1. The Project Introduction

**1.1 Background**

The games industry is expanding and people are demanding and expecting more from games. There is a high demand in realism, especially in simulations like racing and flight games. Simulation games were always found interesting on how they can simulate real world environments in games, but there are always unrealistic features. The understanding on how some simulations can be rated better than others is immeasurable in terms of realism, but most were concluded how one game was more realistic than the other. In the real world a lot of simulations are driven from complex maths and physics, so most simulation game types should really be the same, as they should use the same math and physic concepts.

Driving simulations always had a problem with realism, most game companies focused on either the games realism or the racing aspects. The consumers are always demanding more from racing games, and realism is always a high demand, but it does depend on the market groups, so people under 18 wouldn’t know how to drive, so how would they know what realistic driving is, so would probably demand more on the racing and fun aspects, whereas over 18’s do, so they would probably demand more on the realism aspects.

The real world in games has always considered physic concepts and tries to implement them the best they can. With racing games they use vehicle dynamics as its base physics engine, to determine the speed of the car, and what can affect the car at these speeds. Racing games should use the same laws of physics and vehicle dynamics, so therefore the implementation of vehicle dynamics in racing simulations should really be similar between different games, so what actually differentiates racing simulations?

From this problem, the project will be implementing a physics engine that simulates a car on a track where the user can control the car with a gaming steering wheel, but with physics realism. The study will be researching on vehicle dynamics to look at what can affect a car at certain speeds, turns, braking, road surfaces etc. The end goal is to be able to show a 3D application that can simulate the physics of racing and realism.

**1.2 Objectives**

The project is going to need a lot of research, design and analysis, before implementation, so the objectives are used to guide the project through each step and to help find the relevant information that is needed to implement this application.

* Understand the key concepts of vehicle dynamics and the physics of racing
* Discover what the market is demanding in racing games, is it realism or enjoyment
* Find out what actually differentiates the current racing games user experience
* Explore the concepts of development using XNA or DirectX, and distinguish the best option
* Design relevant models of the physics engine and 3D application using UML Diagrams
* Integrate the use of a gaming steering wheel, using the support of the SDK
* Using the 3D tools and SDK’s to implement models of a car and a track for the application
* Create a simulation of a racing game that demonstrate the use of vehicle dynamics
* Register and potentially exhibit at the graduate exhibition (GradEx) in 2010

**1.3 Resources**

* PC – Must be capable to support the relevant software, for me to implement my project, it needs to support high detailed graphics for the demo application and processing speed for the physics engine.
* Visual Studio 2008 – Is used to build the core application, it needs to support C++ or C# and any relevant development kits.
* Borland Together – Used to produce UML diagrams, to demonstrate the design of the application.
* XNA SDK – Allows easy game development, so easy to integrate the 3D models into the application. It uses C# as its core language to develop applications.
* DirectX SDK – A more complex development kit, which uses C++ as its core language, but is adaptable with C# and other languages. This is more relevant to industrial development in games.
* Gaming Steering Wheel – that can showcase the application realism in racing simulations.

**1.4 Deliverables**

A project plan will be implemented to help organize the project throughout the year. Relevant research reports will be produced to show what is currently in the industry of racing games, and what people really want, also the physics of vehicle dynamics. Report of software methodologies and design patterns, to review which pattern and methodologies best suits this project. Designs of the code architecture using UML diagrams and OO concepts will be implemented to show how the implementation of the application will be developed; also graphical drawings will be designed to show the UI and UX implementation.

The report will consist of an analysis from the research, which will describe what has been found and what is suitable to this project. It will also consist of a wide range of designs of code architecture and graphical designs, which will be described to show the advantages and disadvantages of each design. A number of tests will be applied to the end product to ensure the application works and show cases the application efficiently, the report will show a test plan to show the results and evaluations of tests. There will also be an overall evaluation and conclusion for this project, to discuss the problems that were encountered and how improvements can be made to enhance the project plan and the implementation of the application.

The final outcome of this project is a 3D application that can demonstrate the physics of racing using vehicle dynamics to show realism in racing simulations. The project is to determine how current racing games can be rated better than others, even if the laws of physics and vehicle dynamics should be the same. To demonstrate this theory, the project will develop a physics engine and use real world physics to demonstrate the concepts of vehicle dynamics, by implementing this application it will be able to compare real world physics with the application to current racing games. The application will showcase a 3D model of a car and a track, where the user can drive round the track using the gaming steering wheel. The user will discover the realism in the application, like it was a real car, and observations will be made to determine what the user prefers in racing simulations, either realistic or unrealistic racing.

**1.5 Ethics**

The project research will use references to current games and game companies, which it shall not breach the copyright act of uses of logos and images from their websites, this goes for models of cars and tracks which is used in the application as a demonstration. Any data gathered from users will be kept confidential under the data protection act. Research on comments and discussions in forums will be referenced and quoted within the project. Other research that is needed to copy phrases of text as examples from text books, magazines or forums, will be referenced and quoted under the act of plagiarism.

2. Research

2.1 Current Racing Simulations

**2.1.1 Introduction**

The market current has a wide range of racing simulation games, titles such as Need for Speed and Gran Turismo, but the games are different in many ways, due to how the game engine and physics are implemented. Vehicle dynamics and physics is a key aspect to racing simulations and is what makes the game realistic, but in real world physics there is a set of laws and racing simulators should obey and implement the same laws of physics and use the same mathematical calculations in vehicle dynamics. Obviously this isn’t the case because if the BMW M3 is selected in Gran Turismo and the same car is selected in Need for Speed, there is an immense difference in the acceleration, handling, braking, gear changing etc. The games are implemented differently dependent on the markets demand, as people may demand more on realism and some on entertainment, so it all depends on the game publishers target market.

