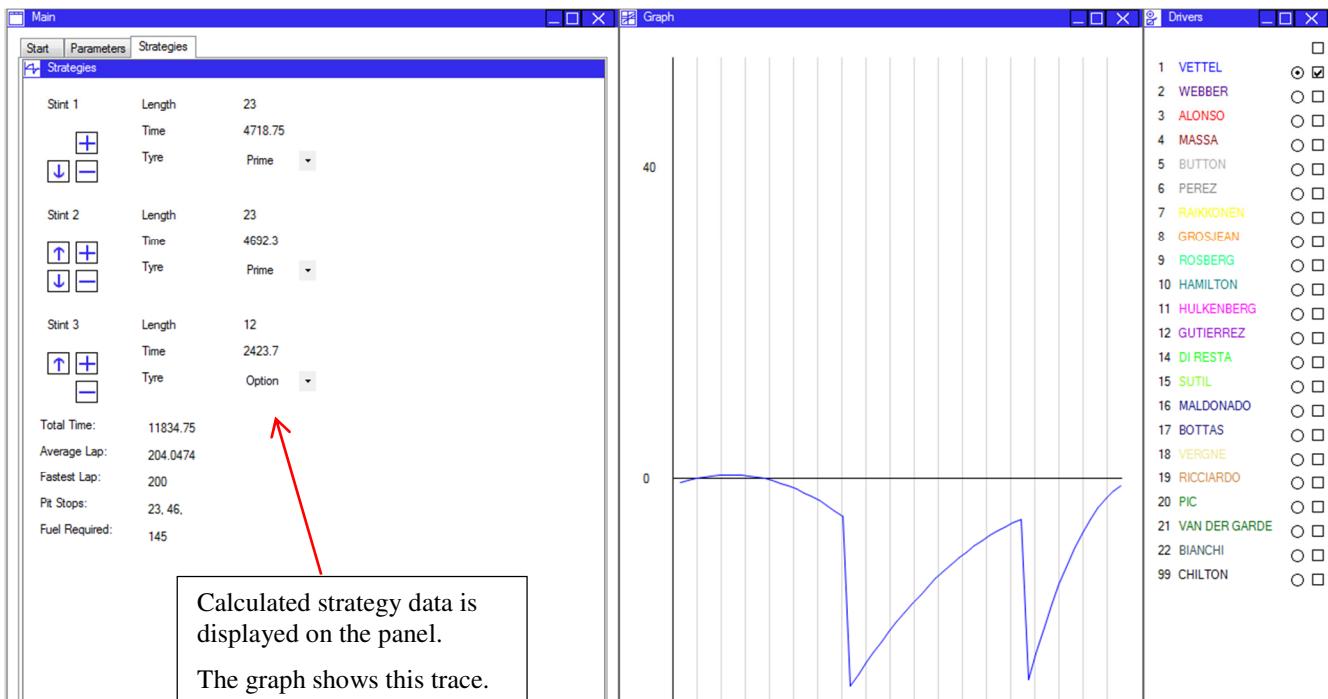
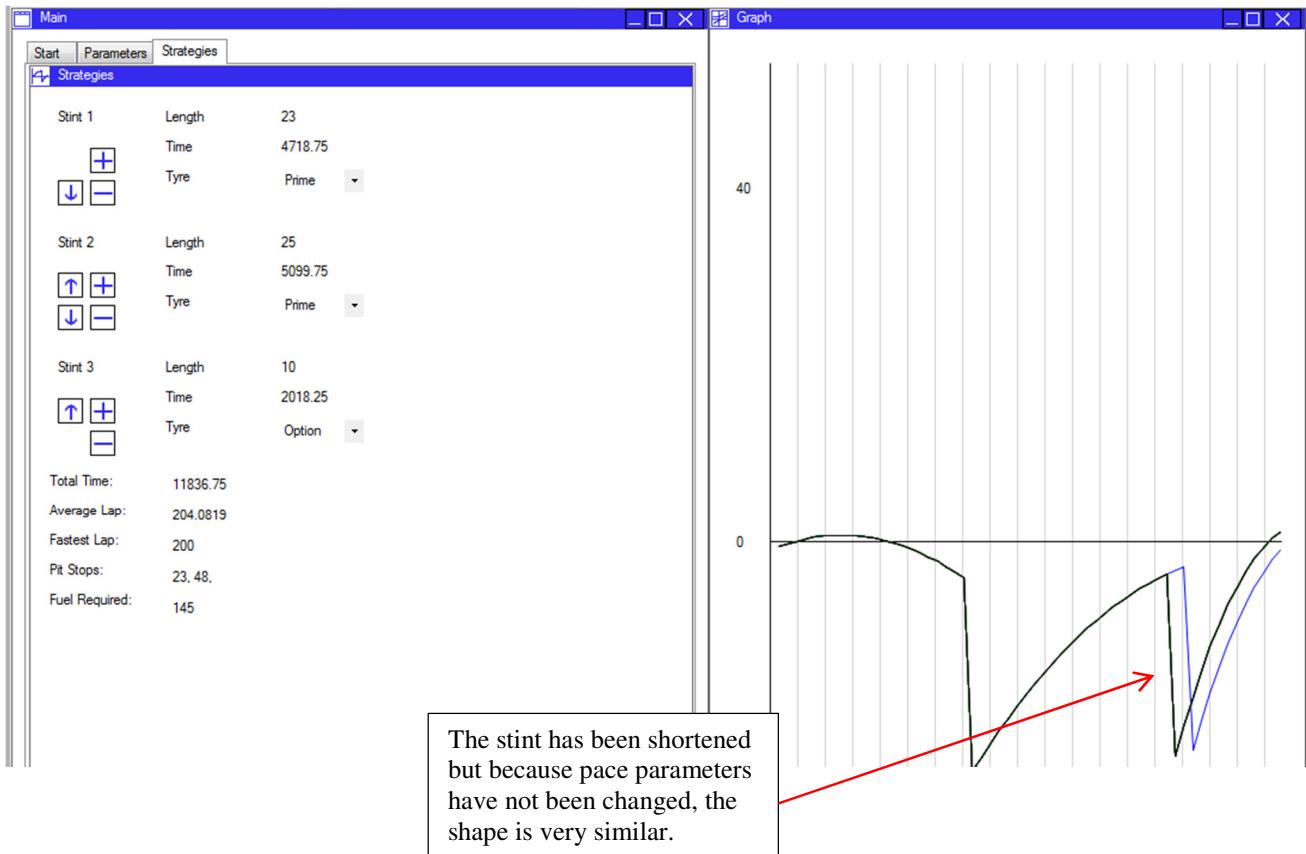


Figure 1.19c: (Initial calculated strategies)



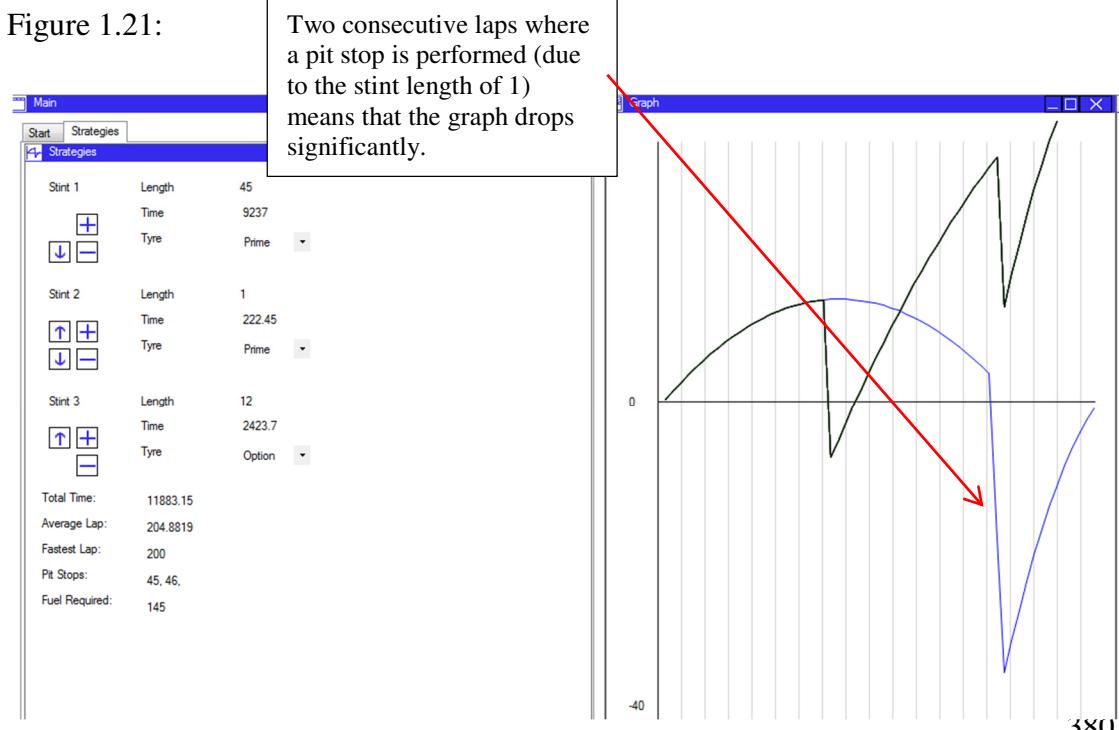
Test 1.6.2: Changing the length of the last stint. The graph updates; the initial situation is shown on the graph for comparison.

Figure 1.20:



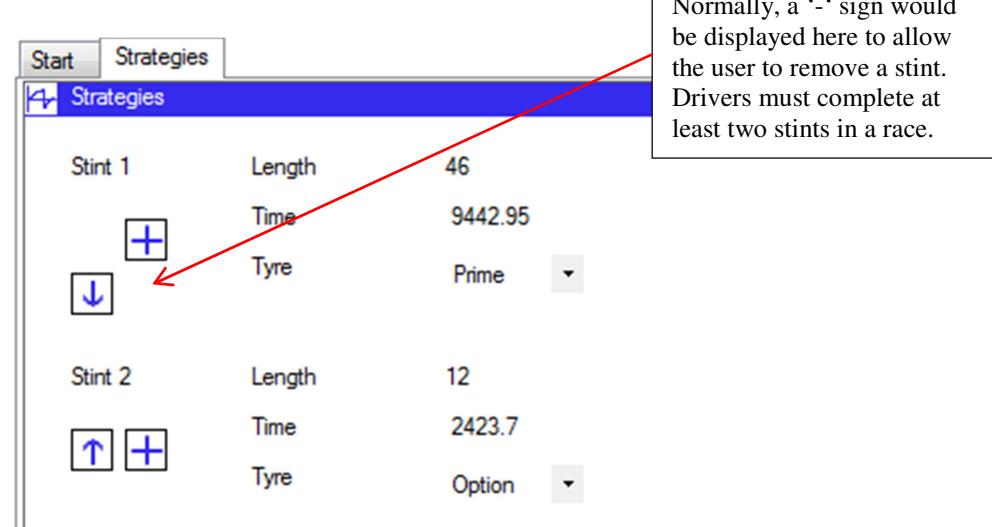
Test 1.6.4a: Retest to check if the program will accept an upper boundary stint length. The program accepts the value and updates the stint accordingly.

Figure 1.21:



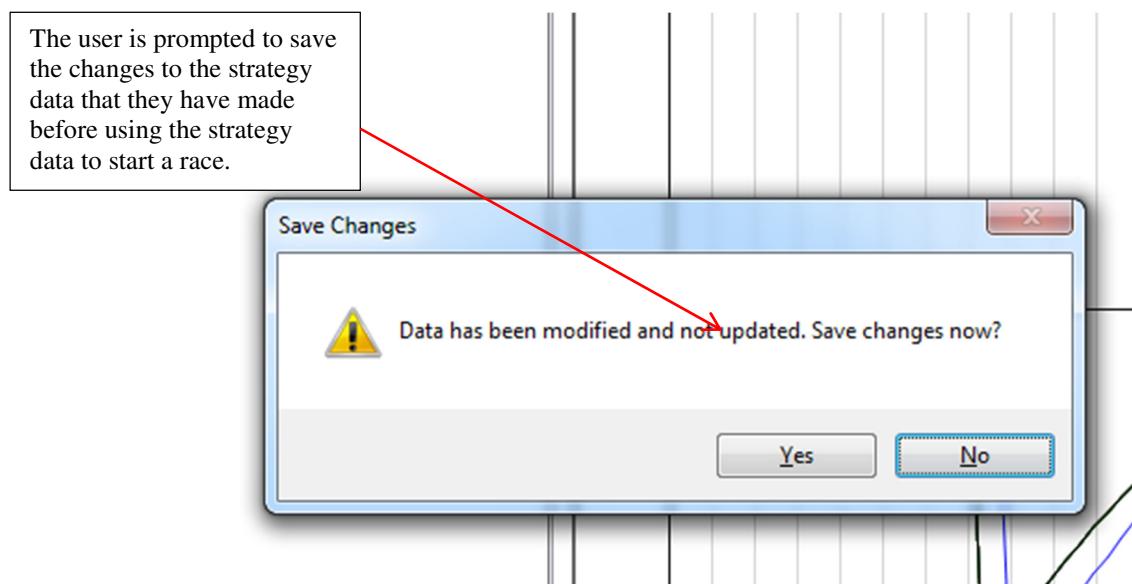
Test 1.6.11: There should be no ‘remove stint’ button displayed when the strategy has only two stints.

Figure 1.22:



Test 1.6.17: Trying to start a race simulation when data has been modified and not updated. An error message is displayed offering the user the option to update the data before they proceed to the race simulation.

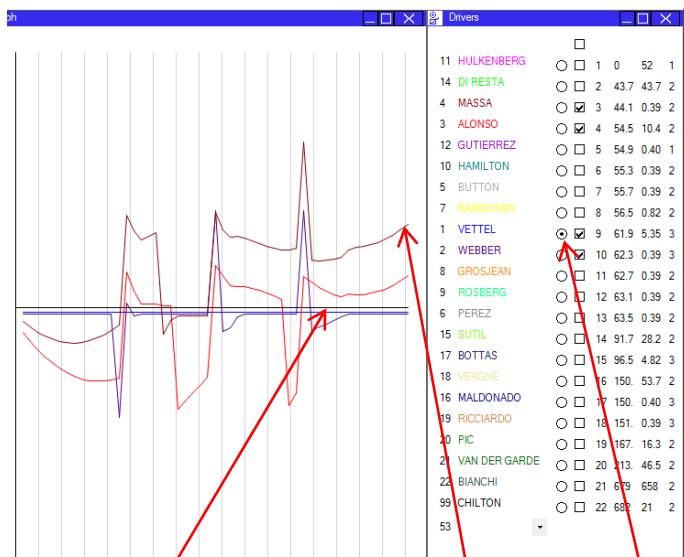
Figure 1.23:



Test 1.7.5: Hiding the trace for the driver on whom the graph is currently normalised.

The program selects the next best driver to normalise on. Here, screenshots show before and after the hiding of Vettel's trace.

Figure 1.24a:



Vettel's blue line is currently straight and flat; he is the normalised driver.

Massa is the fastest driver of this group

The radio button confirms Vettel is the normalised driver.

Vettel's trace is hidden.

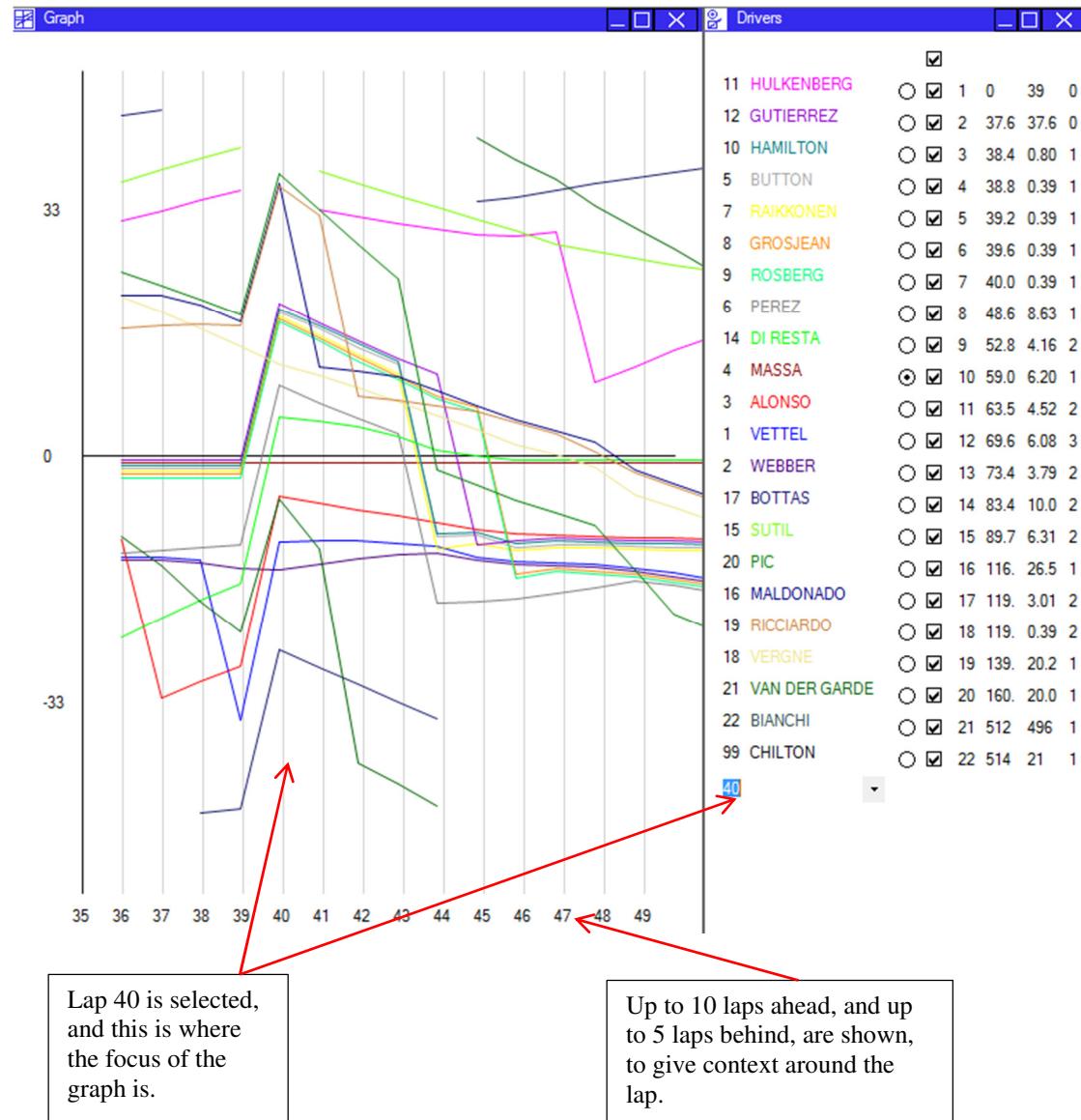
Massa has been selected and his line is now flat.

Figure 1.24b:



Test 1.7.7: Changing the index of the lap number check box. The graph zooms in on this lap and shows it in more detail.

Figure 1.25:



Test 1.9.1: Loading archive data from a previously loaded PDF file. The data is loaded and displayed; these screenshots show a comparison between the system's data and the PDF data.

Only flying laps are shown as the data here will probably be used solely for pace analysis. Only the flying laps are useful for this purpose. Also included are identifiers for the start and end of a stint, which are important for identifying things like tyre degradation.

Figure 1.26:

NO	TIME	NO	TIME
1 P	10:04:19	13	2:18.419
2 P	2:35.269	14	1:38.801
3	36:07.465	15	1:38.965
4	1:35.217	16	1:38.810
5	1:46.635	17	1:38.631
6	1:35.049	18	1:38.955
7	1:44.842	19	1:39.366
8	<b>1:34.768</b>	20	1:40.592
9 P	1:51.469	21	1:39.782
10	12:20.894	22	1:40.175
11	1:39.014	23	1:40.662
12 P	1:59.415	24 P	1:49.113

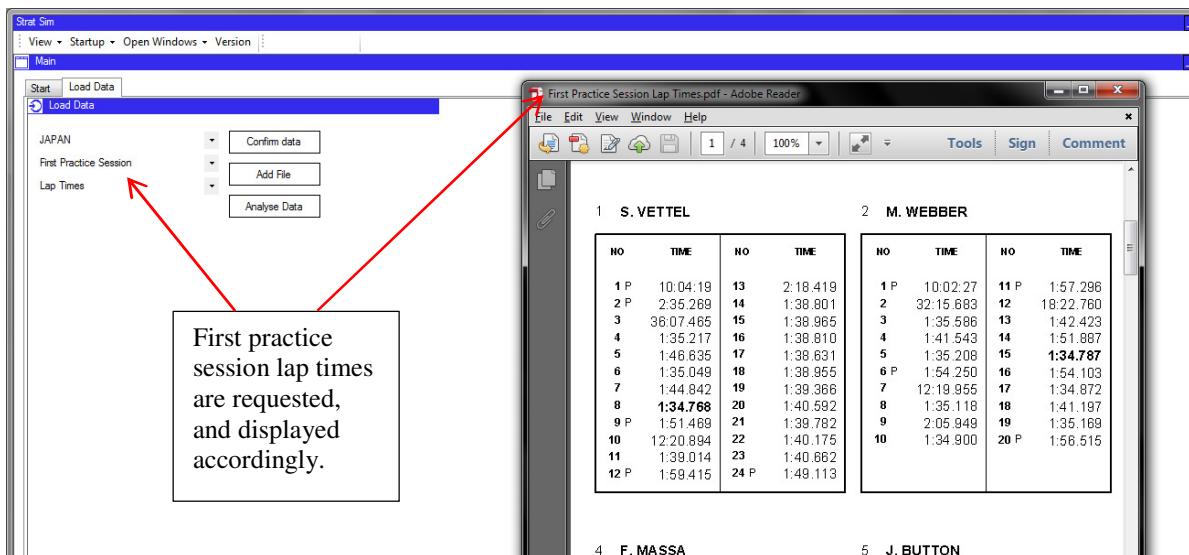
Test 1.9.4a: Retest of trying to load data from Malaysia in the data archives panel. The system should not display any lap times because there is no data from Malaysia. It does not crash but displays no lap times, which is exactly as intended.

Figure 1.27:

## 2. Specification Testing

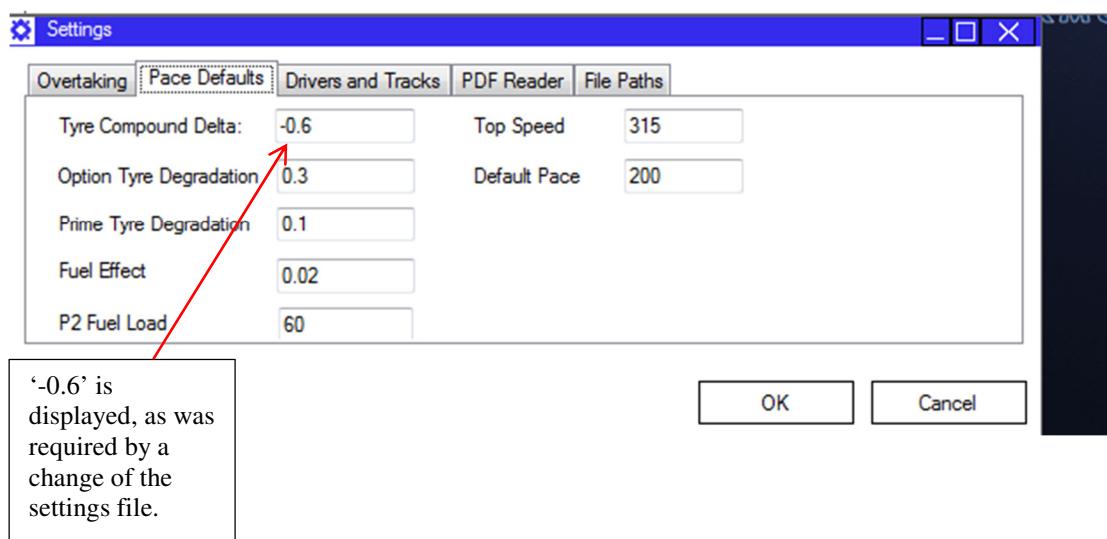
Test 2.2.2: When the PDF file combo boxes are populated, the program will open the correct PDF file.

Figure 2.1



Test 2.2.5: Editing the settings externally and viewing them within the program. The updated values are displayed.

Figure 2.2



Test 2.3.1: Opening a lap data file in Microsoft Excel. The program opens with data separated into columns as appropriate. The first column is a time to 3dp, the second is an integer.

Figure 2.3

1	VETTEL
2	612.2
3	143.182
4	1410.38
5	87.875
6	86.056
7	85.914
8	91.403
9	86.119
10	91.062
11	566.212
12	86.108
13	85.753
14	99.52299
15	829.55
16	89.32
17	88.444
18	88.258
19	88.628
20	88.229
21	88.23599
22	88.313
23	88.227
24	88.267
25	88.443
26	RR 79999

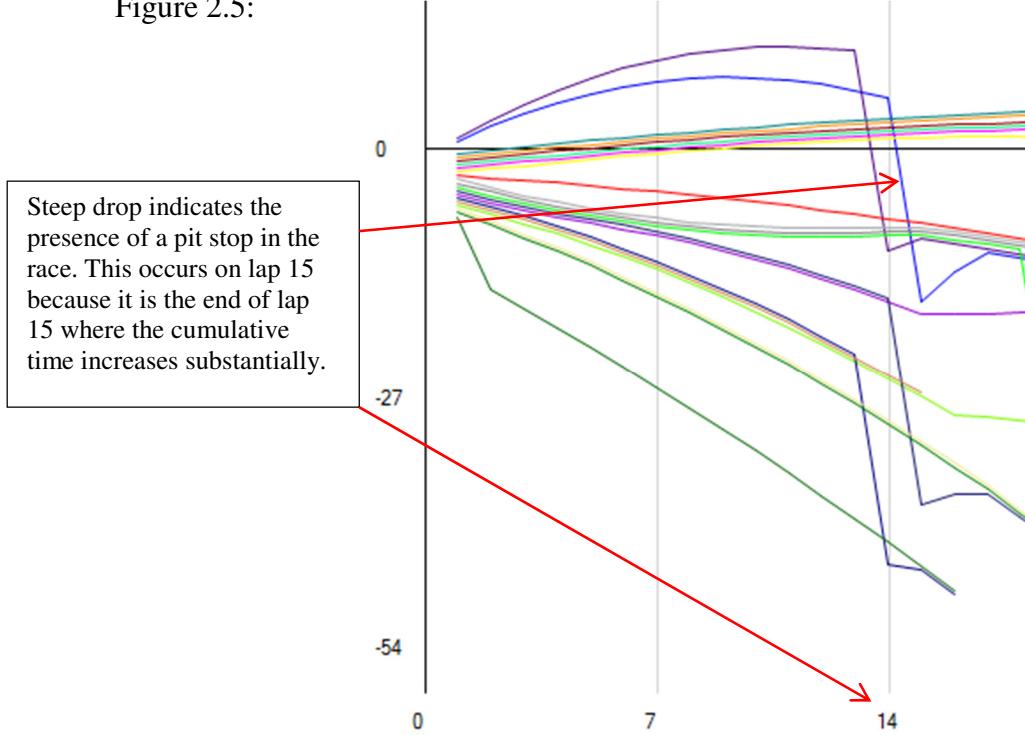
Test 2.4.3: Pace parameters are loaded. Default values are used where the system cannot calculate a value. These are highlighted in yellow as standard so they are obvious to the user if he needs to change them. Note also the red text, indicating an anomalous result according to the default values (i.e. more than 40% error).

Figure 2.4:

Team	Name	Top Speed	Tyre	Prime	Option	Pace	Fuel Effect	Fuel Consumption	P2 fuel
1	VETTEL	336.1	-1.1	0.181	0.212	83.755	0.003	2.5	60
1	WEBBER	336.3	-0.827	0.181	0.212	83.968	0.003	2.5	60
2	ALONSO	338.3	-0.07	0.057	0.174	84.142	0.007	2.5	60
2	MASSA	338.4	-0.73	0.057	0.174	84.132	0.007	2.5	60
3	BUTTON	331.8	-0.303	0.1	0.651	84.515	0.007	2.5	60
3	PEREZ	331.6	-0.18	0.1	0.651	84.502	0.007	2.5	60
4	RAIKKONEN	337.8	-0.625	0.1	0.3	84.61	0.02	2.5	60
4	GROSJEAN	337.1	-0.979	0.1	0.3	84.716	0.02	2.5	60
5	ROSBERG	337.8	-0.137	0.1	0.477	84.192	0.007	2.5	60
5	HAMILTON	338.1	-0.025	0.1	0.477	84.512	0.007	2.5	60
6	HULKENBERG	335.7	-0.639	0.075	0.265	84.065	0.011	2.5	60
6	GUTIERREZ	340	-0.106	0.075	0.265	85.124	0.011	2.5	60

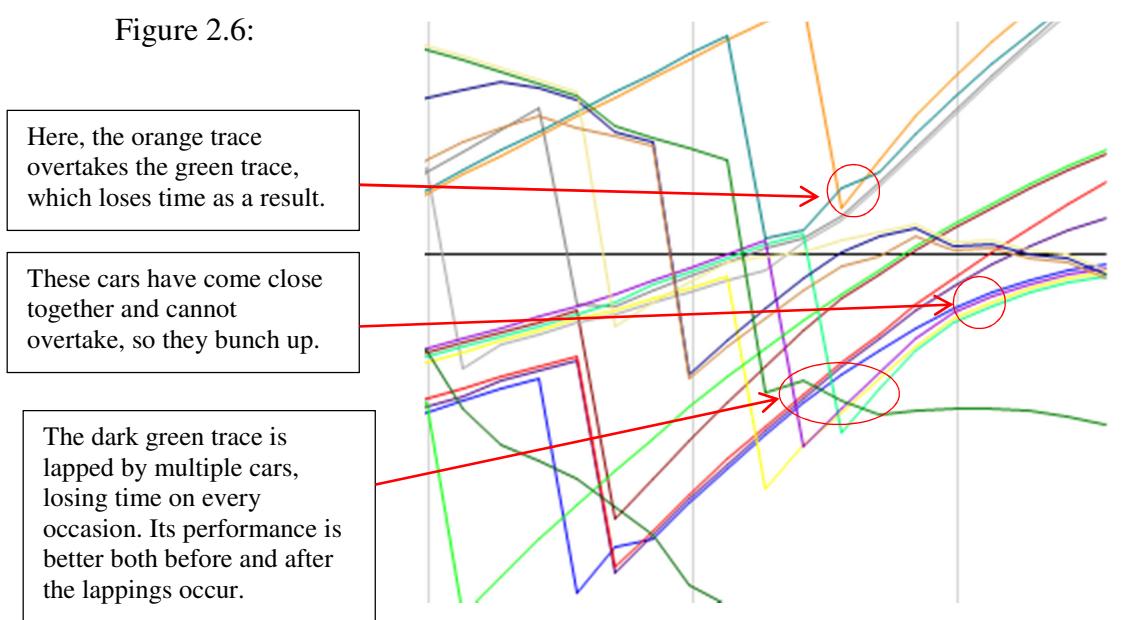
Test 2.6.2b: Vettel – the blue trace – should experience a time loss on lap 15. The lap is significantly longer indicating the presence of the pit stop. The screen shot is an excerpt from the graph.

Figure 2.5:



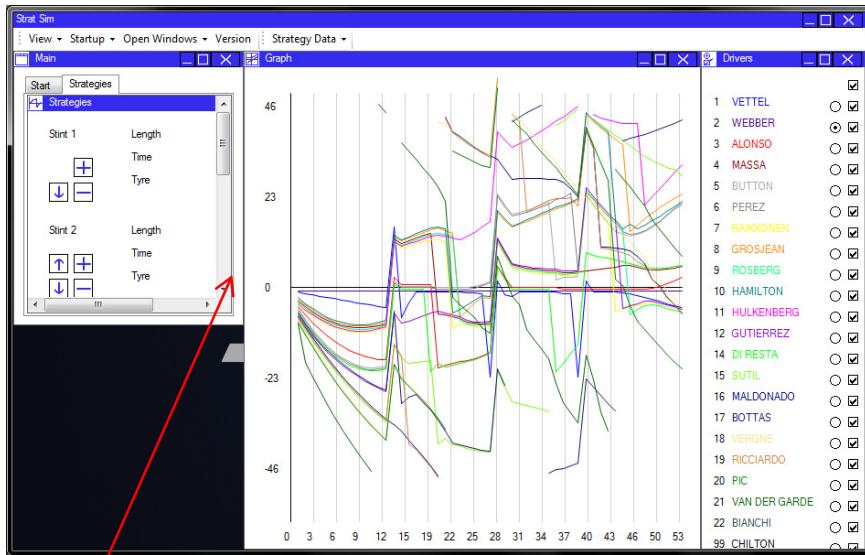
Test 2.6.3b: Checking for evidence of cars stuck in traffic, and making overtakes. The following screenshot shows all of the examples. Where two lines are close together, one car is most likely stuck behind the other unable to overtake. Where one line jumps another one is an example of an overtake, which will cause a time loss to the car behind. However, if a car is lapped, it will be passed by the car ahead, and will lose time being passed. Also note the different colours.

Figure 2.6:



Test 2.9.2: The window should automatically adjust when it is re-sized to layout its controls appropriately. The window here has re-sized and all of the important controls remain at the required size. The graph also remains visible.

Figure 2.7:

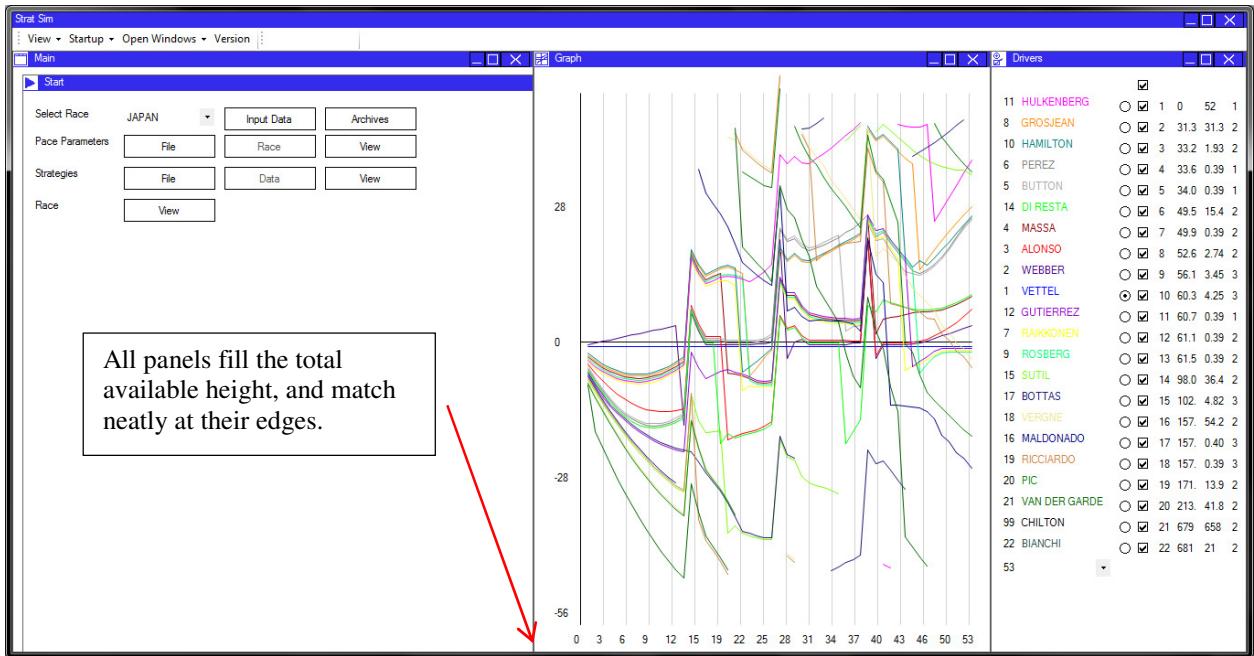


The graph and driver panel remain at their required sizes, while the lesser required panel, the main panel, is shrunk to accommodate this change.

### 3. Flow Layout Testing

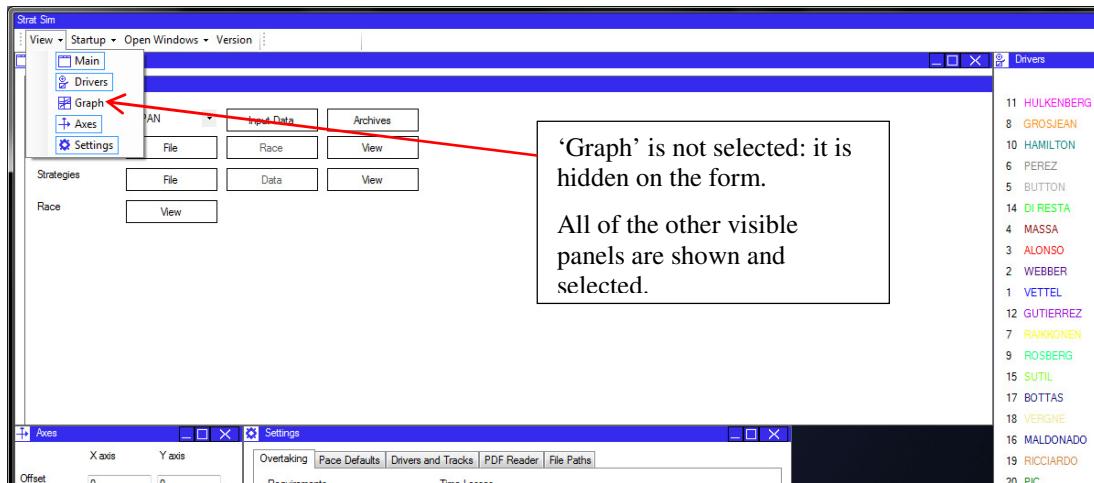
Test 3.1.1: Starting a race simulation to demonstrate the layout of the panels on the window. There is no background showing and the panels are displayed at their required sizes.

Figure 3.1:



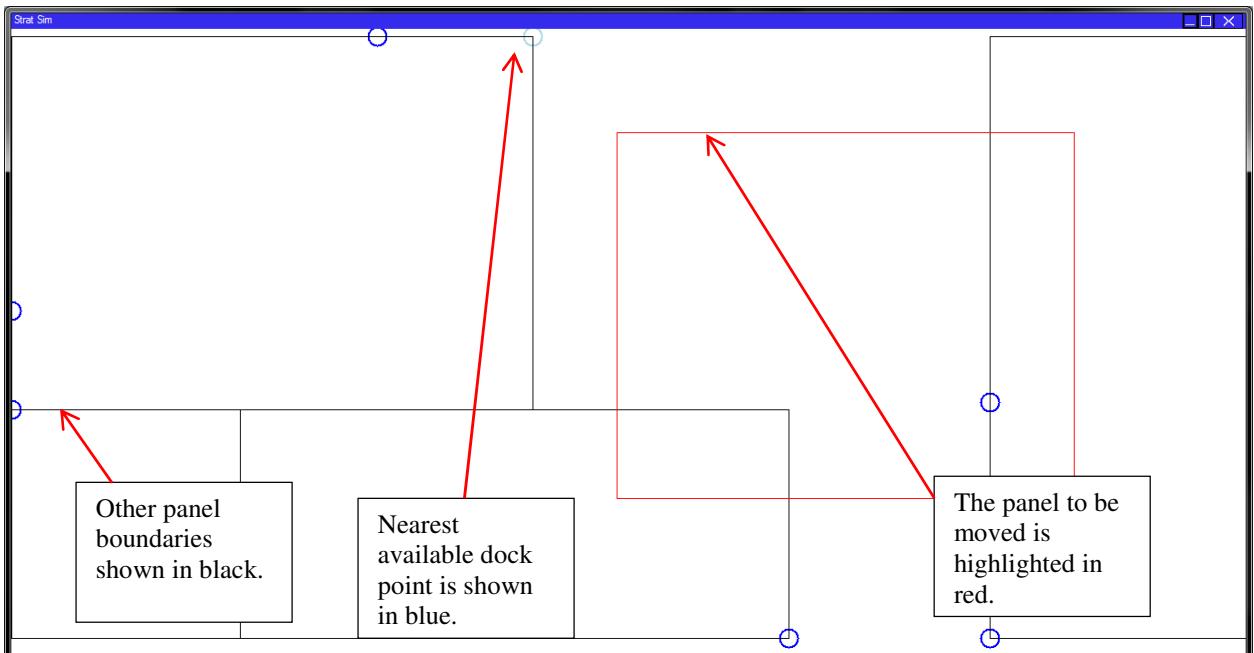
Test 3.2.2: The toolbar shows the graph as hidden and allows the user to toggle the button to show it again. Here, the toolbar is shown before the test is initiated.