This research is to discover the main reasons of what differentiates racing simulations in terms of vehicle dynamics and physics engines to illustrate a moving car around a track. The data gathered needs to be able to demonstrate what people in the market are demanding in racing simulators, how they are realistic or unrealistic, and what makes a good racing game, in terms of physics engines and vehicle dynamics.

**2.1.2 What is the market demanding in racing simulators?**

Finding what the markets is demanding in racing simulators was difficult, as there is a wide range of aspects in a racing simulator, for example the amount of cars, real world race tracks, car collision physics, car damage, and obviously the vehicle dynamics physics engine. Therefore narrowing down the criteria and focusing more on what the project is trying to achieve will help solve this problem, as a result the research will be focusing on realism in racing and vehicle dynamics.

There were a few websites found online which had relevant information to this research, the majority of them showed information about what people wanted, and they were generally forums, where people around the world discussed what is their favourite racing game, what is the most realistic racing game and what people actually preferred in terms of realistic and unrealistic racing simulators.

On this website, The Escapist [Escapist2009], one of the forums discusses about what people prefer, realistic or unrealistic racing games, and people were to vote and explain the reasons why they would choose one over the other. The first post was by the creator of the topic and he explains the two terms, people can vote on, as stated below.

*“‘Realistic’ racing games is straightforward enough, I'm talking Gran Turismo, Forza, any racing game that uses real life cars or at least realistic cars.”, Mewick\_Alex, [Escapist2009].*

*“‘Unrealistic’ racing refers to games like Mario Kart or F-Zero, where weaponry can be used or the gameplay is based around some other comical or unrealistic dynamic.” Mewick\_Alex , [Escapist2009].*

The quotes explain what people are voting on, but it doesn’t specify the aspects or criteria of the game in term of driving realism, it states more on game play realism, but it does state “*unrealistic dynamic”,* which does reference racing realism.

The results of the vote are shown on the site, and it indicates that 64.5% of people prefer Unrealistic racing simulators and 35.5% prefer Realistic racing simulators.

**Figure 1 Percentage of what people are demanding in terms of realistic and unrealistic racing games**

Figure 1 shows what the current market and the people who play racing simulators are looking for in a racing game. The majority of people are looking for more of an unrealistic aspect of racing in a game, but for what reasons, the next question asked and research should determine the reasons of people’s choices.

**2.1.3 What makes a racing simulator realistic or unrealistic?**

There are many different racing simulation games out in the current market and they all go in different categories either realistic or unrealistic. Here is a list of some of the current racing simulator titles in the market.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Title | Developers | Publishers | Realistic | Unrealistic |
| Gran Turismo | Polyphony Digital | Sony Computer Entertainment | O | X |
| Need For Speed | Electronic Arts | Electronic Arts | X | O |
| Forza | Turn 10 Studios | Microsoft Game Studios | O | X |
| Motorstorm | Evolution Studios | Sony Computer Entertainment | X | O |
| Burnout | Electronic Arts | Electronic Arts | X | O |
| Juiced | Juice Games Paradigm Entertainment | THQ | X | O |
| Wipeout | SCE Studio Liverpool | SCEE | X | O |
| Mario Kart | Nintendo | Nintendo | X | O |
| Dirt | Codemasters | Codemasters | O | X |
| Grid | Codemasters | Codemasters | O | X |
| Ridge Racer | Namco | Namco Bandai | X | O |
| Midnight Club | Rockstar | Rockstar Games | X | O |
| Blur | Bizarre Creations | Activision | X | O |
| Split Second | Black Rock Studio | Disney Interactive Studios | X | O |
| Fuel | Asobo Studio | Codemasters | X | O |
| Project Gothem | Bizarre Creations | Microsoft Game Studios | O | X |
| Test Drive | Eden Games | Atari | O | X |

**Table 1 how current racing simulators are rated by the public in terms of realism**

The category selected for each game was determined by what people stated and commented on forums. By reading and observing some of the comments on the forum (The Escapist) stated earlier, can help determine the reasons why people prefer realistic or unrealistic racing simulators.

Jandau, [Escapist2009] quoted *“Realistic enough to be believable, but not so realistic that it stops being fun”*, this indicates that a racing game should have realism aspects, so that it seems as real as driving a real car, but using pure physics and vehicle dynamics can actually be boring and not as enjoyable, therefore as stated by Akai Shizuku, [Escapist2009], *“Realism is boring”.*

“*unrealistic by a mile. realism is good in small doses in games. why else other than to escape reality are you playing games?”,* bloodmage2, [Escapist2009].

The quote stated by bloodmage2, [Escapist2009] signifies that people who play games don’t want to be stuck in a world of reality, and want to make unbelievable aspects in the world be put into games. If everything was real in games, it defeats the point in a game, as stated by another participant *“If I wanted realism, I could go outside and drive a car.”* popdafoo*,* [Escapist2009], why else would people play games if they can’t escape the world of reality, but then again, is everyone capable of racing a car around a track, probably not, so people turn to games to make it as realistic as possible and get the thrill of racing within the game, as they are incapable of performing it in real life.

*“you just don't get the euphony of a powerful racing engine”*, RAKtheUndead, [Escapist2009].

*“details of handling which make actual automotive racing a supreme challenge”,* RAKtheUndead, [Escapist2009].

*“The feeling of momentum as you try to brake heavily before a hairpin or any of the great details that go into a realistic racing experience.”* RAKtheUndead, [Escapist2009].

These quotes state how realistic racing games are better, and one of the main reasons is because of the thrill and the excitement of racing a car around a track. Also using realistic physics and vehicle dynamics in a racing game is more challenging and is harder to control the car at high speeds, but it gives a real life thrill in racing. Small changes to the car can either put the player in first or second place, and it’s because of these aspects people prefer realistic simulators as stated by TopGearFTW, [Escapist2009] , *“I love the fact that, for instance, changing the weight distribution by the smallest amount can net you a first place”.*

On this other website, Trap17 [Trap2006], it is questioning what is the most popular realistic racing simulator, and currently it shows there is a lot of votes for Gran Turismo and some mentioned a PC simulator known as Live For Speed. One of the comments that stood out indicates what a realistic racing game has.