Figure 3.2:



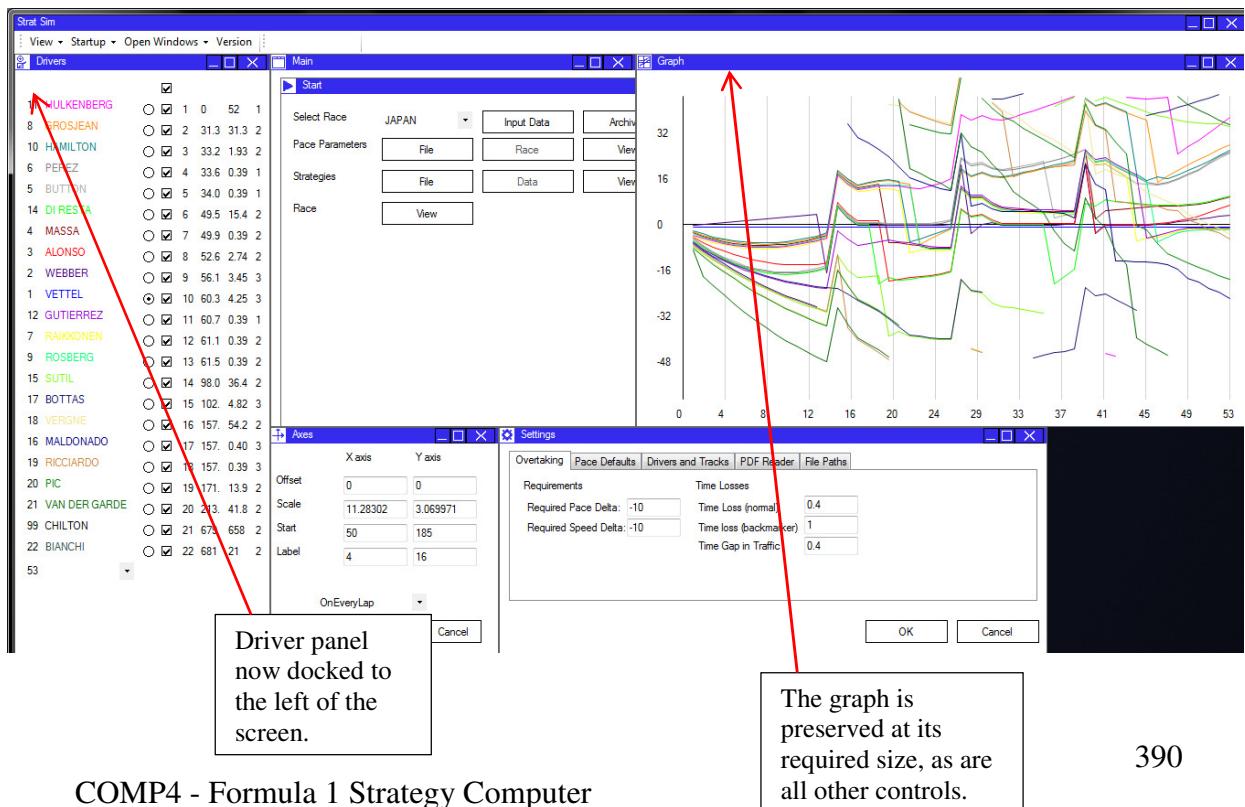
Test 3.4.1: Selecting a panel's header will activate the reordering function. This is a simplified window which allows the selected panel to be moved. Displayed panels are shown in black outlines with the selected panel in red.

Figure 3.3:



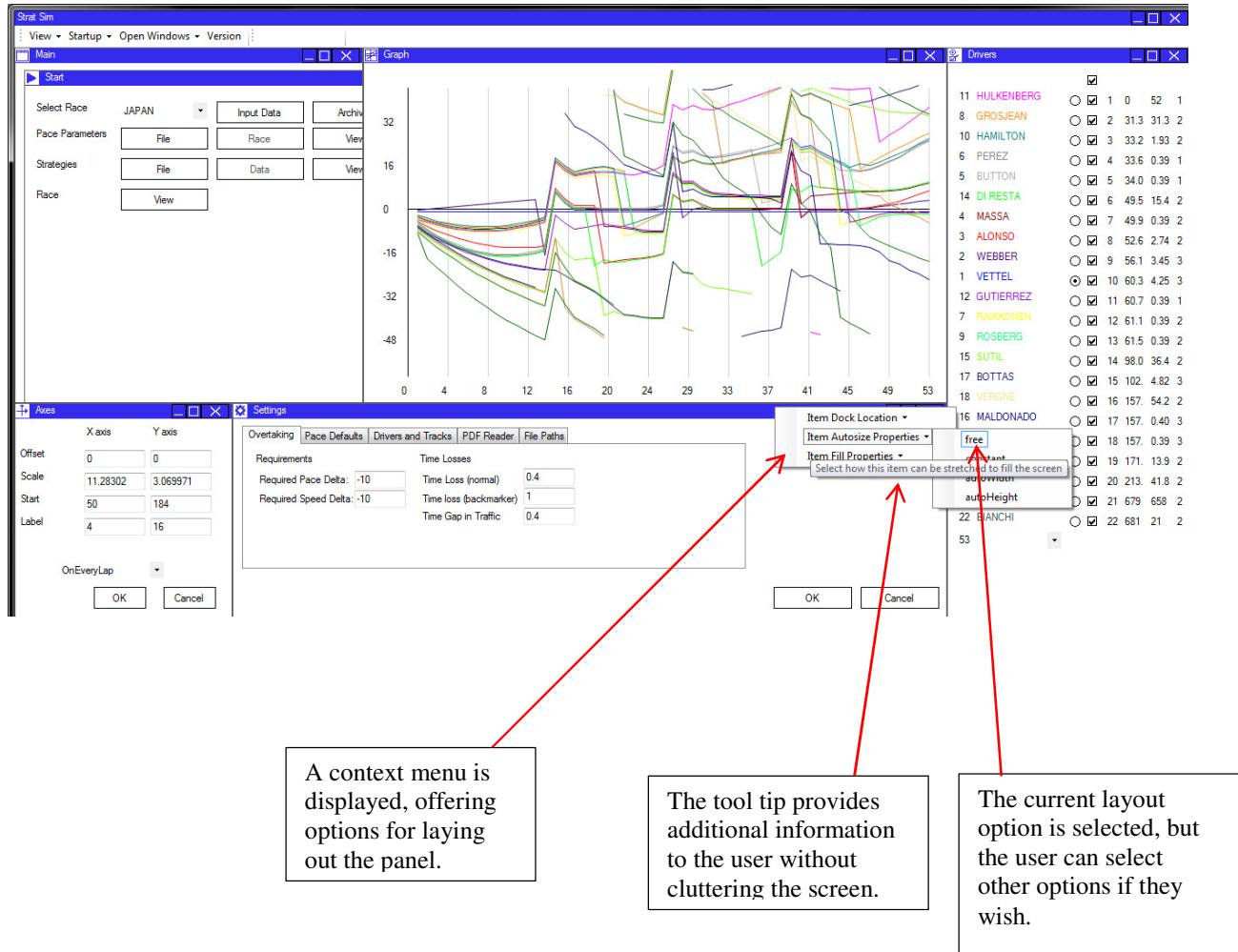
Test 3.5.1: Changing the dock type of the driver panel (on the right by default) to the left side of the screen. The panel is moved and the window is laid out accordingly.

Figure 3.4:



Test 3.6.2: Setting the autosize type of the settings panel to ‘free’ allows it to resize in all directions. It will fill the space to the right of it. Its current autosize type is also selected in the context menu. Note also the tooltip shown on the context menu.

Figure 3.5:



## System Maintenance

The following section is designed to be used by future developers in control of the system. It documents how to perform crucial updates or maintenance to the system, and will provide information about some of the specific features of the program.

### System Overview

#### System Details

Name ..... StratSim  
Developer ..... Alex McCormick  
Release Date..... 01/04/2014  
Copyright ..... © Alex McCormick 2013-2014  
Description..... System for calculating and optimising the strategies used by cars in a Formula One Event.

#### System Specification

Language..... C#.NET  
Developed with ..... Microsoft ® Visual Studio ™ 2012  
Design ..... Object Oriented  
Testing..... MSTest Unit Testing  
Size..... 3.3 Mb

#### Detailed System Description

StratSim is a detailed and high-precision strategy optimisation program, tailored for use by Formula One strategy teams. It can be used once vehicle performance data is known to simulate the optimum race strategies, and subsequently simulate a race, allowing the users to see how their car might perform when run against other cars.

The system will calculate pace data from the timing data produced by the FIA from timekeeping transponders in the cars. It can also allow users to update this data for greater accuracy.

This data is used to optimise strategies, selecting the strategy which contributes to the fastest possible race time. This strategy data is also displayed and can be altered by the user if necessary.

A race simulation can then be run using the stored strategy data and pace data. This will produce an expected race outcome. Changes can be made to the strategy and the race re-run to see how these changes affect the race outcome.

The system will allow data to be saved to and loaded from files stored elsewhere on the computer. This allows the user to save data for later analysis, or save and distribute the information contributing to the most effective end result.

### Additional System Features

- When data is loaded from the FIA PDF timing files, it is saved to archive files. These can be accessed at any time to view the lap times from past events, in an archive function.
- Archived data can also be used to re-run old simulations. This data is loaded automatically and can be processed quickly to save having to enter data repeatedly each time the system is loaded.
- The system contains functionality for altering multiple settings used in the program. This includes the standard file input and output paths, parameter defaults, and race-specific settings.
- The system is linked to a database to populate driver and track data. Updating the race calendar and driver lists can be done through the database. Similarly, the database can be used in future development when necessary; if tyre data is to be pooled and used to produce a more accurate degradation model, or if driver pace data is to be linked to the database so it can be analysed by other departments in the team, for example.
- The system is displayed using a prototype flow layout framework. Panels are resized on a single form, and can be moved to additional forms if required. The layout system is fully customisable and panels can be re-arranged through click-drag commands in the window.
- The system will display strategy and race simulations on a graph. This graph has adjustable axes and events can be executed on the graph to alter the strategies and the outcome of the race.

### Further Information

Refer to the user manual for information on how to use the system day-to-day, and how to modify system settings. The user manual also contains information on how to update the system between seasons, and provides troubleshooting information and installation instructions.

Refer to the ‘Final User Requirements’ on page 45 for information regarding the capabilities of the system. These requirements can be read in conjunction with the code to get a detailed breakdown of how the system works.

Finally, the structure charts shown in the design section give a good demonstration of the flow of data and control through the system. Many of the methods will share a name with the structure charts, and these should be used as a reference, along with the class inheritance listing, and the description of the modular structure of the system.

### Detailed Algorithms

See the ‘pseudocode’ part of the design section for detailed explanations of the construction of algorithms central to the project.

### Annotated listings

See the program listing for this. The system is extensively commented and is designed to be readable so units should be self-documenting at every level.

## Procedure/function and variable lists

### Commented Routines

All of the routines in the system are commented where necessary, and should be clear enough to be self-documenting. To a large extent, parameters, fields, and methods are entirely understandable through inspection. However, some of the finer aspects of the logic in the more complex methods are explained with comments.

Important routines are also commented using the Visual Studio framework, so that a tooltip explaining their purpose, return data, and parameters appears when they are referenced in the system.

For example, in my unit testing methods, I call the method ‘SetStintLengths’ on an instance of the Strategy class. When I hover over the function call, the following tool tip appears:

```
strategy.SetStintLengths(raceLaps, primeStints, optionStints);
void Strategy.SetStintLengths(int lapsInRace, int noOfPrimeStints, int noOfOptionStints)
Sets the optimum prime and option stint lengths for the stint
```

When writing the function call, a similar tool tip appears:

```
strategy.SetS
Assert.AreEqual(expectedPrimeLength, sti
Assert.AreEqual(expectedOptionLength, sti
void Strategy.SetStintLengths(int lapsInRace, int noOfPrimeStints, int noOfOptionStints)
Sets the optimum prime and option stint lengths for the stint
```

Documenting the code in this way means that both during inspection and when writing the code, it is clear what the effect of any method will be.

When I move on to setting the parameters for this routine, the tool tip is changed slightly, to include an explanation of the parameters required.

```
strategy.SetStintLengths()
void Strategy.SetStintLengths(int lapsInRace, int noOfPrimeStints, int noOfOptionStints)
Sets the optimum prime and option stint lengths for the stint
lapsInRace: The number of laps in the race being simulated
```

This information is obtained by using the following code before the start of the routine:

```

/// <summary>
/// Sets the optimum prime and option stint lengths for the stint
/// </summary>
/// <param name="lapsInRace">The number of laps in the race being
simulated</param>
/// <param name="noOfOptionStints">The number of stints to be completed on the
prime tyre</param>
/// <param name="noOfPrimeStints">The number of stints to be completed on the
option tyre</param>
public void SetStintLengths(int lapsInRace, int noOfPrimeStints, int
noOfOptionStints)

```

### Class Summaries

All classes are given a summary as for the routine above. This summary is similar to the summary given in the inheritance list. It helps the developer to understand the exact purpose of the class, and therefore if new methods are to be written it will guide the location of the method and the type of data it requires.

An example is given below:

```

/// <summary>
Contains a set of lines bounding the panels on the form, used for re-sizing the
panels on a window flow panel. Contains methods for adjusting the lines when
panels are added, and methods for finding the closest and furthest lines,
allowing panels to be resized.
/// </summary>
class LayoutLines

```

The layout lines class generates a network of lines on the window flow panel. The network of lines surrounds all of the panels on screen when they are initially added, and then panels can re-size until, for example, the lines at the top of the flow panel match the lines at the bottom of the flow panel. At this point there is no remaining space for a content panel to size vertically, and the flow layout has occurred.

The comment means that a developer can instantly understand the purpose of any time where LayoutLines is used, and also understand the purpose of function calls on an instance of the class.

### Self-documentation

Most important methods are summarised this way. However, it is not necessary where the code is self-explanatory. The following routine, for example, does not require much explanation, especially when the functions called within it have their own explanations. It would be inefficient use of time to write comments for this code.

```

public void ChangeStintTyreType(int stintToChange, TyreType newTyreType)
{
    listOfStints[stintToChange].tyreType = newTyreType;

    int lapsThroughRace = listOfStints[stintToChange].startLap;
    listOfStints[stintToChange] = PopulateSingleStint(Stints[stintToChange],
ref lapsThroughRace);

    UpdateStrategyParameters();
    MyEvents.OnStrategyModified(this.Driver, this, false);
}

```

### Important Routines

For this section, details of some of the important routines in the system are given here.

They are copied in as found in the file, with the purpose, input and output variables, and any important warning described in the comments.

### Data Controller

```

/// <summary>
/// Controls the processing of data loaded from a timing data file.
/// </summary>
/// <param name="data">The complete file contents</param>
/// <param name="fileType">The type of data contained by the file</param>
public void ProcessData(string data, TimingDataType fileType)

/// <summary>
/// Gets the type of data that is to be loaded from the given file type
/// </summary>
/// <param name="data">The loaded data string that will be used to generate
information</param>
/// <param name="fileType">The type of file that was loaded</param>
/// <returns>A populated instance of the correct class for the type of data
that was passed</returns>
ISessionData GetDataType(string data, TimingDataType fileType)

```

### Timing Data

```

/// <summary>
/// Gets the full file name with extension for the given session, track, and
data type
/// </summary>
/// <param name="sessionNumber">The session to find the file for</param>
/// <param name="raceIndex">The index of the race to find the file for</param>
/// <param name="dataType">The type of data being processed</param>
/// <returns>The complete file name with extension for locating the PDF file
containing the required data.</returns>
public string GetFileName(int sessionNumber, int raceIndex, TimingDataType
dataType)

/// <summary>
/// Checks if the line contains a driver's name and number, and if it is a
valid combination
/// </summary>
/// <param name="testLine">The line to check for names</param>
/// <returns>True if the line contains a driver name and matching
number</returns>
public static bool NameCheck(string testLine)

/// <returns>The file path of the directory containing the required
file</returns>

```

```
public string GetTimingDataDirectory(int raceIndex)
```

## Driver

```
/// <summary>
/// Creates a new instance of the Driver class
/// </summary>
/// <param name="passDriverIndex">The index of the driver to be created</param>
/// <param name="properties">A string array containing the driver properties,
/// loaded
/// from the driver text file.</param>
public Driver(int passDriverIndex, string driverName, string driverTeam, int
number, Color passColour)

/// <summary>
/// Initialises a driver with specified parameters.
/// </summary>
/// <param name="parameters">An array of 8 parameters to be loaded</param>
public Driver(float[] parameters)

/// <summary>
/// Loads all drivers from the database
/// </summary>
/// <param name="numberOfDriversInSeason">A variable to which the number of
/// drivers found is assigned</param>
/// <returns>The populated array of drivers</returns>
public static Driver[] InitialiseDrivers(out int numberOfDriversInSeason)

/// <summary>
/// Gets the difference between the prime and option tyre pace for this driver.
/// </summary>
/// <returns>The fastest option tyre lap - fastest prime tyre lap across all
/// sessions</returns>
public float GetTyreDelta()

/// <summary>
/// Calculates the effect of fuel for this driver
/// </summary>
/// <returns>The time loss per kilogram of fuel carried in the car</returns>
public float GetFuelEffect()

/// <summary>
/// Calculates the degradation per lap on the prime tyre
/// </summary>
/// <returns>The seconds per lap loss from new on the prime tyre</returns>
public float GetPrimeDegradation()

/// <summary>
/// Calculates the fastest lap from the race weekend, representing the low fuel
/// pace
/// of the driver
/// </summary>
/// <returns>The driver's fastest lap from the race weekend</returns>
public float GetLowFuelPace()

/// <summary>
/// Averages all relevant driver parameters across their teams to improve
/// data accuracy
/// </summary>
public static void AverageBetweenTeammates()

/// <summary>
/// Calculates the optimum strategy for the to use to complete the race
/// </summary>
```

```
/// <param name="driverIndex">The index of the driver being calculated</param>
/// <param name="trackIndex">The index of the track that the race will be run
on</param>
/// <returns>The optimised strategy with lap times and pit stops
calculated</returns>
public Strategy OptimiseStrategy(Driver driver, int trackIndex)
```

## Functions

```
/// <summary>
/// Sorts an array of type T
/// </summary>
/// <typeparam name="T">The type of elements in the array to be
sorted</typeparam>
/// <param name="Array">An array of elements to be sorted</param>
/// <param name="first">The location at which to commence the sort</param>
/// <param name="last">The location at which to end the sort</param>
/// <param name="Comparer">A comparing function for two types in the array.
/// a > b produces an ascending sort.
/// The comparer should not include the 'equal to' statement.</param>
public static void QuickSort<T>(ref T[] Array, int first, int last, Func<T, T,
bool> Comparer)

/// <summary>
/// Initialises a 2D array with every element at the specified value
/// </summary>
/// <typeparam name="T">he type of the array</typeparam>
/// <param name="columns">The width of the array</param>
/// <param name="rows">The length of the array</param>
/// <param name="valueToSet">The value to set all elements to</param>
/// <returns>The populated array of values</returns>
public static T[,] InitialiseArrayWithValue<T>(int columns, int rows, T
valueToSet)

/// <summary>
/// Opens a dialog box with the specified message and caption.
/// </summary>
/// <param name="message">The message to be displayed to the user, identifying
the error and
/// providing information on how to fix it.</param>
/// <param name="caption">The message to be displayed in the header of the
page</param>
/// <returns>True if the 'yes' button is clicked, else false</returns>
public static bool StartDialog(string message, string caption)

/// <summary>
/// Controls the calculation all of the pace parameters for each driver
/// </summary>
/// <param name="raceIndex">The race index on which the race is being
run</param>
public static void CalculatePaceParameters(int raceIndex)

/// <summary>
/// Optimises the strategies of all drivers.
/// Sets the driver's strategy to the optimised strategy.
/// </summary>
public static void OptimiseAllStrategies(int raceIndex)
```

## Pit Stop

```
/// <summary>
/// Updates timings in a race when a pit stop is made
/// </summary>
```

```

    ///<param name="strategies">The array of strategies representing a
    race</param>
    public void SimulateStop(ref Strategy[] strategies)

    ///<summary>
    /// Updates the positions when a pit stop is made using a simple insert sort.
    /// Caution, O(n) for the size of the race.
    ///</summary>
    ///<param name="strategies">The list of strategies representing a race</param>
    ///<param name="pitStops">The list of pit stops to process</param>
    ///<param name="trackIndex">The index of the track on which the race is taking
    place</param>
    public static void UpdateRacePositionsAfterPitStop(ref Strategy[] strategies,
    List<PitStop> pitStops, int trackIndex)

    ///<summary>
    /// Gets or sets the position from which a driver makes the pit stop
    ///</summary>
    public int Position

    ///<summary>
    /// Gets the lap number on which a driver makes a pit stop
    ///</summary>
    public int LapNumber

```

## Race

```

    ///<summary>
    /// Sets up a race and updates the graph displayed on screen
    ///</summary>
    ///<param name="TrackIndex">The index of the track on which the race is taking
    place</param>
    ///<param name="Strategies">The list of strategies representing drivers who
    are to take part in the race</param>
    ///<param name="AssociatedForm">The form on which the graph is to be
    displayed</param>
    public Race(int TrackIndex, Strategy[] Strategies, MainForm AssociatedForm)

    ///<summary>
    /// Sorts the strategies by pace to simulate a qualifying setup.
    ///</summary>
    public void SetupGrid()

    ///<summary>
    /// Runs a race simulation and outputs the results to a graph and to file.
    ///</summary>
    public void SimulateRace()

    ///<summary>
    /// Gets the probability that an overtake will occur
    ///</summary>
    ///<param name="paceDelta">The pace of the first car - pace of second
    car</param>
    ///<param name="speedDelta">Speed of first car - speed of second car</param>
    ///<param name="totalDelta">Total time gap between the two cars</param>
    ///<param name="requiredPaceDelta">Required pace delta to make an overtake
    possible</param>
    ///<param name="requiredSpeedDelta">Required speed delta to make an overtake
    possible</param>
    ///<returns>The probability between 0 and 1 that an overtake occurs</returns>
    public float GetOvertakeProbability(float paceDelta, float speedDelta, float
    totalDelta, float requiredPaceDelta, float requiredSpeedDelta)

```

```
//<summary>
```

```

/// Restarts the race simulation when parameters have been changed
/// </summary>
/// <param name="graph">The new graph which is output when the race simulation
has been completed</param>
/// <param name="strategies">The list of strategies to be used in the race
simulation</param>
public void RestartSimulation(out CumulativeTimeGraph graph, Strategy[]
strategies)

/// <summary>
/// Reads the settings data from a file
/// </summary>
public void LoadData()

/// <summary>
/// Writes the settings data to a file
/// </summary>
public void WriteSettingsData()

/// <returns>The average time reduction per lap due to any factor except track
evolution for the stints</returns>
public float AverageDegradation()

```

### Strategy

```

/// <summary>
/// Starts a strategy from driver data and stint information and optimises the
strategy
/// </summary>
/// <param name="_noOfStints">The number of stints to complete</param>
/// <param name="_primeStints">The number of prime tyre stints to be
completed</param>
/// <param name="_driverIndex">The index of the driver who is completing the
stint</param>
public Strategy(int _noOfStints, int _primeStints, Driver driver)

/// <summary>
/// Starts a strategy linked to a driver with a list of pre-determined stints
to load
/// </summary>
public Strategy(Driver driver, List<Stint> _stints)

/// <summary>
/// Copy constructor for the strategy class.
/// </summary>
public Strategy(Strategy s)

/// <summary>
/// Sets the optimum prime and option stint lengths for the stint
/// </summary>
/// <param name="lapsInRace">The number of laps in the race being
simulated</param>
/// <param name="noOfOptionStints">The number of stints to be completed on the
prime tyre</param>
/// <param name="noOfPrimeStints">The number of stints to be completed on the
option tyre</param>
public void SetStintLengths(int lapsInRace, int noOfPrimeStints, int
noOfOptionStints)

/// <summary>
/// Sets the total strategy time and the list of lap times used for the
strategy,
/// and calculates the number of stints in the strategy.
/// Sorts the pit stops using a quicksort.

```

```

/// </summary>
public void UpdateStrategyParameters()

/// <summary>
/// Populates the stints of the strategy with lap times
/// Populates the list of pit stops
/// </summary>
void PopulateAllStints()

/// <summary>
/// Populates a stint with lap times
/// </summary>
/// <param name="lapsThroughRace">The race lap on which the stint will start.
It is updated within the method</param>
/// <returns>The populated stint</returns>
Stint PopulateSingleStint(Stint stintToPopulate, ref int lapsThroughRace)

/// <summary>
/// Changes the length of a stint in the strategy
/// </summary>
/// <returns>The full list of stints in the new strategy</returns>
public List<Stint> ChangeStintLength(int stintToChange, int newLength)

```

## Track

```

/// <summary>
/// Creates a new instance of the track class
/// </summary>
/// <param name="Name">The name of the track (normally the country)</param>
/// <param name="Laps">The laps in the race</param>
/// <param name="fuelConsumption">The fuel use in kg/lap for the track</param>
/// <param name="_pitStopLoss">The time loss due to a pit stop</param>
/// <param name="index">The zero-based position in the year of the race</param>
public Track(string Name, int Laps, float fuelConsumption, float _pitStopLoss,
int index)

```

## Content Tab Control

```

/// <summary>
/// Adds a panel to the tab control and re-draws the control.
/// </summary>
/// <param name="PanelToAdd">The panel to be added to the control.</param>
public void AddPanelToTabControl(MyPanel PanelToAdd)

/// <summary>
/// Removes a panel from the content tab control, triggers the PanelClosed
event and re-draws the control.
/// </summary>
/// <param name="PanelToRemove">The panel to be removed from the tab
control</param>
public void RemovePanelFromControl(MyPanel PanelToRemove)

```

## Drag Drop Controller

```

/// <summary>
/// Starts the dynamic drag-drop process on the form by displaying the drag-
drop interface.
/// Events are subscribed to so that when the panel is clicked again the panel
is dropped.
/// </summary>
/// <param name="panelBeingDragged">The panel that has been selected</param>
/// <param name="visibleControls">A list of all visible panels on the
form</param>
/// <param name="startLocation">The initial click location</param>

```

```

/// <param name="dockPoints">A dictionary of the available points the panel can
be docked to on the form</param>
public void StartDragDropLayout(MyPanel panelBeingDragged, List<MyPanel>
visibleControls, Point startLocation, Dictionary<DockTypes,Point> dockPoints)

/// <summary>
/// Unsubscribes from events and hides the layout panel, returning to the
original window state.
/// </summary>
void FinishLayout()

/// <summary>
/// Drops the panel at the nearest dock point to the point where it has been
dropped.
/// </summary>
/// <param name="LocationToDrop">A point representing the coordinates of the
location that the panel was dropped at</param>
void DropPanel(Point LocationToDrop)

```

### Line

```

/// <summary>
/// Creates a new instance of the line class that is either horizontal or
vertical
/// </summary>
/// <param name="start">The start location of the line, on the axis parallel to
the line direction</param>
/// <param name="end">The end location of the line, on the axis parallel to the
line direction</param>
/// <param name="staticPosition">The location of the line on the axis
perpendicular to the line direction</param>
/// <param name="direction">The direction in which the line is oriented</param>
public Line(int start, int end, int staticPosition, LineDirection direction)

/// <summary>
/// Copy constructor for the line class
/// </summary>
public Line(Line l)

```

### Layout Lines

```

/// <summary>
/// <para>Finds the indices of lines that are intersected by edges of a
panel.</para>
/// <para>These lines are the lines that must be re-sized to allow the panel to
re-size in the given direction</para>
/// </summary>
/// <param name="panelToBeSized">The control that is to be re-sized on the
form</param>
/// <param name="directionToLookIn">The direction the panel is to be resized
in</param>
/// <returns>An array of two indices, in ascending order, of the indices of the
lines that restrict panel sizing</returns>
int[] GetIndicesOfLinesRestrictingPanel(Control panelToBeSized, Locations
directionToLookIn)

/// <summary>
/// Inserts a panel into the layout lines system, and updates the lines around
it.
/// </summary>
/// <param name="p">The control to add to the layout lines system</param>
/// <param name="DockType">The dock type of the control that is added to the
system</param>
public void AddPanel(Control p, DockTypes DockType)

```

```

/// <summary>
/// Gets the maximum amount that the panel's size can be increased in the
selected location
/// </summary>
/// <returns>The number of pixels that the panel can be increased in size
by</returns>
public int GetSizeChangeAllowed(Control p, Locations location)

```

## Main Form

```

/// <summary>
/// Sets up the header of the form with minimise, maximise, and close buttons.
/// </summary>
void SetupHeader()

/// <summary>
/// <para>Alters the form window state after the 'restore' button is
clicked.</para>
/// <para>Changes the button image to reflect the current window state.</para>
/// </summary>
void SetButtonImages(object sender, EventArgs e)

```

## Main Form IO Controller

```

/// <summary>
/// Sets up the initial set of controls on the form when it is initialised.
/// </summary>
public void SetupControls()

/// <summary>
/// Adds a control to the form.
/// </summary>
/// <param name="ControlToAdd">The IDockableControl to add to the form</param>
public void AddPanel(IDockableControl ControlToAdd)

/// <summary>
/// Removes a control from the form.
/// </summary>
/// <param name="ControlToRemove">The IDockableControl to remove from the
form</param>
public void RemovePanel(IDockableControl ControlToRemove)

/// <summary>
/// Adds the content tab control to the form.
/// </summary>
/// <param name="MainTabControl">The instance of a ContentTabControl to add to
the form</param>
public void AddContentTabControl(ContentTabControl MainTabControl)

/// <summary>
/// Adds a panel to the content tab control
/// </summary>
/// <param name="ControlToAdd">The panel to add within the main tab
control</param>
void AddContentPanel(MyPanel ControlToAdd)

/// <summary>
/// Removes a panel from the content tab control
/// </summary>
/// <param name="ControlToRemove">The panel to remove from the main tab
control</param>
void RemoveContentPanel(MyPanel ControlToRemove)

```

## My Context Menu

```

/// <summary>
/// Selects the correct buttons in the context menu based on panel properties
/// </summary>
public void SetCheckButtons()

/// <summary>
/// Handles a click event on a context menu button
/// </summary>
void ContextMenuClick(MyToolStripButton b)

```

## My Panel

```

/// <summary>
/// <para>Starts a new instance of a panel to be displayed and resized dynamically
/// on a window flow panel.</para>
/// <para>Contains methods for setting up and manipulating the panels.</para>
/// <para>After the panel is constructed, 'set panel properties' must be called.</para>
/// </summary>
/// <param name="width">The minimum width of the panel to be displayed</param>
/// <param name="height">The minimum height of the panel to be displayed</param>
/// <param name="title">The text to display in the header of the panel</param>
/// <param name="ParentForm">The form on which the panel is to be displayed</param>
/// <param name="Icon">A png file path representing the icon to be displayed in the header of the panel</param>
public MyPanel(int width, int height, string title, MainForm ParentForm, Image Icon)

/// <summary>
/// Sets the panel's layout properties to the specified values.
/// </summary>
public void SetPanelProperties(DockTypes dockType, AutosizeTypes autosizeType, FillStyles fillStyle, Size size)

/// <returns>True if the dock type is associated with the top of the panel</returns>
public static bool IsDockedAtTop(DockTypes dockType)

/// <returns>True if the dock type is associated with the left of the panel</returns>
public static bool IsDockedAtLeft(DockTypes dockType)

```

## My Toolbar

```

/// <summary>
/// Adds the functionality to control a panel from the toolbar
/// </summary>
/// <param name="panelIndex">The index of the panel to add with respect to the form</param>
public void AddPanel(int panelIndex)

/// <summary>
/// Removes the show panel button for a panel from the toolbar
/// </summary>
/// <param name="panelIndex">The index of the panel to remove</param>
public void RemovePanel(int panelIndex)

```

## Window Flow Panel

```
/// <summary>
```

```

/// Should be called after panels have been added to the window requiring it to
/// be laid out again
/// </summary>
public void FinishedAddingPanels()

/// <summary>
/// Physically adds a control to the panel
/// </summary>
/// <param name="controlToAdd"></param>
/// <param name="notAlreadyOnForm"></param>
public void AddControl(IDockableControl controlToAdd, bool notAlreadyOnForm)

/// <summary>
/// Removes a control from the panel and therefore the form.
/// </summary>
/// <param name="panelToRemove">The control to remove from the form</param>
public void RemoveControl(IDockableControl panelToRemove)

/// <summary>
/// Lays out the controls dynamically
/// Adds the controls to their respective lists for FillStyle
/// Adds the controls in order so that their maximum size is observed
/// Then adds all other controls using the layout lines to size the panels
/// </summary>
void RepeatLayout()

/// <summary>
/// Resizes all panels on the form using the layout lines
/// All panels are displayed at the maximum size possible
/// </summary>
void ResizeUsingLayoutLines()

/// <summary>
/// Checks if the AutoSizeType for a panel allows it to resize in the specified
/// direction
/// </summary>
/// <param name="direction">The direction the panel is to be sized in</param>
/// <param name="controlToResize">The control to be resized</param>
/// <returns>True if the panel can be resized in the specified
/// direction</returns>
static bool AutosizeTypeAllowsResize(Locations direction, IDockableControl
controlToResize)

```

## Axes Window

```

/// <summary>
/// Populates the text boxes on the panel with data from the given axes
/// </summary>
void PopulateAxesBoxes(axisParameters horizontalAxis, axisParameters
verticalAxis, NormalisationType normalisationType)

/// <summary>
/// Validates the data in the text boxes,
/// and populates the local axis data with the data from the text boxes
/// </summary>
/// <param name="incorrectValue">Returns true if any one of the data items is
/// invalid</param>
void PopulateAxisValues(ref bool incorrectValue)

```

## Info Panel

```

/// <summary>
/// Writes data to the output text box
/// </summary>

```

```

/// <param name="dataToWrite">The string to be written to the text box.
/// This should be informative about what the system is currently
processing.</param>
public void WriteData(string dataToWrite)

/// <summary>
/// Populates and adds the controls to the panel
/// </summary>
void AddControls()

```

### Settings Panel

```

/// <summary>
/// Saves the current settings data and writes it to files if it is valid
/// </summary>
void ConfirmChanges()

/// <summary>
/// Reverts the settings data to previous values
/// </summary>
void RevertChanges()

```

### Strategy Graph

```

/// <summary>
/// Draws the graph to the screen
/// </summary>
/// <param name="tracesToShow">A list of StrategyLine traces to show on the
graph</param>
/// <param name="showAllOnGraph">Represents whether all traces should be shown,
or only the traces specified in the driver select panel</param>
/// <param name="changeNormalised">Represents whether the graph should be re-
normalised on the fastest trace</param>
public void DrawGraph(List<StrategyLine> tracesToShow, bool showAllOnGraph,
bool changeNormalised)

/// <summary>
/// Gets a list of normalised strategy lines based on the driver selected for
normalisation
/// </summary>
List<StrategyLine> GetTraces(List<StrategyLine> originalTraces, int
normalisedDriverIndex)

/// <summary>
/// Normalises all of the strategy lines in the list of strategy lines passed
to the method
/// </summary>
/// <param name="bestTime">The time of the fastest driver</param>
/// <param name="laps">The number of laps in the race</param>
/// <param name="originalTraces">The traces to normalise</param>
/// <returns>A list of strategy lines which are normalised using the selected
normalisation method</returns>
List<StrategyLine> NormaliseAllLines(float bestTime, float laps,
List<StrategyLine> originalTraces)

/// <summary>
/// Normalises a strategy on an average time
/// </summary>
/// <param name="line">The line to normalise</param>
/// <param name="adjustPerLap">The amount by which to adjust the trace each
lap</param>
/// <returns>The new normalised line</returns>
StrategyLine GetNormalisedStrategyLine(StrategyLine line, float adjustPerLap)

```

```

/// <summary>
/// Normalises a strategy line on every lap.
/// </summary>
/// <param name="line">The line to normalise</param>
/// <param name="normalisationTrace">The trace used for normalisation</param>
/// <returns>The new normalised line</returns>
StrategyLine GetNormalisedStrategyLine(StrategyLine line, StrategyLine
normalisationTrace)

/// <summary>
/// Draws alll of the traces that are to be drawn to the screen
/// </summary>
/// <param name="g">The graphics to use to draw the lines on screen</param>
/// <param name="upToLap">The number of laps to show on the graph</param>
void DrawLines(Graphics g, int upToLap)

/// <summary>
/// Converts a panel coordinate location into a lapDataPoint
/// </summary>
/// <param name="point">The point to be converted</param>
/// <param name="HorizontalAxis">The horizontal axis used for
conversion</param>
/// <param name="VerticalAxis">The vertical axis used for conversion</param>
/// <returns>The lapDataPoint closest to the specified point</returns>
public static lapDataPoint GetPositionOfPoint(Point point, axisParameters
HorizontalAxis, axisParameters VerticalAxis)

/// <summary>
/// Sets the number of laps the graph will display
/// </summary>
/// <param name="value">The number of laps to set the graph to display</param>
public void SetRaceLaps(int value)

```

## Strategy Viewer

```

/// <summary>
/// When strategy modifications are completed, re-populates controls
/// </summary>
/// <param name="showAllOnGraph">Represents whether all traces should now be
shown on the graph</param>
void MyEvents_StrategyModificationsComplete(bool showAllOnGraph, bool
changeNormalisedDriver)

/// <summary>
/// Once data is loaded, displays and populates the controls on the panel
/// </summary>
void StrategyViewerData_DataLoaded()

/// <summary>
/// Displays the race stints on the panel after the stints have been modified
/// </summary>
/// <param name="driverToDisplay">The driver whose strategy is to be
displayed</param>
void DisplayStints(int driverToDisplay)

/// <summary>
/// Removes the stint panels from the panel
/// </summary>
void RemoveStintPanels()

/// <summary>
/// Shows the panels and populates with data from the stored strategies
/// </summary>
/// <param name="driverToShow">The driver whose data is being displayed</param>

```

```

void ShowStintPanels(int driverToShow)

<:/// <summary>
<:/// Creates the graph traces and calls the graph draw method to display the
strategies on a graph
<:/// </summary>
<:/// <param name="showAllOnGraph">Represents whether all traces on the graph
will be shown after the update</param>
<:/// <param name="changeNormalised">Represents whether the normalised driver
should be set to the fastest driver,
<:/// or maintained as the current driver</param>
void DrawGraph(bool showAllOnGraph, bool changeNormalised)

```

## Timing Archives

```

<:/// <summary>
<:/// Populates the list box with the lap times required
<:/// </summary>
<:/// <param name="driver">The driver to show</param>
<:/// <param name="race">The race from which to display data</param>
<:/// <param name="session">The session to display data from</param>
void LoadTimingData(int driver, int race, int session)

<:/// <summary>
<:/// Loads data into the combo boxes
<:/// </summary>
void PopulateComboBoxes()

```

## Strategy Line

```

<:/// <summary>
<:/// Draws a the line represented by the list of points using the specified
graphics
<:/// </summary>
<:/// <param name="horizontalAxis">The horizontal axis parameters</param>
<:/// <param name="verticalAxis">The vertical axis parameters</param>
<:/// <param name="g">The graphics to use to draw the line</param>
<:/// <param name="upToLap">The last lap to display</param>
public void DrawLine(axisParameters horizontalAxis, axisParameters
verticalAxis, Graphics g, int upToLap)

<:/// <summary>
<:/// Gets the ordinate on the specified axis that is represented by the locating
value
<:/// </summary>
<:/// <param name="locator">The point that is to be displayed</param>
<:/// <param name="axis">The axis on which the ordinate is required</param>
<:/// <returns>The value of the ordinate that the point is to be drawn
at</returns>
int GetPointOrdinate(float locator, axisParameters axis)

```

## Show Driver Check Box

```

<:/// <summary>
<:/// Creates a new instance of a ShowDriverCheckBox, associated to the specified
driver index
<:/// </summary>
<:/// <param name="driverIndex">The driver index associated to the
control</param>
public ShowDriverCheckBox(int driverIndex)

```

## Normalised Driver Radio Button

```

<:/// <summary>

```

```

/// Creates a new instance of a NormalisedRadioButton, associated with the
/// specified driver index
/// </summary>
/// <param name="driverIndex">The driver index associated with the
/// control</param>
public NormalisedRadioButton(int driverIndex)

```

### My Tool Tip

```

/// <summary>
/// Adds a tool tip with the specified text to a specified control
/// </summary>
/// <param name="control">The control to provide the tool tip for</param>
/// <param name="text">The text to display in the tool tip</param>
public MyToolTip(Control control, string text)

```

### Parameter Text Box

```

/// <summary>
/// Enum containing options for the way the control will be validated
/// </summary>
internal enum ValidationMethod { LessThan, GreaterThan, Between };

/// <summary>
/// Sets the text box properties including color and text
/// </summary>
/// <param name="value">Parameter value to display</param>
/// <param name="IsDefault">Value representing whether the value is equal to
/// the default value</param>
/// <param name="IsAnomaly">Value representing whether the value is an
/// anomaly</param>
public void SetProperties(float value, bool IsDefault, bool IsAnomaly)

/// <summary>
/// Validates the value that is intended for the cell
/// </summary>
/// <param name="value">The intended cell contents</param>
/// <returns>True if the value can be accepted</returns>
bool Validate(float value)

```

### Text Changed Event Args

```

/// <summary>
/// Creates new TextChangedEventArgs
/// </summary>
/// <param name="t">The text box that fired the original event</param>
public TextChangedEventArgs(ParameterTextBox t)

```

### Stint Button Layout Panel

```

/// <summary>
/// Handles the button clicked events on any of the buttons within the control.
/// </summary>
/// <param name="controlNumber">The control number that fired the event</param>
void ButtonClicked(int controlNumber)

```

### Stint Panel

```

/// <summary>
/// Occurs when the user moves away from the text box containing the stint
/// length
/// </summary>
void stintLength_LostFocus(object sender, EventArgs e)

/// <summary>
/// Occurs when the user changes the tyre type selected for this stint.