*“A simulator has full physics, simulation all aspects that make a difference to driving”,* John Doe [Trap2006]

This quote obviously shows the criteria’s and aspects in a realistic racing game, and he mentions a lot on physics and vehicle dynamics, and what can effect a car racing on a track. This indicates and shows what people are looking for, in a realistic racing simulator, which is pure physics and vehicle dynamics in racing, which will give players the thrill of racing and realism.

Here is an article found online, WorldDude [World2009], where the author discusses about realistic racing simulators compared to arcade style racing games.

*“who just want to race and have fun, and not have to worry about the technicalities that go into your racing simulators.”,* WorldDude, [World2009]

This is a quote stated in the article, which indicates why people would choose arcade style racing games over realistic racing, because arcade styled games are more fun, even though there unrealistic but players don’t have to think about the speed there at for a turn and other technicalities in racing.

*“Gran Turismo is something that at this point in time I find completely boring and pointless.”,* WorldDude, [World2009]

Arcade style games are just a lot simpler and enjoyable, compared to realistic simulators as they can be frustrating at times, due to reasons why players would randomly slide of the track, which is probably caused by the timing in braking or the turning angle, but realistic racing is more challenging and gives the player more of a exciting racing experience, even though it can be boring at times, but as the article quotes:

*“I guess it all depends on what you consider ‘fun.’”,* WorldDude, [World2009]

People may find realistic racing games more fun than unrealistic racing games due to the immense thrill and excitement a realistic game can bring as it simulates reality in games, but unrealistic games can also provide the same thrill, but it’s unbelievable, so it may be more enjoyable to other people, overall people can turn away from reality, but with realistic racing simulators, people can play a game which can give them a true sense of experience as they would obtain from undertaking it in a real world scenario, but without being hurt or be put in danger.

**2.1.4 What makes a good racing simulator, in terms of physics engines and vehicle dynamics?**

Good racing simulators need good physics engines and realism incorporated into the game, finding information about what people want is good to see where publishers need to target, but what makes a really good realistic racing simulator, that can thrill and excite a player and create a realistic sense of experience in the game. The best source of information would be from someone who knows how real racing works on the track, understands how the laws of physics and vehicle dynamics can affect the car while racing, and has experienced the stimulation and exhilaration of real world racing. Unfortunately finding and organizing an interview with a real racer is complicated, but due to the release of Gran Turismo 5 Prologue, an Ex-Racer reviewed the game in terms of realism and posted an article online, [Alexander2008].

The article is written by Brian Alexander an ex-racer, he compares real life racing to Gran Turismo, and explains how they can differ and how Gran Turismo has made the game as realistic as it is in real life. One of the first statements Brian Alexander [Alexander2008] makes in the article, *“It's more than just a driving game – it's an experience.”* this indicates how realistic racing simulators should be created; it should give players a sense of experience and realism to real life racing.

Brian’s history goes back many years in racing. He obviously understands how racing on a track works and has experience it, therefore he can compare real life racing to racing in a racing game.

The article goes on to explain what Alexander [Alexander2008] understands in terms of physics and vehicle dynamics. He explains how a car engine is placed in a car, whether it’s a front, mid or rear engine. This indicates how the weight is distributed in a car, and how it can affect a car, by applying more grip and down force on either the rear or front tyres when the car is at different speeds and acceleration when the racer goes into and out of a turn. All of these facts should be considered when creating a racing simulator, obviously they show a sense of realism, and different cars should obviously drive in a different way, and have its own characteristics*.*

The statement below indicates how Gran Turismo implements this feature and adds the characteristics in most of the cars. It also shows how Gran Turismo has implemented the physics of handling a car around a corner by using the different style of cars characteristics.

*“There are several cars that mimic this performance in Gran Turismo 5: Prologue, but none do it better than the '96 Lotus Elise.”* Alexander, [Alexander2008].

Brian now discusses how Gran Turismo is compared to real life racing in terms of physics and vehicle dynamics. Alexander states *“Playing on ‘professional’ mode (real-world physics) with no electronic aids”* this determines how racing can cause cars to go out of control if the speed is too high going through a corner, but every mistake around a corner can be fixed in real world racing, but how? Brian explains that if the car does spin out, by applying the throttle again, will cause the car to straighten up, as Gran Turismo has implemented.

*“The level of precision you can achieve is surprisingly accurate”* Alexander, [Alexander2008].

The statement indicates how the accuracy and the precision of the physics is impressive, and how it replicates real life racing, as he states anyone can cause the car to slip on a turn, but if the player push it too far it will cause the car to spin off, *“just as you would in a real car.”* Alexander, [Alexander2008].

Brian Alexander obviously understands the concepts of racing in real life as shown in the article, which indicates his capability to recognize and demonstrate what a racing simulator should implement in terms of the vehicle dynamics physics engine. Understanding the characteristic of a car and a track is important to creating a realistic racing simulator, as different cars and tracks will affect how the physics and dynamics are applied to the vehicle.

The article is contributing a lot of praise to Gran Turismo, indicating how well the physics engine has been implemented, how accurate and precise the physics is compared to real life racing and how the game shows a good sense of realism to give players a sense of experience. However not everything is perfect, as Alexander [Alexander2008], states *“The one thing Gran Turismo doesn't accurately simulate is the terror of being in a race car”* this statement illustrates what a realistic racing simulator is missing, which is the true sense of experience on a track, the terror and fear when driving at high speeds, the risk of a person’s life if mistakes are made on a track, and the competition with other racers on the track. Theses thrills can push a racers personal limit to take more risks and put their own life in additional danger, but can a game give a player that same experience?