```

```
/// </summary>
void tyreType_SelectedIndexChanged(object sender, EventArgs e)
```

### Results Panel

```
/// <summary>
/// Creates a new instance of a ResultsPanel, linked to the specified driver
and strategy.
/// </summary>
public ResultsPanel(int driverIndex, Strategy thisStrategy)
```

### My Events

```
/// <summary>
/// Fires the AxesModified event
/// </summary>
/// <param name="horizontalAxis">The new horizontal axis</param>
/// <param name="verticalAxis">The new vertical axis</param>
/// <param name="normalisation">The new graph normalisation type</param>
/// <param name="UserModified">True if the user has forced the change; false if
it is computer generated</param>
public static void OnAxesModified(axisParameters horizontalAxis, axisParameters
verticalAxis, NormalisationType normalisation, bool UserModified)

/// <summary>
/// Fires the SettingsModified event
/// </summary>
public static void OnSettingsModified()
```

### Pace Parameter Data

```
/// <summary>
/// If data has been modified and not updated, displays a warning message
warning the user of this
/// before the data is used in processing.
/// </summary>
void NotifyIfModified()

/// <summary>
/// Populates the locally stored data from the driver data held within the
program.
/// </summary>
void PopulateDataFromDrivers()

/// <summary>
/// Populates driver pace data from a csv file selected by the user.
/// </summary>
void PopulateDataFromFile()

/// <summary>
/// Sets the formatting of the text box when its value is changed
/// </summary>
/// <param name="e">The event arguments generated from the value changed
event</param>
void SetColours(TextChangedEventArgs e)

/// <summary>
/// Returns true if the current value is the same as the default value, stored
in the settings file.
/// </summary>
public bool IsDefault(float currentValue, int dataIndex, int driverIndex)

/// <summary>
/// Returns true if the current value is outside of 25% of the default value,
stored in the settings.
```

```
/// </summary>
public bool IsAnomaly(float currentValue, int dataIndex)
```

### Panel Control Events

```
/// <summary>
/// Fires the RemoveGraphPanels event, clearing the main graph panels from the
/// screen to allow the screen to resize accordingly.
/// </summary>
public static void OnRemoveGraphPanels(MainForm form)

///<summary>Shows the version information window.</summary>
public static void OnShowVersionInfo()
```

### Strategy Viewer Data

```
/// <summary>
/// Gets a specific strategy from the list of strategies
/// </summary>
/// <param name="driverIndex">The index at which the strategy is held</param>
public Strategy GetStrategy(int driverIndex)

/// <summary>
/// Sets a locally held strategy to the specified strategy
/// </summary>
/// <param name="driverIndex">The index of the strategy to overwrite</param>
/// <param name="value">The new strategy to set as the locally held
/// strategy</param>
void SetStrategy(int driverIndex, Strategy value)
```

### Program

```
/// <summary>
/// Opens a panel in a new window
/// </summary>
/// <param name="PanelToAddToNewWindow">The panel to open in a new
/// window</param>
/// <param name="Form">The original form on which it was displayed</param>
public static void OpenInNewWindow(MyPanel PanelToAddToNewWindow, MainForm
Form)

/// <summary>
/// Loads driver timing data from the saved files automatically.
/// </summary>
/// <param name="raceIndex">The index of the race data to load</param>
public static void PopulateDriverDataFromFiles(int raceIndex)
```

## User Manual

The user manual below is designed to give potential users of the system an introduction into how to use the system and the type of data processing it is capable of. The document also provides information on potential errors and the fixes for these errors.

The document is designed to be readable by a person with no computing background but with experience in F1 race strategy. It provides no details on the development of the system; solely the use of the system is covered.

### Contents page

1. Introduction.....	2
2. Installation Instructions.....	3
3. Quick Start Guide .....	4
4. Detailed User Information .....	6
5. Screen Displays.....	20
6. Troubleshooting .....	23

## Introduction

*StratSim* is a custom-built tool for F1 race strategy optimisation and simulation. It provides you with the ability to analyse the pace of drivers competing in an F1 race weekend, and use this data to predict the outcome of the race.

The system also provides functionality for viewing times from previous events, and for quickly loading data that has already been input. The tool is intended only as a guide and is no guarantee of actual performance or race outcomes.

The system will use the FIA-produced timing data PDF documents to generate its data. Ensure that these are present on your computer.

See the Samples of Screen Displays section for helpful annotated guides on what to expect at each stage of using the system.

## Installation Instructions

### Downloading and Installing the System

The system will be provided on a CD/ROM for installation either over a large network from one administrator or for installation on each individual machine by a user with administrative privileges.

1. Load the CD/ROM into the disc drive of the computer. After a moment, a prompt will appear, asking you to view files from the disk. Click this option.
2. Navigate to the 'StratSim' folder and double click to open the folder.
3. Inside the StratSim folder, double click the 'StratSim' file. This will open a window asking if you want to install the program. Click install.
4. The system will open a window saying 'verifying application requirements', and then another saying 'installing StratSim'. These will disappear after about 10 seconds.
5. StratSim will then be installed on your computer, along with all of the required files for it to run.

### Local Files

The system saves settings data to a text file, and loads driver and track information from a database. These files are installed along with the program.

You will require Microsoft Access to modify the database, and a text editor to modify the settings. However, these can be altered internally.

## Quick Start Guide

Start the program from the desktop. The *StratSim* window will be displayed.

The ‘Start’ panel is displayed on the screen, allowing you to select a race from a list of races, or to input data or view timing archives.

If you have never used the system before, you will need to modify the settings. Select ‘Open’ from the toolbar, and then click ‘Settings’. There are five tabs, each of which contains important data for the functioning of the system. These boxes need to be populated – see the ‘Settings’ section of this document for more information. Once the data is all loaded, click ‘OK’. You may now hide or close the settings panel.

### Start Menu

There are now three options:

- If you wish to view the stored data about a past race weekend, select ‘Timing Archives’. A new panel will appear, showing three combo boxes. Select the race, driver, and session you wish to display, and click ‘Get Data’. The system will display timing data in the box to the right of the panel. This process can be repeated if required. The box will remain blank if there is no data displayed.
- If you have not already loaded data about the race you wish to display:
- Select ‘Data Input’. Select the race and session you wish to enter data for, and the system will automatically open the required FIA PDF timing data document.
- Within this document, select all of the text and press ‘Ctrl+c’ to copy the data. You should then close the PDF. Switch back to the system, and click ‘Confirm Data’.
- The data from the PDF file is processed and the panel updated to be ready for the next document. It will open automatically, except for qualifying, where you must select the type of data you wish to open.
- Once you have loaded all of the data, click ‘Analyse Data’. Then see the ‘Pace Parameters’ section.
- If you already have data loaded for the race you wish to view, select this race from the drop-down list.

### Pace Parameters

From the start menu, select either ‘file’ or ‘race’ from the pace parameters line. If you select file, a file dialog will open, prompting you to open a CSV file containing pace parameter data. This can be manually designed or from a previous system test. See the detailed description for more information.

If you select ‘race’, the system will automatically process the timing data you have loaded and calculate the pace parameters for every driver. Clicking ‘view’ will display these. They can be manually edited if required.

### Strategy Optimisation

From the start menu, in the strategies section, select either ‘file’ or ‘data’. As before, ‘file’ will open a file dialog, expecting a list of strategy data that has been previously loaded.

‘Data’ will optimise the strategies based on the saved pace parameters. This creates the theoretical fastest strategy for every driver. Click ‘view’ to show this on the screen. It can be manually edited too, by changing stint lengths, changing the tyre type, or using the buttons provided to re-order, add, and remove stints.

### Race Simulation

Selecting ‘view’ from the race section of the start menu will display a graph of the driver’s cumulative times throughout a race. The panel on the right containing driver names is a key to the graph and also shows time gaps and intervals on the specified lap (change this using the drop down provided).

The graph can be used to get an idea of how the race may play out, and can therefore offer tips on how to improve the race strategy from the theoretical model. Overtaking is simulated as probabilistic, and if you believe an overtake will occur, right-clicking the graph on the required driver and lap will force an overtake to happen.

## Detailed description of how to use the system

### Getting Started

Opening the application will display the main screen, with toolbars at the top and a start panel displayed.

### Main Window

In the top right of this screen there are three buttons, minimise, restore, and close. These work in the same way as in other windows applications. Note that there are no warnings when the close button is clicked; if you have unsaved data it will be lost.



### Toolbar

A toolbar is displayed at the top left of the screen, containing four buttons:



- View: This will allow you to hide and show panels on the screen.
- Startup: Quick access to a variety of system functions directly from the toolbar. These allow you to begin using the system. They will be revisited later.
- Open: This will open any panels that you require to use the system.
- Version: Opens the version information window, containing information about the system.

Further buttons will appear in the toolbar as you progress through use of the system.

### Panels

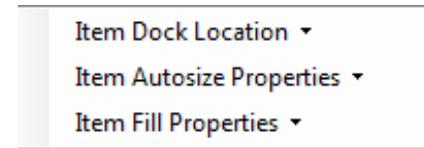
The system's visual output is displayed on panels. These are rectangular items that are displayed on the screen with a blue header and a black outline.

The title of the panel, and a representative icon, are shown in the top left corner of the panel. This gives an idea as to the purpose of the panel.



The panel, like the main window, has three buttons in the top corner. These will hide the panel (it can be shown again from the 'View' button in the toolbar), set the panel to full screen (and back again), and close the panel (it can be re-opened from the 'Open' button in the toolbar).

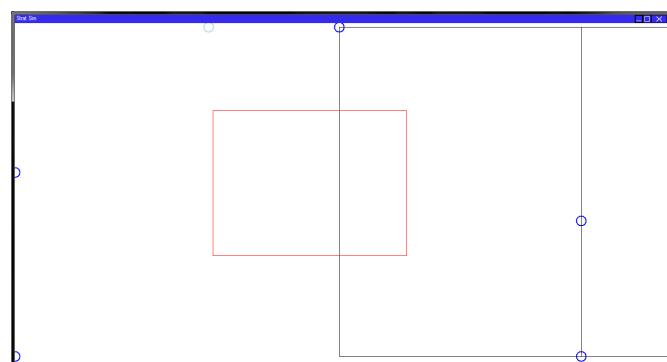
Right-clicking the header of the panel brings up a menu of options:



- Dock Type: You can select what location on the screen the panel will appear in;
- Autosize Type: Panels will automatically expand to fill the available area on the window. Use this option to set how the panel can expand.
- Fill Style: Panels can be set to automatically take up the full width or height of the screen, or full screen. Use this option to display a panel at the full width or height of the screen. To toggle away from these options, select 'None'.

By default, panels are set to give the best layout for standard operation; there is no need to set this up when the program is started.

Left-clicking the panel activates a drag-drop process. The points circled in blue are the locations at which a panel can be docked. Moving the mouse near these points will highlight the closest one, and subsequently clicking will dock the panel to this point and restore the window to normal. Clicking while no points are highlighted will open the selected panel in a new window. It will behave exactly as it did before in this window.



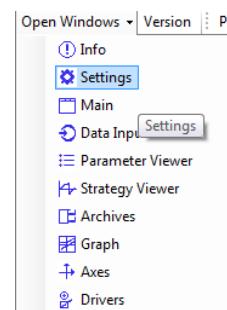
To return a panel to the original window, activate the drag-drop on the panel you want to move, then switch windows and click the background of the window you want the panel to be moved to. The panel will be moved and if necessary the window where it originated will be closed.

The main content of the panel is displayed below the header. This is where you can perform system tasks.

## Settings

### On Startup

The first time the system is used, the settings are set to pre-defined defaults and may not be appropriate for your system and folder structures. To modify the settings as required, select ‘Open Windows’ and then ‘Settings’. Within the settings panel, you can modify the settings data based on your current folder structures and preferred defaults.



Clicking ‘OK’ will save the changes, but ‘cancel’ will revert to the original data. Once ‘OK’ has been clicked, you can close the settings panel and continue to use the system.

Requirements		Time Losses	
Required Pace Delta:	-1	Time Loss (normal)	0.4
Required Speed Delta:	-1	Time loss (backmarker)	1
		Time Gap in Traffic	0.4

Select different groups of settings with the tabs.

Buttons for accepting and cancelling the changes made to the settings.

## **During System Use**

There are some settings that will need to be changed on a race-by-race basis. These include the required pace and speed deltas for overtakes to occur, and defaults for tyre degradation and delta.

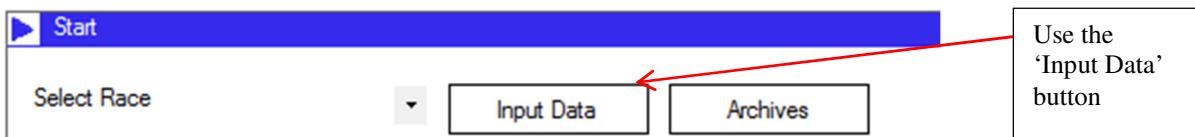
The settings can be updated at any time during use of the program. If the changes will affect a current simulation, you will be prompted that this is the case and can choose to update the current simulation in light of the changes you have made.

The settings panel is validated to ensure that the changes made to the settings will not cause the system to fail. It is therefore recommended that all changes to settings are made from within the system and not in the text file.

## Loading Timing Data

### **Step-By-Step**

1. To load data from the PDF files, select ‘Load Data’ from the start menu. A panel appears with three drop down boxes and two buttons.



2. To start, select the race from which you wish to load data, from the top drop-down. Then, select the session you wish to load data from, from the second drop-down. At this point, the third drop-down will be automatically populated, and the required PDF file will be opened.



3. If the file does not open, check that the folder path specified in the settings is correct and that the file exists.
4. Select all of the text in the PDF file, using ‘Ctrl+a’. Then use ‘Ctrl+c’ to copy the data. It will be copied to the clipboard. You can then close the PDF file.
5. Now select ‘Confirm Data’. The data from the clipboard is automatically analysed and processed, and the second drop down is moved to the next session. This opens the next PDF, if it is present.
6. When all PDFs have been loaded, clicking ‘Analyse Data’ will generate the pace parameters information by processing the data that has been loaded.
7. If an error message appears saying that the session has not been loaded, you may decide to proceed with incomplete data (the system will continue to function) or cancel, and re-select the file that is still required.
8. Once the analyse data button is clicked and the error message does not appear, or ‘OK’ is clicked in the error dialog, the pace parameters will be processed and loaded.

## Pace Parameters

The screenshot shows a software interface titled 'Pace' with a 'Parameters' tab selected. A table lists 23 drivers from 2013, grouped by team. The columns include Number, Team, Name, Top Speed, Tyre, Prime, Option, Pace, Fuel Effect, Fuel Consumption, and P2fuel. Several cells are highlighted: yellow for default values (e.g., Vettel's Pace), red for anomalies (e.g., Alonso's Pace), and blue for user-modified values (e.g., Button's Pace). Red arrows point to specific cells like Alonso's Pace and Button's Pace.

Number	Team	Name	Top Speed	Tyre	Prime	Option	Pace	Fuel Effect	Fuel Consumption	P2fuel
1	Red Bull	VETTEL	299.2	-0.54	0.254	0.342	91.083	0.02	2.5	60
2	Red Bull	WEBBER	302.3	-0.567	0.254	0.342	90.915	0.02	2.5	60
3	Ferrari	ALONSO	299.5	-0.8	0.1	0.199	91.665	0.031	2.5	60
4	Ferrari	MASSA	299.3	-0.228	0.1	0.199	91.378	0.031	2.5	60
5	McLaren	BUTTON	294.4	-0.756	0.002	0.002	91.827	0.035	2.5	60
6	McLaren	PEREZ	294.7	-0.8	0.002	0.052	91.989	0.035	2.5	60
99	Lotus	RAIKKONEN	294.7	-0.962	0.1	0.419	91.662	0.021	2.5	60
8	Lotus	GROSJEAN	295.4	-0.568	0.1	0.419	91.365	0.021	2.5	60
9	Mercedes	ROSBERG	293.9	-0.173	0.088	0.3	91.397	0.03	2.5	60
10	Mercedes	HAMILTON	297.6	-0.8	0.088	0.3	91.253	0.03	2.5	60
11	Sauber	HULKENBERG	297	-0.518	0.017	0.3	91.644	0.026	2.5	60
12	Sauber	GUTIERREZ	296.4	-1.471	0.017	0.3	92.063	0.026	2.5	60
14	Force India	DI RESTA	298.5	-0.924	0.1	0.162	91.992	0.015	2.5	60
15	Force India	SUTIL	298.2	-0.624	0.1	0.162	92.89	0.015	2.5	60
16	Williams	MALDONADO	299.5	-0.8	0.168	0.3	92.093	0.033	2.5	60
17	Williams	BOTTAS	300.5	-0.004	0.168	0.3	92.013	0.033	2.5	60
18	Toro Rosso	VERGNE	299.5	-0.757	0.144	0.3	93.06	0.027	2.5	60
19	Toro Rosso	RICCIARDO	301.8	-0.962	0.144	0.3	92.485	0.027	2.5	60
20	Caterham	PIC	297.6	-0.195	0.1	0.3	94.556	0.02	2.5	60
21	Caterham	VAN DER GARDE	296.2	-0.8	0.1	0.3	94.873	0.02	2.5	60
22	Marussia	BIANCHI	296.7	-0.8	0.1	0.3	95	0.02	2.5	60
23	Marussia	CHILTON	301	-0.8	0.1	0.3	96	0.02	2.5	60

Default values are highlighted in yellow, anomalies in red text, and values that are user-modified in blue. This makes it clear what has changed and which may need changing.