To give a player a true sense of fear and emotionally give them that thrill will indicate how realistic the racing simulation is. Realistic racing simulators are missing the capabilities to give a full racing experience, that professional’s racers experience every day. Overall Brian has explained how realistic a game can be and making a realistic racing game is to give a real life understanding of racing for the player. Even how pure and how accurate the physics engine can be, the emotions of a players thrill and terror will probably never be reached.

**2.1.5 Conclusion**

The discussions of these topic has been interesting and appealing to realism in racing simulators, it shows what people think about current racing games, and how they can be improved in terms of realism. The articles and forums shown indicates the comparison of realistic racing simulators, and by finding what the current market is demanding, shows why people prefer one over the other, and why publishers develop their games the way they do, so that they target the mass market, rather than the niche market for business terms.

Overall the physics engine is the main focus, and the pure vehicle dynamics physics engine for a racing simulator will be implemented in great detail, to give a player a fine experience in racing and to allow players to understand what real life racing is.

2.2 Software Methodologies

**2.2.1 Introduction**

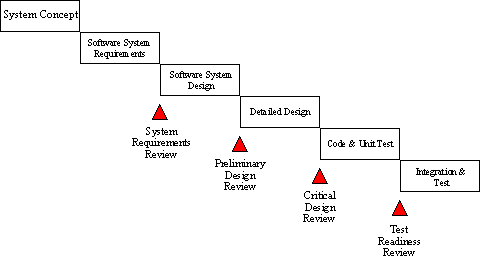
Software methodologies is a practice that helps plan and organize a team or individual developers to improve productivity and quality. The different methodologies use different techniques and methods to apply to a team or individual developers to achieve their end goals. There are many different styles in software development methodologies, each implemented and adapted in different ways to the project and the team, but the decision of the methodology used for the project depends on the project style and how the team members working styles are used effectively on the project. This research will determine the difference between the known software methodologies used in industry and how they are used to gain the best results in development. There will be some assessments on this project to determine which methodology suits this project and works best to achieve the end goal and the completion of this project.

**2.2.2 Standard Software Methodologies**

There are some generic methodologies that were implemented and adapted in industry back in the 1960s. These were known as the Sequential (Waterfall) and Cyclical (Spiral) methodologies, they were the basic design of software development, and later on new software methodologies were developed based on these two methodologies.

Sequential Methodology (Waterfall)

The waterfall methodology is separated into different stages of the development, and other stages cannot begin until the previous stage is complete. The stages tend to be analysis, design, implementation and testing, but this can vary depending on the development team and the project itself. This helps create a sequence of tasks that need to be re-evaluated at the end of each stage, which therefore increases the quality of development. As shown in Figure 2.



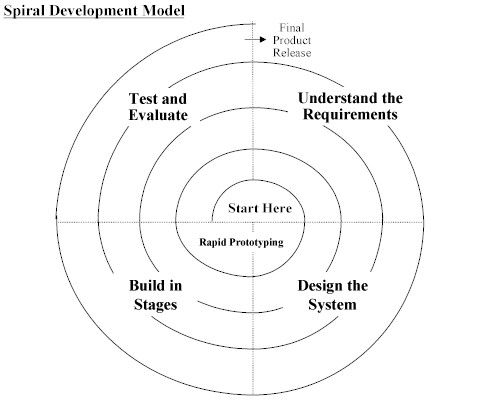
**Figure 2 A basic diagram showing the waterfall methodology in its different stages, [Chapman1997]**

This methodology only works well *“when the complexity of the system is low and requirements are static”* as stated by Ronald Burback [Burback1998]*.* It disciplines the developers to analyze and design the project before implementing any code, this allows project planning and organization to help let individual developers understand the process and the requirements, therefore developers will not be writing code that is not needed but is always focused on the main requirements, this will increase the quality of the product, and ensures that developers productivity is focusing on the requirements only.

There are problems with this methodology, as all decisions and designs must be decided up front, and there cannot be any changes made after the analysis and design stages, *“There is no room for mistakes and no process for correcting errors after the final requirements are released”,* as stated by Ronald Burback [Burback1998]*.* Therefore it is important that all decisions and requirements are made immediately before the development stages, but decisions are always inaccurate and may take longer than expected which leads to delays within the whole methodology as each stage cannot be started until the previous stages are completed and reviewed.

Cyclical (Spiral)

The cyclical methodology is similar to the waterfall methodology as it still has the same stages but it implements iterations through the project, so as an alternative of applying each stage and completing that stage before moving onto the next, the spiral allows the developers to gradually implement small amounts of each stage and then reiterate from the start after the end of each cycle. Therefore the developers are always testing and releasing prototypes. As shown in Figure 3.



**Figure 3 a basic diagram showing the spiral methodology [Chapman1997]**

The cyclical methodology works well because *“It allows for feedback from each team about the complexity of each requirement”,* as stated by Ronald Burback [Burback1998]*.* This allows mistakes to be fixed if there are problems with the requirements, so therefore developers won’t be in situations where requirements change in the middle of development and implementation needs changing. The methodology allows communication between the client and developers as it allows the client to look at the prototypes of the product and give the developers feedback and required changes.

The methodology also has issues when implemented, as there is no authority to control the cycle, which can lead to *“time schedule slips, missing features, or poor quality”,* as stated by Ronald Burback [Burback1998]*.* If requirements are not clear to developers and requirements are missing, the number of cycles will grow as the developers will need more iteration to implement the missing requirements. Each stage is implemented a small amount each iteration, the development team is *“never given a complete picture of the product”* as stated by Ronald Burback [Burback1998]*.* Therefore the development team will never understand and design the full architecture of the product, and may end up changing the architecture every cycle, which leads to the number of cycles increasing and the release date being delayed.

**2.2.3 Current software methodologies used in industry**

There are many different software methodologies adapted in the current computing industry, each works and implemented differently, dependent on the business and the product. From the 2000s businesses adapted new approaches where the methodologies were an iterative implementation, similar to the spiral methodology, they were known as Agile Software Development.

Agile development focuses on the team to achieve their personal, technical and organization skills, and adapts them into the project. Agile Software Development is a group of methodologies that follows a set of practices for iterative development. Following simple code standards and quality assurance practices are adapted to improve the quality and the productivity of the team, but also focus on customer’s needs and requirements, and adapt them into the project plan for implementation. Agile development consists of 4 main values and 12 main principles that every agile methodology should follow and adapt into the project, as shown on the Agile Manifesto site [Manifesto2001]. Throughout the years agile has become popular and companies and developers within the industry designed and developed their own agile methodology which are now used within many companies, but all the methodologies implement the same principles and practices. Some of the agile methodologies are listed on Jurgen’s site, [Jurgen2008]. This research is to discover and compare some of these methodologies and to investigate which one is best for this project. The research will focus on three main methodologies that are known as Extreme Programming (XP) and Rational Unified Process (RUP), which are all well known and used in industry today.

Extreme Programming (XP)

Extreme Programming was created by Kent Beck [Beck2000] as he states “XP is a lightweight methodology for small to medium sized teams” which indicates the methodology is used in small teams, but this is an individual project, therefore implementing this methodology may encounter problems. The methodology takes processes and stages to an extreme level; it was distinguished from other methodologies as stated in Kent Beck’s book on page xvii [Beck2000]. The methodology uses an iterative process and regards “*ongoing changes to requirements*” as stated by [Jurgen2008], it uses feedback from customers and personal testing to acquire changes to the project plan and implementation. The methodology is separated into different stages involving different members of the team, the different stages are known as Planning, Analysis, Design & Coding, Testing and Deployment.

Planning is the initial stage of the project where the programmers and the business experts are put together to develop a project plan, as stated by Shore & Warden [ShoreWarden2008], the business experts “*point the project in the right direction*” and the programmers “*provide estimates and suggestions*”.

Analysis is the phase to determine the requirements of the software, which are determined by the business experts for example customers, and the programmers simple ask for information from the customer. The customers “*are responsible for organizing their work so they are ready when programmers ask for information*” as stated by Shore & Warden [ShoreWarden2008].

Design & Coding is where the programmers design the architecture of the software, but XP “*uses incremental design and architecture to continuously create and improve design in small steps*” as stated by Shore & Warden [ShoreWarden2008]. To perform this process, pair programming is introduced as stated by Kent Beck [Beck2000], “*Pair programming is a dialog between two people trying to simultaneously program*” which indicates both members are developing and testing to improve quality of code.

Testing is the stage where the programmers, customers and testers are testing the software for bugs and issues. Shore & Warden [ShoreWarden2008] states “*Well functioning XP teams produce only a handful of bugs per month*” which indicates the quality of development by following this methodology. The programmers produce “*automated unit and integration tests*” as stated by Shore & Warden [ShoreWarden2008], which allows the programmers to ensure the software is following the requirements and the intent of the software. The customers then test the software to ensure the software is working and the developers are following the customers’ expectations and requirements.

Deployment is where the team prepares to deploy their software to the customers at the end of the iteration to be tested and reviewed in a weekly demo.

This progress is reiterated several times until the end of the project, but planning tends to be a big factor during the start of the project and testing and deployment towards the end, but each cycle should continue to adapt to each of the phases. In the Shore & Warden [ShoreWarden2008] book Table 3.1 on page 21 indicates the simple XP phases and practices.

Rational Unified Progress (RUP)

Rational Unified Progress (RUP) was created by a division part of IBM during the year 2003. The methodology provides “*a disciplined approach to assigning tasks and responsibilities*” as stated by Krutchen [Krutchen2003]. The methodology uses an iterative approach as Krutchen states “*iterations are planned in number, duration and objectives*” which indicates how the methodology is based on achieving objectives and targeting aims at each phase and cycle of the process. Jurgen [Jurgen2008] states “*bloated and too costly to customize for small projects*” which indicates how RUP consists of a problem when the project is small and consists of a small team, as for this project the methodology is based on team development but this is an individual project. RUP is defined and modelled by “*who is doing what, how and when*” as stated by Krutchen [Krutchen2003], which defines five primary elements involved to build this model.

Roles indicates the who criteria. A role is defined as the responsibilities of an individual within a team working together. Krutchen [Krutchen2003] states “*each role is associated with a set of cohesive activities*” which indicates the role selection for the individual will be selected by who will perform best for the role. “Roles are not individuals, nor job titles” as stated by Krutchen [Krutchen2003], which explains how roles can be changed throughout the project process.

Activities indicates the how criteria. Krutchen [Krutchen2003] states “*Roles have activities, which define the work they perform*” which specifies that an activity is a selection of task and work for a specific role, they are tasks set for individuals who are performing that role to produce results for the project.

Artefacts indicates the what criteria. An artefact is “*a piece of information that is produced, modified, or used by a process*” as stated by Krutchen [Krutched2003]. The individuals use these artefacts to perform the activities given from their role. The artefacts help progress towards the projects objectives and final product. The artefacts tend to normally subject to version control but mainly the container artefacts as Krutchen [Krutchen2003] states “*Artefacts can also be composed of other artefacts*”.

Workflow indicates the when criteria. A workflow is “*a sequence of activities that produces a result of observable value*” as stated by Krutchen [Krutchen2003] which indicates how the workflow controls and organizes the process of each activity within the project and for each individual role. The workflow helps organize the activities to produce meaningful and valuable results to achieve the projects objectives and final product.