To change a parameter, simply overtype the value in the text box. This triggers an update in the PaceParameterData class associated with the panel.

## Viewing

The pace parameters panel can be displayed by clicking 'View' on the pace parameters, by completing data loading from PDFs, or by using the quick-access startup toolbar.

When the panel is displayed, it will be automatically populated with the current data. This is displayed in a table with the drivers on the left of the window, and headings along the top. Hover over the headings for a detailed description of the data held in the system.

Pace parameters are sorted by team, and they are averaged between the two drivers in a team where appropriate to improve data accuracy.

## Guide To Colour Codes

Values that are entered into the pace parameters table are colour coded to make it clear how they came about and to assist the user in modifying and updating the data to make it more reliable.

- Yellow Highlight – The value has been set to the default value because the system has not generated a reasonable value for the data item.
- Blue Highlight – The value has been modified by the user and the blue highlight shows this.
- Red Text – The value that has been entered is potentially anomalous, being more than 40% away from the default value for that data item.

## Modifying the Pace Parameters

To modify a value, select the value that is currently displayed by highlighting it, and then type in the value that you require. Clicking away from the control or pressing tab will set the value.

If the value you have entered is invalid, the system will display a warning message and reset the data to its original value. You can try again in case this was a mistype, or set the value to something that is within the valid bounds.

The selected fuel load for P2 is used to generate an approximate effect of fuel on the lap times of the car. This adjustment is subsequently applied throughout the rest of the calculations. As a result, after changing this fuel load the system will ask if you want to save this value and update the rest of the pace parameters. It is recommended that OK is clicked.

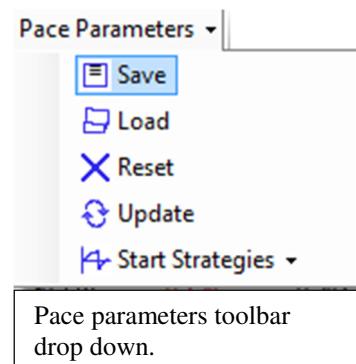
## Saving To Files

Using the toolbar, it is possible to load and save pace parameter data to files. This allows previous pace parameter setups to be loaded if required, and data to be saved and distributed using Microsoft Excel for other reporting.

To save a set of pace parameters, locate the ‘Save’ button in the Pace Parameters toolbar button. Clicking this button brings up a file dialog. It opens in the default timing data directory but this can be changed manually inside the folder structure.

Enter the name you would like to assign to the file, and click ‘OK’. This will save the file in the folder you have specified.

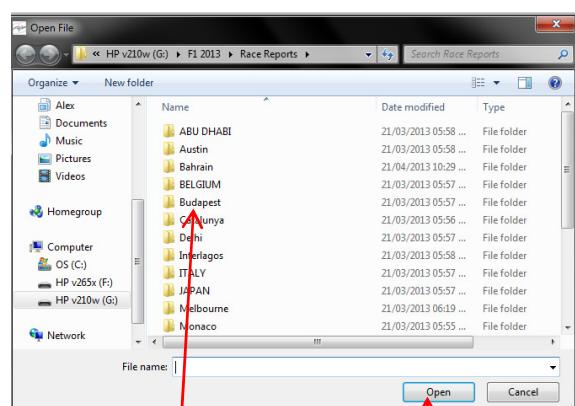
The file can now be viewed in Microsoft Excel, and distributed as a ‘.csv’ type file.



## Loading From Files

To load from files, select ‘Load’ from the toolbar. A file dialog will open. Navigate through the folders and select the file that you want to load from.

Select this file and then click ‘Open’. The system will populate the data in the parameters table with data from the file. You can now make changes to the data in the usual manner. Once data has been loaded from a file, resetting the data will reset it to the parameters specified in the file.



Select a folder from the folder list

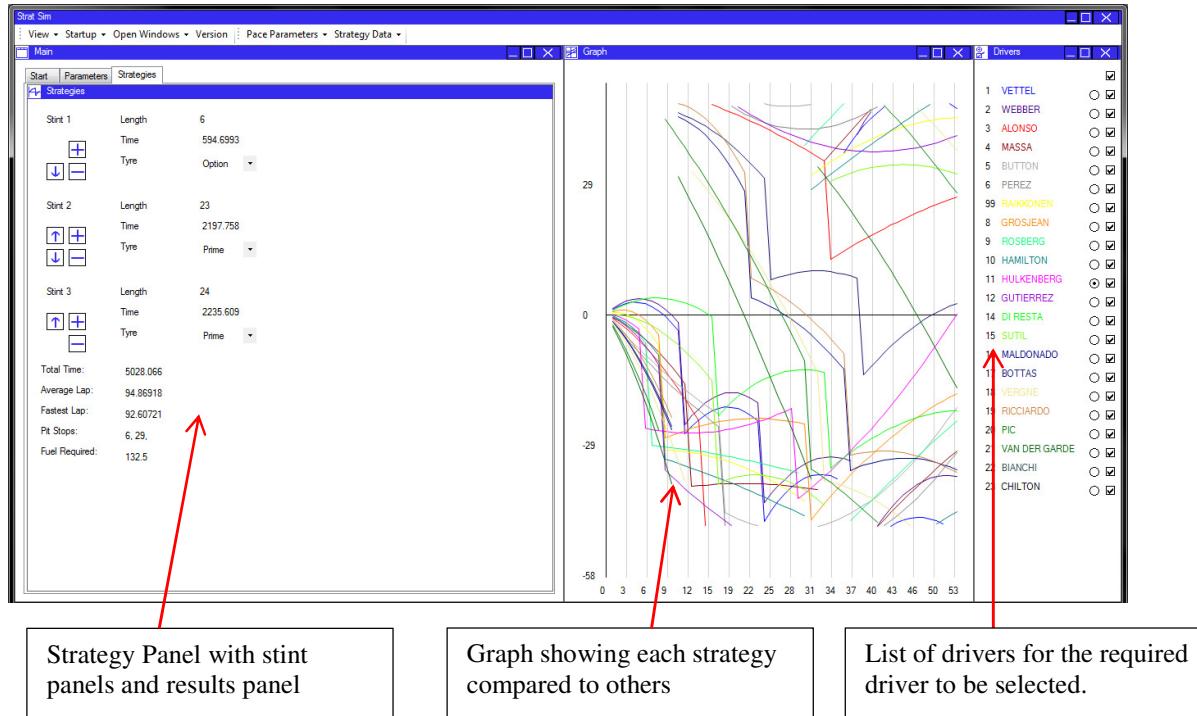
Click ‘Open’ to load the data

**Updating and Resetting Data**

Once data has been modified within the panel, it must be updated within the system before strategy optimisation takes place. This means that the ‘Update’ button in the toolbar must be pressed.

If data has been modified on the panel but you wish to discard the changes, the ‘Reset’ button in the toolbar will revert all changes on the panel back to either the last update, or to the system status when data was first loaded.

## Strategies



### **Viewing and Modifying Strategies**

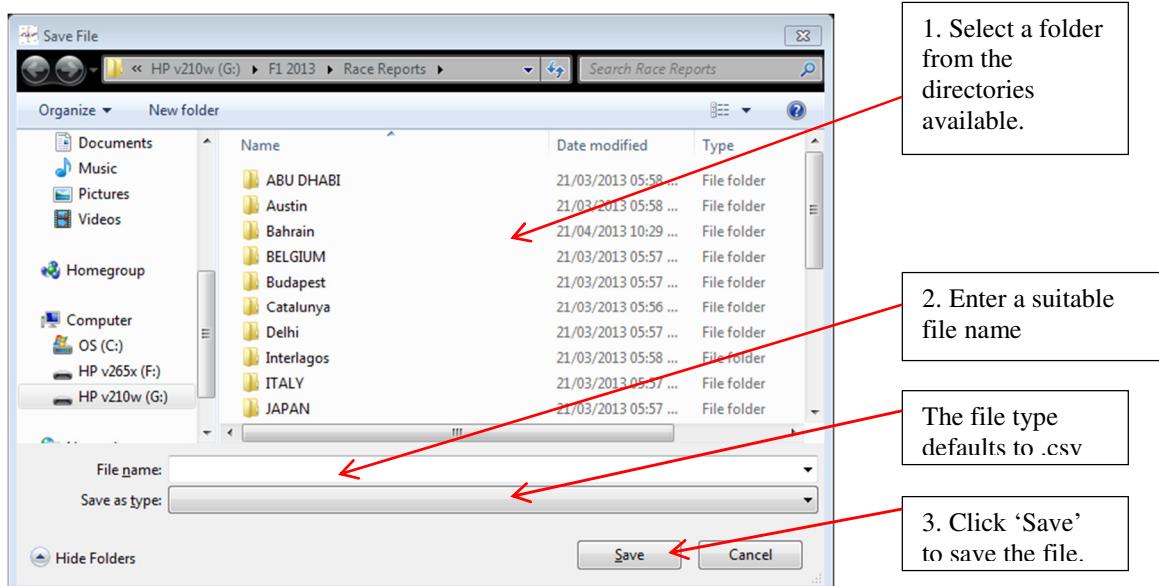
Strategies can be optimised by selecting ‘Strategies – Data’ from the start menu, or using the toolbar option provided in the Pace Parameters panel. They can be viewed by selecting ‘View Strategies’ from the start panel.

When the strategies panel is opened, the driver with the theoretical fastest strategy is displayed. Each stint in the strategy is summarised in a single panel, and the whole strategy is summarised in a panel below this. The stint panels contain controls for modifying the strategies:

1	Stint 2	3	Length	23	5
2	<span style="border: 1px solid black; padding: 2px;">↑</span> <span style="border: 1px solid black; padding: 2px;">+</span> <span style="border: 1px solid black; padding: 2px;">↓</span> <span style="border: 1px solid black; padding: 2px;">-</span>	4	Time	2197.758	
			Tyre	Prime	6

1. Up button – swaps the stint with the stint above
2. Down button – swaps the stint with the stint below
3. ‘+’ Splits the current stint, adding a new one just before the current stint.
4. ‘-’ Removes the current stint, merging the laps into the preceding or subsequent stint as appropriate.
5. Laps text box – Overtyping a new lap number in this text box will change the length of the stint.
6. Tyre type combo box – Selecting a tyre type will update the tyre type used for the stint.

A different driver can be selected using the DriverSelectPanel on the right hand side of the screen. Clicking the radio buttons next to the driver names selects this driver and shows his stints on the strategy panel. These can be modified in exactly the same way.



### Saving To Files

Strategy data can be saved to file in a similar way to the pace parameter data. This again allows it to be viewed or transferred outside of the system.

To save strategy data, use the 'Save' button in the Strategies toolbar drop down. This will open a file dialog.

Select the folder you wish to save the file in, and enter an appropriate file name, and the data will be saved into a .csv file in that location. This can then be viewed outside of the program.

### Loading From Files

Data can be loaded from files if previous work has been done to optimise the strategies. This can be done using the 'Load' button in the toolbar.

Selecting this button opens a file dialog. Select the file you require and click 'Open' and the file will be opened. The driver with the fastest overall strategy will be displayed, but all drivers can be selected and viewed or edited.

### Updating and Resetting Data

Once data has been modified within the panel, it must be updated within the system before a race simulation is run. This means that the 'Update' button in the toolbar must be pressed.

If strategy data has been modified on the panel but the changes are not required, the 'Reset' button in the toolbar will revert all changes on the panel back to either the last update, or to the system status when data was first loaded.

## Race Simulation

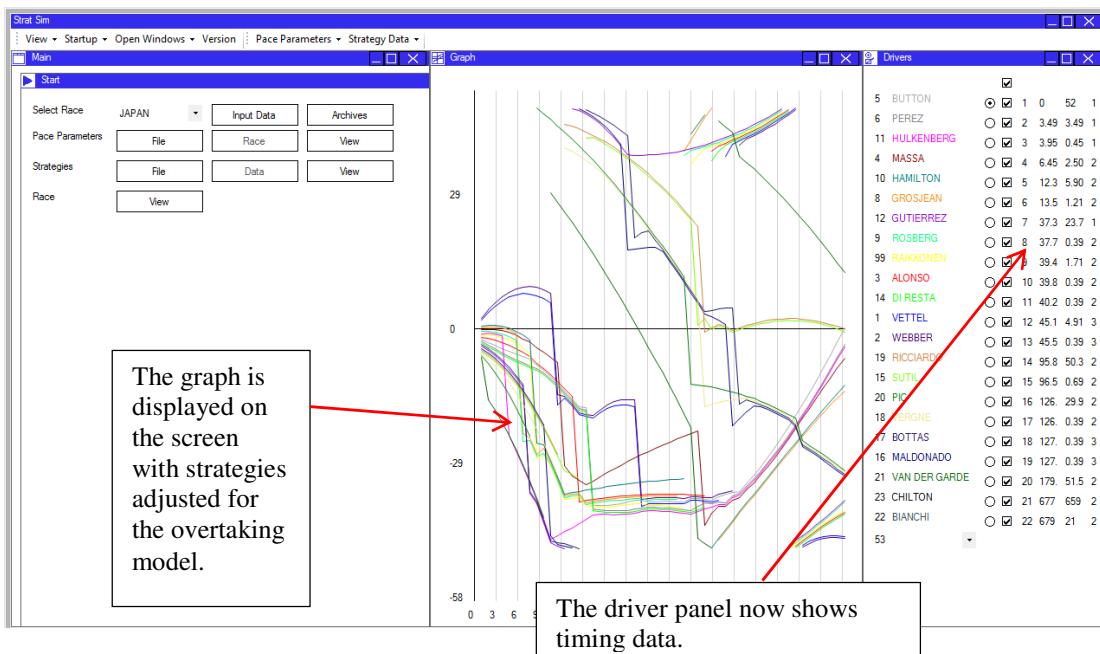
### Summary

The race simulation feature is a guide to how the race may pan out over the course of time. All strategies are simulated lap-by-lap, with pit stops, overtakes and lapped cars assessed at each stage. The simulation is not guaranteed to be accurate in real world situations and cannot simulate safety cars, incidents, or weather in the race.

### Starting and Restarting

Race simulations can be started and restarted by clicking the ‘View Race’ button in the start panel, or via the toolbar option on the Strategy Panel. The race can also be started through the ‘Startup’ toolbar option.

If there is no race running, selecting any of these options will start a new race simulation based on the current strategies loaded in the system. If a race is already running, it will re-simulate the race. This will also reset any forced overtakes that have been specified.



### Viewing

The race simulation can be viewed in two different ways. Firstly, the cumulative time graph on the screen will show a visual representation of the cars times against laps, effectively showing track position at each stage.

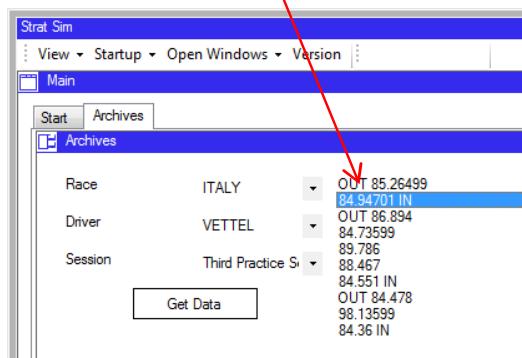
Secondly, the Driver Panel on the right will show time gaps, intervals, and positions, on the selected lap, which allows the situation of the race to be analysed in detail on each lap.

The panel is designed to mirror the standard timing screens by also providing a count of how many pit stops the drivers have had up to that point.

## Archives

The archives panel allows previously loaded timing data to be viewed within the program. To view data, select the race, session and driver you wish to view, and then click ‘Get Data’. The lap times will appear in a list box to the right. This can be repeated to view different data if required.

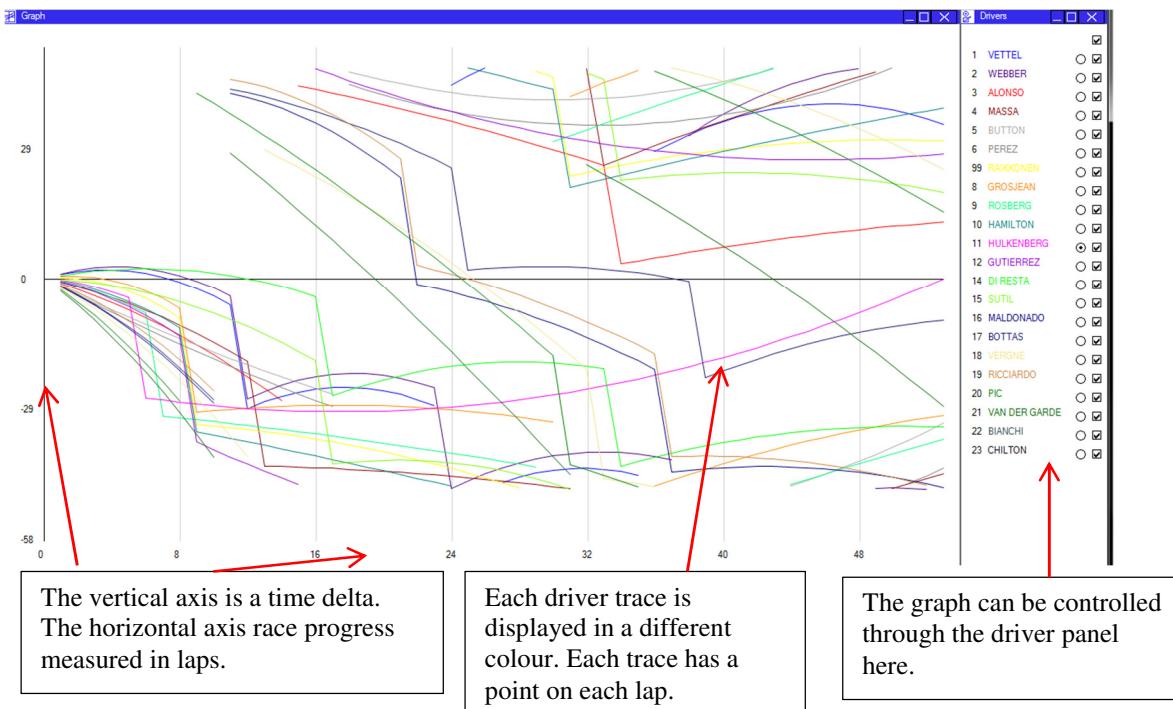
The first stint was two laps long, bookended by in- and out- laps.



The list box will contain only flying lap times. This is to eliminate in-laps and out-laps if data is analysed. These laps are replaced with ‘IN’ and ‘OUT’, to signify the end and start of stints respectively.

## Graph

A graph will be displayed on the panel when the strategies or race simulation are active. The graph displays cumulative time against laps, and is a guide to how the lap times vary through the race. It also provides a useful comparative indicator as to how one driver’s race is varying against another’s.



## Axes

The horizontal axis is laps. Each driver will have one point on each lap, representing their total cumulative time up to this point.

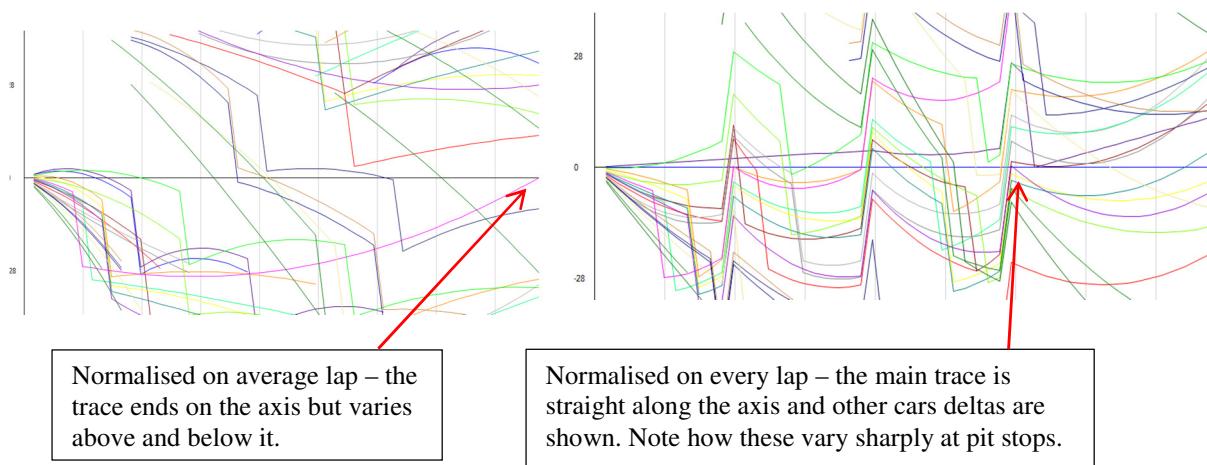
The vertical axis is a delta time. It is specifically the time difference between the driver’s actual time, and the time of the normalised driver at this point. If the normalisation type is ‘Average Lap’, the selected driver’s average lap time is used, so

the value represented is the difference between the actual cumulative time and the sum of all of the average laps up to this point.