Disciplines is the main container of the four elements in this model, it helps organize the activities into processes. There are nine main disciplines, where six are defines as technical and the three as supporting, which are shown in Krutchen’s book [Krutchen2003] on page 30.

The process of RUP uses iterations each phase to reach its end target, but by giving each member of the team a specific role to perform and applying the disciplines then the team is able to strictly perform tasks towards the projects objective.

**2.2.4 Comparing the Methodologies**

XP and RUP seem similar that they are both iterative development with requirements produced by the customer throughout the process of the project, but in some cases they can be different in many ways. Nicolette [Nicolette2005] states “*RUP is intended to support iterative development within the framework of a traditional Waterfall process, while XP is intended to support Agile development*” which indicates how the methodologies differ, and how they are implemented in a team of developers. The difference shows how each of the phrases and iterations can be processed as they follow different rules and criteria’s to reach its end goal.

XP and RUP has different amount of roles as RUP actually specifies more different roles within the development of the project, but the difference on why the roles differ are explained in this article by John Smith [Smith2003] as he states “*XP is not covering all of the RUP disciplines*” and “*XP roles are actually closer to positions than RUP roles*”.

In terms of disciplines and workflow, RUP are defined in [Krutchen2003] on page 30, and the XP only uses the basic activities which are coding, testing, listening and designing, which is performed using a specific set of practices noted in [Beck2000] and in [Smith2003] on page 15. Smith [Smith2003] discusses how the RUP structure uses a Requirements discipline, but XP does not need this stage or activity as requirements are defined by the business experts as Smith [Smith2003] states “*XP- the customer defines and provides the requirement*s”.

The different activities shown in RUP and XP are fairly similar in terms of coding and testing, but XP uses practices like test-first programming and integrated testing where RUP is “*to develop executable tests that mimic the actual behaviour of the code*” as stated by Nicolette [Nicolette2005], but however RUP is capable to incorporate some of the practices XP has developed into the RUP development process.

RUP has a large amount of artefacts compared to XP, but XP doesn’t need that many artefacts as RUP has more disciplines to implement therefore more artefacts and processes. Smith [Smith2003] describes how some of the disciplines can be outside the scope of XP so therefore they do not concern the XP process, so artefacts in RUP can be ignored in XP. In the article [Smith2003] page 12 indicates some of the comparison of RUP and XP in terms of artefacts.

RUP and XP show a significant amount of differences but as shown by the articles and books indicate how RUP is more complicated and requires a large amount of time to develop and is really used for large projects, as for XP is for simpler and small projects, which requires less time to complete the project, but requires business experts to drive the goals and objective of the project. The uses of either methodology can be useful to this project, but both methodologies are either targeted for a large project or a team based project, which this is not, but an individual project, therefore both of these methodologies would not work very well for this project, unless they can be manipulated for specifically use of a single person.

**2.2.5 Personal Extreme Programming**

Personal Extreme Programming is similar to extreme programming but this is applied for a single person instead of a team of developers. This was introduced and created by Ravikant Agarwal and David Umphress, as their article [AgaUmp2008] shows how analyzing the key features of XP can be altered or adapted for a single person.

XP focuses on basic team work within small teams, and uses practices like pair programming and code reviews as part of the methodology which requires other team members. In cases for independent projects, like this project, where pair programming becomes unnecessary as a coding partner is unattainable therefore implementing XP itself can cause issues during each phase and practice, so a single person methodology would be perfect for this project.

Agarwal and Umphress [AgaUmp2008] stated “*we scale XP to produce a methodology that uses its practices, but in a form that an individual within a traditional project framework can use*” which indicates how they used the XP practices and by modifying the specific practices that require more than one person, produces practices where only a single person is required, they named the methodology PXP (Personal Extreme Programming).

Table 1 on the second page (83) of the article [AgaUmp2008] indicates how PXP is scripted and used in the different stages. The requirements and features are based on stories written by the user, each story represents small details of the requirements to be implemented where the developer simply chooses a story and process it, this becomes iterative to represent a customer giving feedback and requirements back to the developer. This is created within the planning stages. [AgaUmp2008] states “*developer maintains three baselines, namely development, refactor and production*” which indicates how each phase are completed in a sequential order, so development the code is written and maintained by versioning, and integrated testing is applied, then the phase changes to re-factor, where the developer changes or fixes any bugs found from the tests, and finally production to release the product.

Table 2 on the third page (84) of the article [AgaUmp2008] shows the comparisons of the practices between a standard XP methodology to a PXP methodology, the table indicates how some of the practices can be similar and same different to maintain the purpose of the methodology. For example pair programming is where two developers are programming together on one machine, which therefore help improve quality and understanding of code, but with PXP pair programming is impossible but [AgaUmp2008] states “*ask a colleague to give you reminders often*” which indicates for PXP the developer can ask for assistance and advice constantly to ensure quality and understandings of code.

PXP has been tested but “*has not yet been applied to a large audience*” as stated by Agarwal & Umphress [AgaUmp2008], which indicates a risk to use this methodology but from the test they did implement, it seems like the methodology was successful. Agarwal & Umphress [AgaUmp2008] states “*PXP can still be refined and improved*” which explains that the methodology is not perfect and is looking for improvements, but the main idea of this methodology is to allow a single developer to follow the XP practices and the important aspects of XP without running into issues when team work is required. This methodology simplifies XP by refining the values of team development.

**2.2.6 Conclusion**

The overall research has been useful to understand the concepts of software development, and how the industry of computing implement these methodologies within their organization to help developers work together to archive their objectives and production of their product. The methodologies ensure quality development and teamwork, but for this project it is an independent project so no team is required, but the majority of methodologies are implemented and created for team purposes. Each methodology has their own advantages and disadvantages but only some are best for this project, and from the research it seems like extreme programming was the best methodology as its based for small projects and is using an iterative development, where the rational unified process seems to advance, and is more likely for large projects and teams to implement that methodology. Time scale is also a factor why extreme programming would benefit this project compared to rational unified process.