If the normalisation type is ‘Every Lap’, the value is simply the difference in cumulative times between the selected driver and the other drivers being viewed.

The graph scale is reversed so that cars that have a lower cumulative time are higher on the axis.

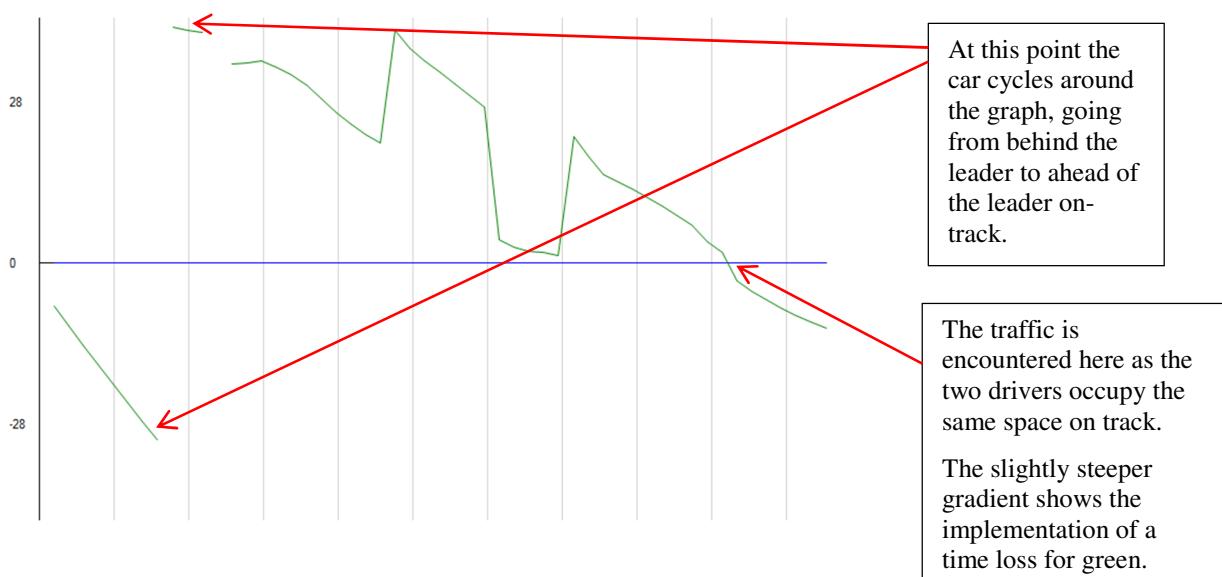
The graph’s axes can be modified using the axes panel – see the Axes Panel section for details.



### Lapped Cars

Lapped cars cycle from the bottom to the top of the graph. This keeps them within one single view on the graph. Cars that are particularly fast may also cycle off the top of the graph and back on to the bottom.

The graph can therefore be used to estimate when traffic may be encountered.



### Right-click events

Right-clicking on the graph will have different effects depending on the type of graph that is displayed. The click location will represent an event for the driver closest to the click, on the lap nearest to the click.

If the graph is representing strategies, the right-click event will add a pit stop at the location of the click. If a pit stop already exists for the driver at this location, the pit stop will be removed.

If the graph is showing a race simulation, the right-click event will force an overtake to occur between the clicked driver and the driver ahead.

This will occur regardless of the track position difference, and is simulated by a time loss for the car ahead.



### Showing Traces

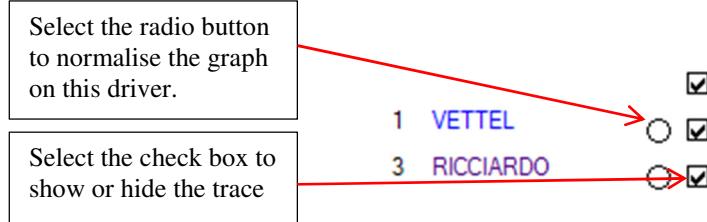
Graph traces for each individual driver can be shown and hidden using the check boxes in the Driver panel. The boxes will toggle to hide and show the traces.

The check box at the top of the column of boxes will show or hide all of the traces.

### Normalising the Graph

The graph is normalised on the driver that is selected by the radio buttons in the Driver panel. Changing the selected radio button will change the driver that the graph is normalised on.

Note that you cannot normalise the graph on a driver whose trace is hidden, and that when the normalised driver's trace is hidden normalisation passes to the next fastest driver.



### Colours

Each driver's trace is shown in a different colour. The driver's colour matches the colour that their name is displayed in in the Driver panel, which acts as a key.

## Driver Panel

### Driver List

The driver list on the left of the panel serves as a key for the graph.

The driver's name and number are shown along with his name displayed in his graph trace display colour.

When a race simulation is active, the list of drivers is sorted according to the positions at that time, so that the order of drivers in the race is clear.

Driver names are shown in the colour that they appear on the graph.

Drivers	
<input checked="" type="checkbox"/>	1 VETTEL
<input type="checkbox"/>	3 RICCIARDO
<input checked="" type="checkbox"/>	14 ALONSO
<input type="checkbox"/>	7 RAIKKONEN
<input type="checkbox"/>	22 BUTTON
<input type="checkbox"/>	20 MAGNUSEN
<input type="checkbox"/>	8 GROSJEAN
<input type="checkbox"/>	13 MALDONADO
<input type="checkbox"/>	6 ROSBERG
<input checked="" type="checkbox"/>	44 HAMILTON
<input type="checkbox"/>	99 SUTIL
<input type="checkbox"/>	21 GUTIERREZ
<input type="checkbox"/>	27 HULKENBERG
<input type="checkbox"/>	11 PEREZ
<input type="checkbox"/>	19 MASSA
<input type="checkbox"/>	77 BOTTAS
<input type="checkbox"/>	25 VERGNE
<input type="checkbox"/>	26 KVYAT
<input type="checkbox"/>	9 ERICSSON
<input type="checkbox"/>	10 KOBAYASHI
<input type="checkbox"/>	17 BIANCHI
<input type="checkbox"/>	4 CHILTON

### Radio Buttons

The radio buttons on the panel are associated with the driver next to them. Selecting a radio button will normalise the graph on that driver. If strategies are active, selecting this button will also display this driver's strategy.

Only one radio button can be selected at any one time.

### Check Boxes

The check boxes on the panel are used to select whether driver's traces are displayed on the graph. Multiple check boxes can be selected at any one time to show multiple traces against each other.

The check box at the top of the column is used for showing or hiding all of the traces as required.

### Timing Data

When a race simulation is active, the Driver Panel will show the position of the driver, the gap between him and the leading driver, the interval between him and the driver ahead, and the number of pit stops that have been completed up to that lap in the race.

The timing data mirrors the timing screens provided at races, and allows the race to be tracked in detail as it progresses.

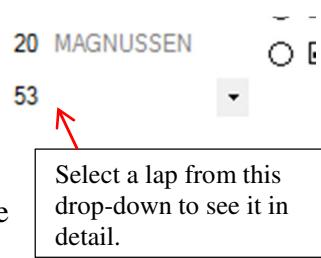
14 ALONSO	<input checked="" type="checkbox"/>	1	0	52	1
6 ROSBERG	<input checked="" type="checkbox"/>	2	8.24	8.24	1
44 HAMILTON	<input checked="" type="checkbox"/>	3	9.06	0.82	1
27 HULKENBERG	<input checked="" type="checkbox"/>	4	26.2	17.1	2

For the lead driver, the interval is given as the lap number, in line with standards.

The timing data displayed is position, gap, interval, pit stops, in that order.

### Lap Select Box

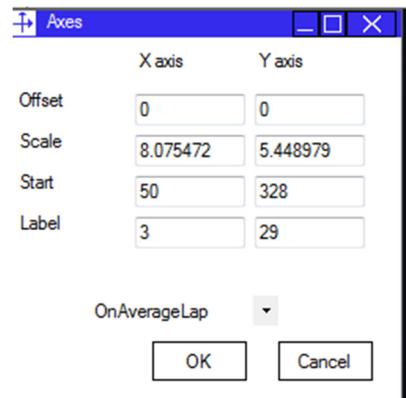
Below the list of drivers, a drop down allows the race lap to be selected. When this is changed, the graph will update to show this lap in detail, by adjusting the horizontal scale and offset. The driver's positions will also be set to the positions on this lap, and the timing data on the right of the panel will be updated to its predicted values for this lap of the race.



### Axes

The axes panel allows the axes displayed on the graph to be modified. The four options for modification are:

- Base offset – the value at which the axis starts.
- Scale factor – the number of pixels per unit on the axis.
- Start location – the location on the panel that the axis starts. This can be used to shift an axis up and down.
- Label spacing – specifies the number of units to separate the axis labels at.



The axes panel will update when the axes are modified outside of the user's control, for example after a form re-size. It also allows changes to be accepted using 'OK' or rejected, using 'Cancel'. Data input is validated to ensure that only reasonable values are used for the axes.

Axes values are set to defaults calculated from the available dimensions for the graph when the graph is started.

### Modifying Driver and Race Data

Driver and race data is loaded from a database. This means that it is easy to update the driver and race delta for new seasons, for example.

To modify this data, the system must first be closed. It will only update when re-opened again.

1. Locate the folder where the application file is installed.
2. Within this folder, select and open the StratSim Microsoft Access Database file.
3. The database contains tables named 'Drivers' and 'Tracks'. To edit data, open the relevant table.
4. Locate the record that needs to be updated, and modify the data in the cell.
5. Save the table and then close the database.
6. Reopen the system to see the changes.

Note that the tracks should be stored in the order that they are run in the season, and the drivers should be sorted by team, to ensure that the system functions properly and logically.

## Samples of actual screen displays

### During System Operation

See the detailed description of how to use the system for screenshots as the system is used. These demonstrate what purpose controls have, and how to use the system.

### File Formats

The system uses a number of file types to be stored on the hard drive. These files could be generated outside of the system and used within it, so the file formats are listed here for openness.

### **Pace Parameters**

Pace parameter data is saved as a CSV file. Data starts in row 2 column 2 and the first row and column provide keys to the data. It is read from r2c2 horizontally and then down.

The format is as follows:

Driver	Speed	Tyre Delta	Prime Degradation	Option Degradation	Pace	Fuel Effect	Fuel Consumption	P2 fuel load
String	Single	Single	Single	Single	Single	Single	Single	Single
Example	315	-1.5	0.1	0.3	94.2	0.03	1.8	30

### **Strategies**

Strategies are written to a CSV file. This file has four columns, and data is written to all three, starting in the second row and first column.

The data is written in a database-type format, where each unique stint, identified by a driver name and start lap, is listed on a separate row. For example:

Driver	Start Lap	Length	Tyre
VETTEL	0	16	Option
VETTEL	16	16	Option
VETTEL	32	16	Option
VETTEL	48	10	Prime
RICCIARDO	0	16	Option
RICCIARDO	16	16	Option
RICCIARDO	32	16	Option
RICCIARDO	48	10	Prime
ALONSO	0	12	Option
ALONSO	12	12	Option
ALONSO	24	17	Prime
ALONSO	41	17	Prime

### Database – Drivers

The list of drivers is stored in a database. Each driver has an index, name, unique number, line colour (for displaying on the graph), and a team. The team name is linked to the list of teams.

Driver data does not need to be sorted in any order. However, the system will display information in the order it is loaded in the database, so it is recommended that data is sorted by team and then by team position.

Index	DriverName	DriverNumt	LineColour	Team	Click to Add
0	VETTEL	1	Blue	Red Bull Racing	
1	WEBBER	2	Indigo	Red Bull Racing	
2	ALONSO	3	Red	Ferrari	
3	MASSA	4	DarkRed	Ferrari	
4	BUTTON	5	DarkGray	McLaren	

### Database – Tracks

The tracks are also stored in a database. Each track has a country, which is its unique identifier (use Europe for the European Grand Prix), a number of laps, a default fuel consumption and a default pit lane time loss.

Data is displayed in the order it is stored in the database, so it is recommended that it is stored in season order.

Index	Country	Laps	PitStopLoss	FuelConsumption	Click to Add
0	AUSTRALIA	58	21	2.5	
1	MALAYSIA	56	21	2.5	
2	CHINA	56	21	2.5	
3	BAHRAIN	57	21	2.5	
4	SPAIN	66	21	2.5	

### Text File – Settings

The settings text file contains a single setting on each line. This is saved and updated within the program, and there is no need to update the file outside of the system.

However, a guide to its contents is provided here. Only the first column of the table is saved in the file, the second and third columns describe the contents.

Example Data	Format	Description
-1	Single	Required pace delta for an overtake
-1	Single	Required speed delta for an overtake
0.4	Single	Minimum interval between two cars following
1	Single	Time loss when car is lapped
0.4	Single	Time loss when car is overtaken
-0.8	Single	Delta between tyre types

Example Data	Format	Description
0.3	Single	Default option tyre degradation
0.1	Single	Default prime tyre degradation
315	Single	Default top speed
0.02	Single	Default fuel consumption
200	Single	Default pace
60	Single	Default P2 fuel level
../	String	Base file path for timing data
0.6,0.4,0.2,0,-0.2	Single[5]	Track evolution – delta to qualifying for each lap
-0.3	Single	Track improvement per lap
AcroRd32	String	Process name of PDF reader application
G:\F1 2013\Race Reports\	String	Base file path for PDF timing data files.

## Troubleshooting

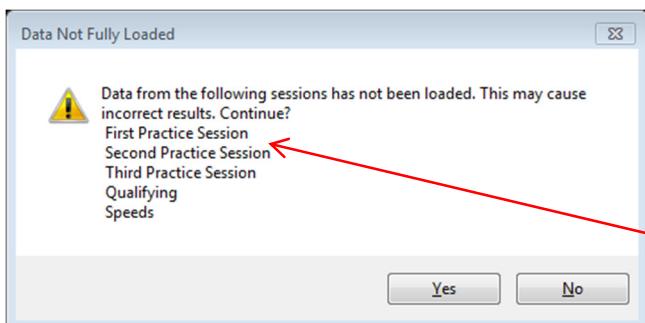
### Crashes

The system should never crash during normal operation. If it does, re-open the system and continue to use it as normal. Data may be lost when this occurs so I recommend that whenever race parameters or strategies are calculated they are saved.

If the problem persists, contact the developer, who may be able to provide assistance.

### Error Messages

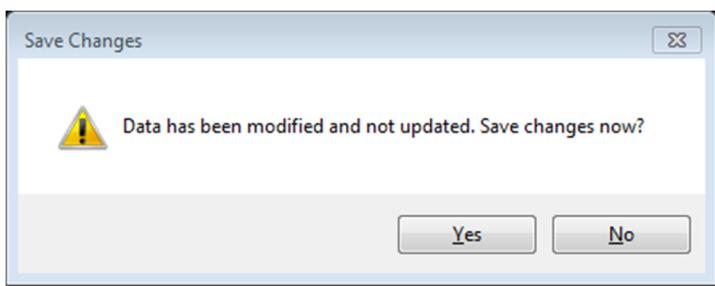
#### Loading Data



This error message appears when not all data has been loaded from PDF files. The sessions that have not been loaded are listed.

List of sessions that have not had data loaded yet.

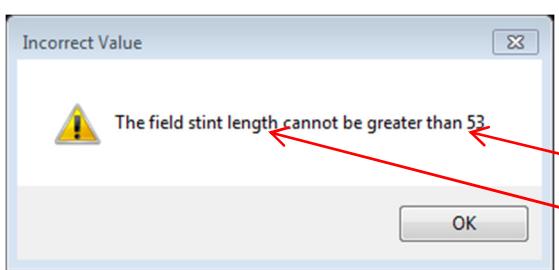
#### Update Data



Attempting to load strategies before parameter data has been updated will generate this message.

The changes must be updated to take effect in the calculation of strategies.

#### Validation

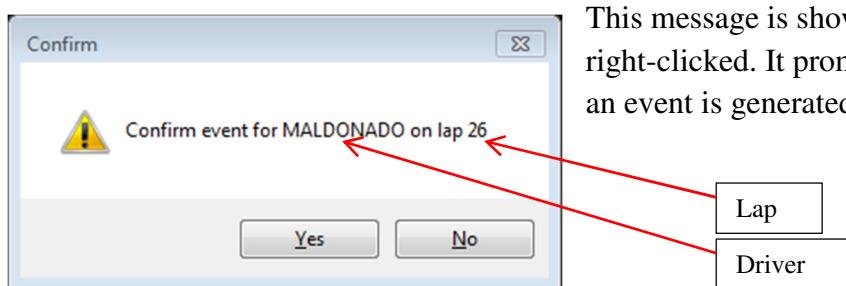


This message is displayed when an invalid value is entered. The message prompts the user to the field concerned and the limit on the valid values.

Valid limit

Field concerned

### Graph Events



This message is shown when the graph is right-clicked. It prompts the user when an event is generated on the graph.

## Appraisal

### Comparison of project performance against objectives

#### Specification Acceptance

During the testing phase, it was determined that the specification was met to a large extent. All tests that were carried out were passed. However, there were some tests that were not completed because it was known that the required functions were not implemented.

These were:

Requirement	Description	Reason
2.6.3e	The system should simulate the effects of a ‘tyre cliff’, where tyre performance degrades exponentially past a certain number of laps.	This decay model would have greatly complicated the algorithm used for optimising stint lengths. However, it will be listed as a possible extension as it is a real parameter in the simulation that needs to be simulated for accuracy purposes.
2.8.2	The system will be able to adapt to new regulation changes in future years.	This would involve the option for the user to vary fuel consumption through a race. This has not been implemented due to time constraints. Similarly, used tyres will have a different degradation pattern to new ones, which needs to be implemented as the number of available sets of tyres is reduced by the FIA. These features will be listed as extensions.

#### Real World Testing

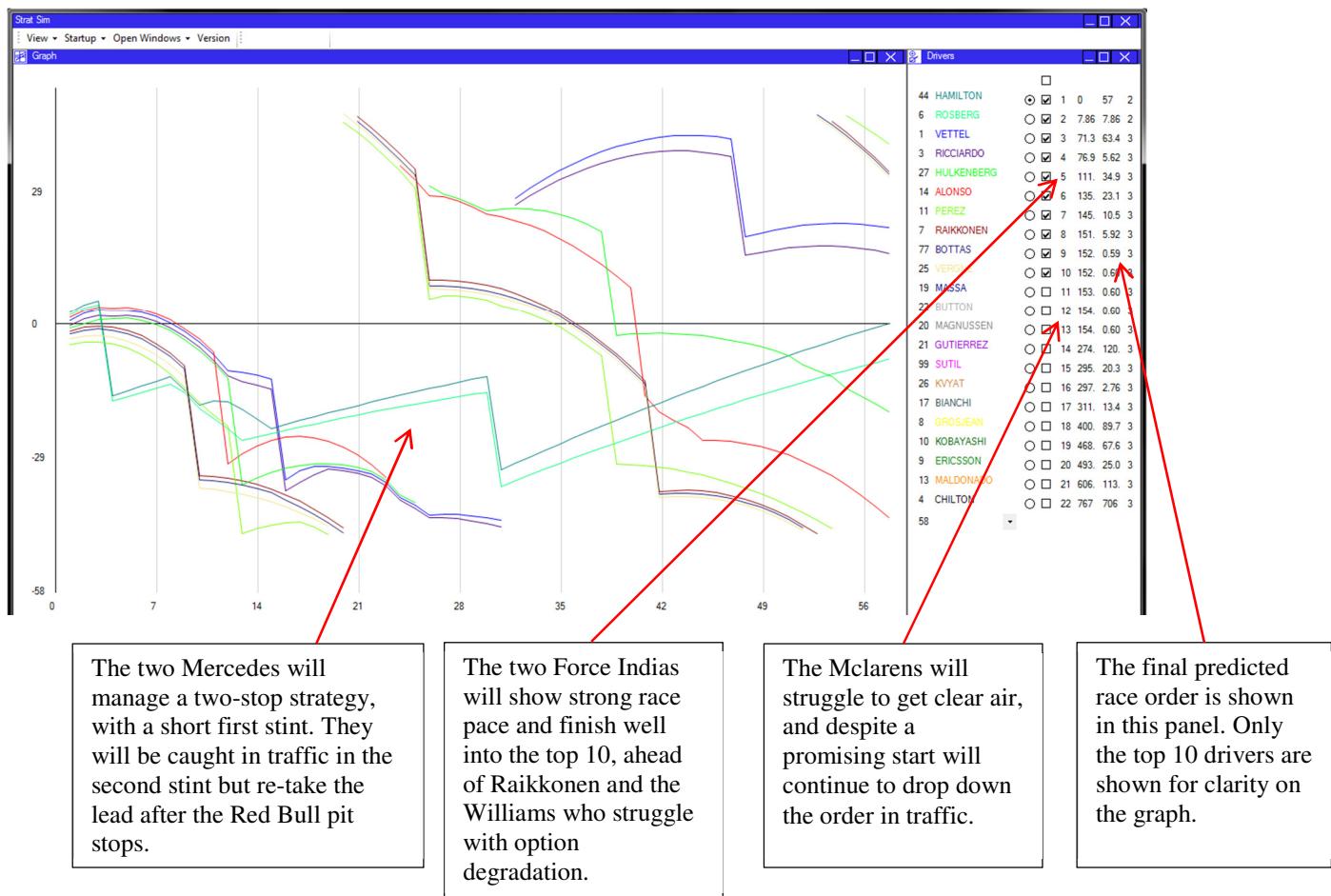
The system was finished in time for the Australian Grand Prix on 14<sup>th</sup> – 16<sup>th</sup> March. I loaded data from the practice sessions and qualifying on the Saturday night. I modified the settings as required by the race, but none of the system’s calculated parameters or optimised strategies were changed.

The grid order, determined by real qualifying, was as follows:

Grid	Number	Name	Team
1	44	Lewis Hamilton	Mercedes
2	3	Daniel Ricciardo	Red Bull Racing-Renault
3	6	Nico Rosberg	Mercedes
4	20	Kevin Magnussen	McLaren-Mercedes
5	14	Fernando Alonso	Ferrari
6	25	Jean-Eric Vergne	STR-Renault

7	27	Nico Hulkenberg	Force India-Mercedes
8	26	Daniil Kvyat	STR-Renault
9	19	Felipe Massa	Williams-Mercedes
10	77	Valtteri Bottas	Williams-Mercedes
11	22	Jenson Button	McLaren-Mercedes
12	7	Kimi Räikkönen	Ferrari
13	1	Sebastian Vettel	Red Bull Racing-Renault
14	99	Adrian Sutil	Sauber-Ferrari
15	10	Kamui Kobayashi	Caterham-Renault
16	11	Sergio Perez	Force India-Mercedes
17	4	Max Chilton	Marussia-Ferrari
18	17	Jules Bianchi	Marussia-Ferrari
19	21	Esteban Gutierrez	Sauber-Ferrari
20	9	Marcus Ericsson	Caterham-Renault
21	8	Romain Grosjean	Lotus-Renault
22	13	Pastor Maldonado	Lotus-Renault

From here, using the calculated pace parameters and optimised strategy data, the system runs a race simulation. This was annotated pre-race to point out some of the important predictions:



Grid	Name	Team	Predicted Race	Actual Race	Delta
1	Lewis Hamilton	Mercedes	1	DNF	
2	Daniel Ricciardo	Red Bull Racing-Renault	4	2*	-2
3	Nico Rosberg	Mercedes	2	1	+1
4	Kevin Magnussen	McLaren-Mercedes	13	3	-10
5	Fernando Alonso	Ferrari	6	5	-1
6	Jean-Eric Vergne	STR-Renault	10	9	-1
7	Nico Hulkenberg	Force India-Mercedes	5	7	+2
8	Daniil Kvyat	STR-Renault	16	10	-6
9	Felipe Massa	Williams-Mercedes	11	DNF	
10	Valtteri Bottas	Williams-Mercedes	9	6	-3
11	Jenson Button	McLaren-Mercedes	12	4	-8
12	Kimi Räikkönen	Ferrari	8	8	0
13	Sebastian Vettel	Red Bull Racing-Renault	3	DNF	
14	Adrian Sutil	Sauber-Ferrari	14	12	-2
15	Kamui Kobayashi	Caterham-Renault	19	DNF	
16	Sergio Perez	Force India-Mercedes	7	11	+4
17	Max Chilton	Marussia-Ferrari	17	14	-3
18	Jules Bianchi	Marussia-Ferrari	22	NC	
19	Esteban Gutierrez	Sauber-Ferrari	14	13	-1
20	Marcus Ericsson	Caterham-Renault	15	DNF	
21	Romain Grosjean	Lotus-Renault	18	DNF	
22	Pastor Maldonado	Lotus-Renault	21	DNF	

\*Ricciardo was disqualified for a fuel irregularity that had no bearing on his actual finishing position.

The race included a safety car, but because this was taken at the same time as the first round of pit stops, it made very little difference to the actual race outcome. There was also an additional parade lap at the start of the race, reducing the effective race length to 57 laps after Chilton stalled on the grid. Finally, it is worth noting that Bottas, Perez, and Sutil were all involved in incidents that caused them to lose time, but still finish the race. This accounts somewhat for the inaccuracies in Perez' position, but accentuates the error in Bottas and Sutil's position.

From this table it is possible to assess how accurate the system was at predicting the race results. The following conclusions can be drawn:

- The system was fairly accurate for positions 1-10, with the exception of 3 and 4.
- The system was less accurate for positions 10-14, although it was quite close for some positions.
- The system did not predict DNFs, but this is not part of its specification.

The graph of the race progression, and in-depth study of the actual race results, allow the errors in the system to be accurately traced. We can then look in more detail at where the simulation was correct and where it was not.

- The system predicted a Mercedes 1-2 with Hamilton ahead of Rosberg, which was the likely outcome had Hamilton not had reliability issues. It is therefore good at identifying the fastest cars.
- The system predicted Vettel catching up and finishing ahead of Ricciardo. We will never know where Vettel could have got to, but the system was accurate for Ricciardo. This is likely to be an extension of the above point that the system is effective for the fastest cars.
- The system correctly predicted the finishing positions of the two Saubers at the back of the field (once the retirements ahead have been taken into account). The system is therefore quite good at identifying and simulating the slower cars.
- The system suggested that Button and Magnussen would drop through the field after being caught in traffic. This was not the case so the overtaking model needs some fine-tuning.
- The system accurately predicted the traffic around positions 6-10, with Hulkenberg, Raikkonen, Bottas, and Vergne all close together in the first stint. However, the order of these cars was not correct and therefore the overtaking model could be improved.
- The system incorrectly suggested that a 3-stop strategy was ideal. This was mostly due to slightly inflated option tyre degradation figures. It is possible that the settings were not set optimally for the simulation.

Using these points, I believe that the area where greatest improvements to the system can be made is the overtaking model. This definitely needs to be looked at along with more in-depth study of some races to observe the exact parameters that allow an overtake to occur.

However, I can draw confidence from the fact that the system is able to accurately predict things like traffic, where multiple cars are close together. This is important because it is the major factor in pit stop optimisation for each team – they will want to avoid traffic if at all possible.

## Possible extensions

There are several possible extensions to the project that would be completed if there was more time to finish it. A project can always be improved and at this stage this is the list of improvements that I personally feel would make the biggest difference to the project.

I have divided the list up into additional features, modification of existing features, and code refactoring.

### Features

There are some features that I would like to add to the system to allow it to meet the specification and to extend the number of functions available to the user when simulating the race. These are:

- Tyre Cliff Simulation – F1 tyres are designed to degrade slowly and then reach a ‘cliff’, where the performance suddenly drops off. This feature in a tyre model would improve the simulation of strategies and provide more information to the user.
- New/Used Tyre Model – Tyres degrade at different rates depending on whether they have been used or not. I want to allow the user to select if a tyre has been used or not. It is possible to warm a tyre up in practice sessions before using it in the race, and this will affect its degradation characteristics, which could improve race performance.
- Variable Fuel Use Model – It is common in an F1 race to start with not enough fuel to make the end of the race. It is then possible to increase and decrease fuel consumption through the race to provide power and economy when they are required. This would extend the ability for the user to simulate the race and improve the performance of the cars. I would have to create a map between fuel consumption and pace, linking the two in a realistic way, so that the user can optimise their fuel consumption through the race.
- Race Strategy Optimisation (after race simulation) – The specification asked for the system to provide functionality for optimising strategies after a race simulation had taken place. This would mean the system could vary the timing of pit stops to avoid encountering traffic during a race, optimising the finishing position. This would improve the quality of the predicted strategy.

### Accuracy

I would like to make improvements to some sections with the aim of increasing the accuracy of the simulation. This will assist the optimisation of strategies and contribute to better race results. These improvements are:

- Overtaking Model – As mentioned previously greater study of past races is required to work out if a car will be able to overtake another car. Getting a reliable idea of top speed and pace will assist with this, but working out the algorithm to reliably simulate reality will improve the quality of the race simulation.
- Track Evolution through sessions – As the race weekend progresses, the track ‘rubbers in’ and gets more grippy. This means that the lap times set in these

conditions are artificially higher than the actual pace of the car. Because the system uses comparisons between speeds in different sessions, it is important that this is modelled accurately. I would like to make sure that the offsets are applied accurately and appropriately throughout the project.

### Code Refactoring

Finally, looking back on the project and the state it is in now, I believe it would be beneficial to split up some of the code and write it in more efficient ways. This will allow it to contain further developments without the code becoming unclear or redundant. The following changes are proposed:

- Separation of Flow Layout into a different project; inheritance of these classes within the StratSim project. This will allow the flow layout logic to be used in other projects and allow the system to expand in the future.
- Moving calculation methods (e.g. OptimiseStrategies) into a static CalculationController class. This will neaten up the code and allow separation of the functions class from the main program. The functions class will become a self-contained module that could be used in other projects.
- Strategies, graph traces, and collections of pace parameters inherit from the driver class rather than containing an instance of a driver. At the moment, a strategy must contain an instance of a driver, which is not economical on RAM and does cause update errors because the local instance is updated and not the reference. By ensuring that a strategy derives from a driver, the strategy and driver will be permanently linked.
- More thorough implementation of interfaces (Framework and user-defined) so that future code changes are guaranteed to maintain functionality. I would like to interface the driver and strategy classes to make sure that if they are updated in future all of the methods required are present. Implementing common interfaces such as IComparable and ICloneable will mean that it is easier for future changes to use framework functions, such as Array and List sorts.

## User feedback

### Appraisal

I contacted my user with the completed system, and after reviewing the system and the results of the simulation for the Australian Grand Prix, he has given the following appraisal:

---

*"The F1 simulation project has been completed as per the user requirements. Alex has continued to keep me informed of progress throughout the project.*

*"Validation of the system's output against actual results from the first race of the F1 season shows a strong correlation between the simulation and real world while highlighting some areas (such as the overtaking model) for further refinement. This is very common with this form of simulation, with tweaks to the simulation required between races on the RBR [Red Bull Racing] implementation. It must be noted that the adoption of a probabilistic overtaking model has proved difficult at RBR as the variance in predicted outcomes between subsequent simulation runs during live race prediction is too great to be useful so a deterministic approach was taken.*

*"The overall accuracy with which the system predicted the outcome of the first race in Australia was very good. The logical extension of this project would be to start providing live predictions during the race, but I understand that access to the live timing data may be difficult outside of the industry.*

*Overall, as an initial implementation of a solution to a complex problem, with source data limited to that available in the public domain this is an impressive result."*

---

*Richard Dutton  
Software Technical Team Lead  
Red Bull Racing*

The client is very pleased with the final outcome and is satisfied that it meets the requirements defined earlier. There are some very definite areas to improve and some useful suggestions for trial solutions.

I am pleased that the limitations of the system have been recognised and satisfied that given time and more resources the changes could be implemented.

### Actions

In order to meet the requirements specified by the user in his review, there are two major areas that can be improved.

Firstly, the overtaking model could be replaced with a deterministic model. This would centre around two values – the pace delta of the cars, and the top speed delta of

the two cars. If both were above a threshold value, an overtake would be certain to occur. I can do this very easily, but whether it accurately models reality is less certain.

Should the system be expanded later on, the probabilistic model is also ideal for converging towards a monte-carlo-type simulation. With increased computing power and better data, this would be able to produce an optimised strategy taking into account the race and interactions with other cars. I will opt to further fine tune the current model to try and improve the realism of the simulation.

Secondly, the ability to run the system live on race days is an extension that is mentioned. I would like to be able to do this and it would mean that the race strategists have a very valuable tool at their disposal for interpreting how the race may finish, given how it has started.

Some pilot research into this area suggests that it is possible to collect lap times from FIA timing streams, and that applications do exist to do this. However, their format is not designed for this type of application so it is very difficult to get the right data from the feed.

I feel that the provision for live data input would be a very valuable addition and is something that with more time, and perhaps more resources, I would be able to incorporate fairly easily. The race simulation is designed so that it could, in theory, be started with a set of known data on a lap other than lap 1; the changes to current code would be limited.

However, the amount and complexity of new code that would have to be written means it is not a viable addition to this version of the system.

## Project Evaluation

*During the development of this project, I have gained a greater understanding of the requirements of programming effectively and logically. The project has therefore improved as it has progressed, with older code being harder to maintain and improve than newer code.*

*There are some sections which are not entirely clear, and possibly are inefficient at their purpose.*

*However, the system meets nearly all of the specification points, and I will go on to develop it further for the client so that it meets all of the points and could be used as a strategy simulator.*

*The development of the project over time has inflated to a greater extent than I expected, and it is now a very large project. The project and code structure was not designed to cope efficiently with this amount of code, and therefore the system would benefit from a good refactor to separate sections out and to make sure that each module is stand-alone.*

*I feel that the project could have done with more thorough testing, and the implementation of unit tests should have been done much sooner. This would have aided development and made sure that only the features required for the system to function were programmed.*

*To deal with the above points, a much more in-depth plan of the structure of the program, and specifically the design of the output, would have been very beneficial. I feel that currently the data models are not entirely optimised for the design of the output, but in most cases it is still not worthwhile including a.viewmodel.*

*This plan would also include testing, and the VS framework would then allow the program to be built up piece by piece, using bottom-up and integration testing, contributing to a very efficient and neat piece of code.*

*I now have plans for expanding the system and combining some other projects with it; it has the capability for this functionality to be added. I will also refactor some of the major aspects at this stage because this will improve the efficiency of the code.*

*The project has been completed on-time and with user satisfaction, which is the ultimate goal of any programming project, and as such I am pleased with the general outcome.*

---

Alex McCormick

## Appendices

### Appendix 1

#### Calculus used for strategy optimisation

To calculate the optimum strategy requires two steps. Firstly, strategies must be generated using every possible combination of numbers of stints on the prime and option tyres. These strategies must be optimised.

Secondly, the time taken for these strategies needs to be calculated, and the one with the least time needs to be selected for use by the given driver.

The first step is detailed here; the second step is a very simple most wanted holder loop. This first step requires a little calculus to optimise the stint lengths, and these calculations are detailed here.

#### Abstraction

In order to simplify the problem, I can eliminate some variables in the calculation:

- Driver Pace is a constant addition to every lap time and does not vary depending on the strategy. It will not affect the calculation so it is ignored.
- The effect of fuel will be the same regardless of the strategy selected, so it will not affect the calculation.
- It is not possible to assess the effects of other cars, so only one car is assessed in the strategy.
- Every option stint and prime stint will be the same length as any other option stint and prime stint.
- 

#### Defining the Problem

We can write the total time taken for a strategy in terms of his pace parameters, the number of laps in the race, and the number of stints he is to do on each tyre.

These variables are summarised in the table below:

Identifier	Variable	Description
$s_p, s_o$	Laps in stint	The number of laps in an option (o) or prime (p) stint.
$l$	Laps in race	The number of laps in the race being simulated.
$d_p, d_o$	Degradation	The tyre degradation per lap on prime and option tyres.
$n_p, n_o$	Number of stints	The number of stints on the prime or option tyre.
$\Delta$	Tyre Delta	The difference between speeds on the two tyre types.

#### Strategy Time

The time taken for a strategy to be completed is the sum of the time lost due to tyre degradation on each tyre, and the time gained through the tyre performance delta:

Tyre degradation is modelled by a constant lap time increase per lap. This means that the increase in time is linear and starts at zero, and therefore the total time loss in an s-lap stint where the tyre degradation is  $d$  is equal to  $t = \int_0^s (d \times s) ds$ .

On integration, this yields  $t = \frac{ds^2}{2}$ . The time loss in a number of stints on this tyre,  $n$ , is equal to  $t = \frac{n ds^2}{2}$ .

The time gained due to using the option tyre delta for a number of laps is simply  $ns\Delta$  where  $n$  and  $s$  relate to the option tyre.

We can write the sum of all of these terms:

$$T = \frac{n_p d_p s_p^2}{2} + \frac{n_o d_o s_o^2}{2} + n_o s_o \Delta$$

Now, we can use the relationship between number of option and prime stints, and the number of laps in the race, to replace all references to  $s_o$  with an expression in terms of  $s_p$ . This yields an expression that we can differentiate later.

$$n_p s_p + n_o s_o = l$$

$$\therefore s_o = \frac{l - n_p s_p}{n_o}$$

Substituting:

$$T = \frac{n_p d_p s_p^2}{2} + \frac{n_o d_o \left( \frac{l - n_p s_p}{n_o} \right)^2}{2} + n_o \left( \frac{l - n_p s_p}{n_o} \right) \Delta$$

We can now find out how time varies according to the length of a prime stint (all other ‘variables’ are constant for the calculation). Differentiation of this expression with respect to  $T$ , in other words finding the rate of change of  $T$  with respect to the prime stint length, yields:

$$\frac{dT}{ds_p} = n_p d_p s_p - \frac{d_o l n_p}{n_o} + \frac{d_o n_p^2 s_p}{n_o} - n_p \Delta$$

It is logical to assume that the point at which time does not change with stint length is the point at which the stint length is optimal – it is the minimum time, and any change in stint length around it will cause a greater time to be found. We therefore equate with zero and solve for  $s_p$ , which gives:

$$s_p = \frac{\frac{d_o l}{n_o} + \Delta}{d_p + \frac{d_o n_p}{n_o}}$$

Using the relationship between laps, prime stints, and option stints, we can calculate the optimum option stint length too. This optimum stint length is then rounded to an integer and is then saved as the optimum stint length for this particular strategy.

The rounding is done by a separate method that ensures the value is rounded up or down appropriately. For example, even if a value is below the 0.5 limit, it could be rounded up to save extending the stints on the other tyre.

The methods that implement this can be found in the `Strategy` class:

SetStintLengths (controls the process).....	156
OptimisePrime (implements the above algorithm).....	158
RoundToIntegerBasedOnRatio.....	159

The strategy times are then evaluated, ready for comparison in stage 2 of the optimisation process.

## Notice

The material herein is copyright © 2014 Alex McCormick. Permission for reproduction for purposes other than those relating to education or necessary for the processing of this document as a submission for an academic qualification must be sought from its author.

This document is standalone and does not require or rely upon any other documents or projects. Where referenced these other projects are only cited for guidance.

Some items of the contents are classified, and distribution or attempt to distribute the intellectual property or trade secrets within to any other organisation, especially Formula One teams, with the exception of Red Bull Racing, will be taken seriously, unless it is for the sole purpose of processing and marking this document.

The code for the project itself is free for reproduction and improvement. It is provided as-is, with no guarantee of maintenance, continued function, or of suitability to the required task.