Overall extreme programming is a team based methodology and with Agarwal & Umphress [AgaUmp2008] methodology, personal extreme programming, which is perfect for this project, as it’s a methodology based on the practices that extreme programming implement but for an individual project, even though the methodology hasn’t been tested fully for a large audience the practices and ideas make perfect sense to implement, so therefore this project will test this methodology itself.

2.3 Platforms

**2.3.1 Introduction**

There are many different platforms for developing software and games within the current computing industry; depending on what the company develops they tend to choose the best suitable platform to develop on for their main product. Platforms can expand to a wide range from deciding which language base to use and what frameworks or API’s can work best with the chosen code base. The main focus is to look into current games industries and see what platforms they use to develop their games, and also to discover what libraries and frameworks are available to assist the development of this project.

**2.3.2 Top 10 Programming Languages**

The computing industry is advancing and has established a lot of different programming languages, but which one is the current most popular? TIOBE [Tiobe2010] is a website that monitors the current computing industries programming languages, and gives a monthly feedback on which language is most popular; this is worked out based on the number of skilled engineers and searches within the most popular search engines. Below shows only the top 10 languages from TIOBE [Table 2], visit this website to view the full table.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Position Dec 2009 | Position Dec 2008 | Delta in Position | Programming Language | Ratings Dec 2009 | Delta  Dec 2008 | Status |
| 1 | 1 | http://www.tiobe.com/tiobe_index/images/Same.gif | [Java](http://www.tiobe.com/content/paperinfo/tpci/Java.html) | 17.061% | -2.31% | A |
| 2 | 2 | http://www.tiobe.com/tiobe_index/images/Same.gif | [C](http://www.tiobe.com/content/paperinfo/tpci/C.html) | 16.285% | +0.12% | A |
| 3 | 4 | http://www.tiobe.com/tiobe_index/images/Up.gif | [PHP](http://www.tiobe.com/content/paperinfo/tpci/PHP.html) | 9.770% | +0.29% | A |
| 4 | 3 | http://www.tiobe.com/tiobe_index/images/Down.gif | [C++](http://www.tiobe.com/content/paperinfo/tpci/C__.html) | 9.175% | -1.72% | A |
| 5 | 5 | http://www.tiobe.com/tiobe_index/images/Same.gif | [(Visual) Basic](http://www.tiobe.com/content/paperinfo/tpci/(Visual)_Basic.html) | 7.778% | -1.70% | A |
| 6 | 6 | http://www.tiobe.com/tiobe_index/images/Same.gif | [C#](http://www.tiobe.com/content/paperinfo/tpci/C_.html) | 6.258% | +1.61% | A |
| 7 | 7 | http://www.tiobe.com/tiobe_index/images/Same.gif | [Python](http://www.tiobe.com/content/paperinfo/tpci/Python.html) | 5.185% | +0.62% | A |
| 8 | 9 | http://www.tiobe.com/tiobe_index/images/Up.gif | [JavaScript](http://www.tiobe.com/content/paperinfo/tpci/JavaScript.html) | 3.515% | +0.45% | A |
| 9 | 8 | http://www.tiobe.com/tiobe_index/images/Down.gif | [Perl](http://www.tiobe.com/content/paperinfo/tpci/Perl.html) | 2.692% | -0.91% | A |
| 10 | 11 | http://www.tiobe.com/tiobe_index/images/Up.gif | [Ruby](http://www.tiobe.com/content/paperinfo/tpci/Ruby.html) | 2.653% | +0.34% | A |

**Table 2 Top 10 programming languages of month December 2009 [Tiobe2010]**

**2.3.3 The Game Industry**

The current games industry is expanding and is always looking for talented young developers, but they are always looking for specific programming skills from developers. The games industry is always focusing on C++ developers and being able to use the language fluently and to be able to adapt to the language complexity. From the jobs listed in the references from EA [EAJobs], Travelers Tales [TTJobs], and Blizzard [BlizzardJobs]. It shows the jobs require strong C/C++ programming skills, but these are for the large games distributers. Small companies that develop games for websites tend to use Flash or Java as it’s an easy way for developers and games enthusiasts to develop and release their own games. Mobile games tend to use Java as its base language, but with the success of the iPhone Objective C has become a popular language to develop applications and games for the iPhone. As shown by the research, there is a wide range of development within the games industry that uses different platforms depending on the system the game will be running on. This project is focusing on a windows game, so one of the best options is either using C++ or Flash, but with the rise of a new framework that uses C# as its base, known as XNA, which makes complex games easy to develop and release. Currently C# is not an established language within the games industry as shown in the article by DoolWind [Dool2009], due to the performance, as consoles like the Xbox need fast execution but with C# it can be slower than C++, without the XNA frameworks assist.

**2.3.4 The Language**

From what has been found within the games industry, the majority of the companies tend to use C++ as its main base language, but with the new XNA framework, developers and companies can go into the C# and .NET language base, but what is actually stopping these big companies from moving into that direction.

C++ is a high level language and is defined by its predecessor C, but C++ supports the aids of object oriented and generic programming but the language is still unmanaged as it does not control or manage the applications memory in the heap or stack, therefore the developer always has to control and manage the objects instantiated that will need garbage collecting within the duration of the running application.

C# is another high level language but is a managed coding base unlike C++. The language supports object oriented, generic and dynamic programming and it allows the user to develop applications easily. The language is more productive as it supports goods designs and allows developers to use features like interfaces, delegates and generics, which makes the development of the application cleaner.

|  |  |
| --- | --- |
| Advantages of using C++ or C# | |
| C++ | **C#** |
| Better performance | Garbage Collection |
| Portability | Array Bounds Checking |
| Multiple inheritance | Types have a defined size |
| Offers pointers | No need for headers files and #includes |
| Support for global variables, functions, constants | Supports properties |
| Deterministic destruction | Built in support for threads |
| Implicit interfaces on generics | Static constructors |
| Support for macros | Conditional functions |

**Table 3 Advantage of using C++ or C# [Suess2007]**

Table 3 shows the pros of both languages and determines the difference between the languages and why the languages are and are not established within the games industry. Both languages are different and shows there best features in different ways, and how one language can implement something better than the other. The table only shows a small amount of advantages; visit this website article by Suess [Suess2007] for the full set of advantages.

**2.3.5 The Frameworks**

C# and C++ uses many different frameworks, but which frameworks works best to develop games on and which supports an easy way of developing games. The main 3D libraries used within the games industry are DirectX and OpenGL where the Xbox supports DirectX and OpenGL supports the Playstation. Both libraries are powerful and have a wide range of functionality to compute graphical applications.

DirectX is an interface that supports the computer system graphics for games and allows the developers to communicate with hardware using the COM supports. DirectX also contained a library SDK that exposes these graphical API’s to allow developers to build games and multimedia applications for windows. The SDK supports C++ and C#, but is mainly used with C++ in the current games industry. The Microsoft Xbox 360 uses a version of DirectX9 to support the games developed for the Xbox and the hardware in the console.

OpenGL is similar to DirectX as it also supports computer systems graphics for games development, and allows developers to develop games using the OpenGL 3D library. OpenGL was created in 1992 and was a popular graphical API used to develop games for the many platforms. Unfortunately OpenGL was advancing slowly compared to the graphic hardware’s, and DirectX came into play and was more stable and was growing as technology grew.

|  |  |  |
| --- | --- | --- |
| Comparing OpenGL & DirectX Features | | |
| Feature: | OpenGL | DirectX |
| Vertex Blending | N/A | Yes |
| Multiple Operating Systems | Yes | No |
| Extension Mechanism | Yes | Yes |
| Development | Multiple member Board | Microsoft |
| Thorough Specification | Yes | No |
| Two-sided lighting | Yes | No |
| Volume Textures | Yes | Yes |
| Hardware independent Z-buffers | Yes | No |
| Accumulation buffers | Yes | No |
| Full-screen Ant aliasing | Yes | Yes |
| Motion Blur | Yes | Yes |
| Depth of field | Yes | Yes |
| Stereo Rendering | Yes | Yes |
| Point-size/line-width attributes | Yes | Yes |
| Picking | Yes | No |
| Parametric curves and surfaces | Yes | Yes |
| Cache geometry | Display Lists | Vertex Buffers |
| System emulation | Hardware not present | Let app determine |
| Interface | Procedure calls | COM |
| Updates | Yearly | Yearly |
| Source Code | Sample | SDK Implementation |

**Table 4 Comparison of OpenGL and DirectX SDK features**

Table 4 shows the comparison of features available to users, when developing with either SDK. The API’s have their advantages and disadvantages, but obviously shows equal features that are similar. The table is just a small subset of features; the full list is shown on this website, where they discuss the comparisons and features.

**2.3.6 The XNA Framework**

The XNA Framework is a new and innovative framework which was created by Microsoft; it allows developers to implement and create games very easily, it also gave game developers the opportunity to upload their games to the Xbox live network for other people to both purchase and play the game. The Framework uses the C# and .NET language as its base, which gives the developers a simple and efficient language to use to develop their games. The XNA Framework is actually built on the DirectX Framework and uses the exact same features and functionality in the DirectX API, but the XNA just manages the framework with better and more user friendly interfaces to access the DirectX Framework.

|  |  |
| --- | --- |
| Advantages of using C++ with DirectX or C# with XNA | |
| C++ with DirectX | **C# with XNA** |
| Aims towards jobs in the games industry | Upload games to the Xbox 360 |
| Better performance | Using C# which is managed and easier to develop on |
| Access advance Direct3D calls | Easy to set up a application |
|  | Can use HLSL to create effects |
|  | Less time consuming |

**Table 5 Advantages of using C++ with DirectX or C# with XNA**

Table 5 shows the comparison of C++ with DirectX and C# with XNA. XNA does have a more advantages than C++ with DirectX, but the advantages are based on the codes language capabilities and the language future prospects.

**2.3.7 Code Samples**

Here are some samples of applications, which will determine and compare the complexity of developing a 3D application using either C++ with DirectX or C# with XNA.

C# & XNA With XNA the IDE actually creates the whole project setup and the Game Loop and other game programming concepts are created with it, it should just show a simple window when the application is running.

C++ & DirectX C++ uses the Win32 project to create the base of the window, and then the DirectX calls needs to be setup before the project can use any of the DirectX calls. Following the steps from the DirectX documentation [DXSDK], there are a lot of steps that is carried out to create the DirectX window.

* Create the Device and Swap Chain
* Create the Render Target View
* Create the Depth/Stencil Buffer and View
* Set the Viewport

**2.3.8 Conclusion**

Overall every language discussed and reviewed are similar and have their own advantages and disadvantages compared to each other. There are many different languages and each one needs to be learnt and adapted differently, but with C++ there is an immense learning curve as it is a complex language to learn and understand, but is still a powerful language, as it is still used and adapted in present day game companies. C# is a very easy language to learn and adapt, but it isn’t as complex as C++ and doesn’t have the performance power that C++ has, but defiantly has the framework to develop easy and simple games that can be released into the market.

The frameworks used are complex and yet again there is a learning curve, but being able to understand the frameworks on how they work will help decide which framework is best for this project. The project is focusing on realism in racing games, and is really focusing on developing a physics engine and library and apply it into a game, so a easy and quick way to develop a 3D application to demonstrate the physics engine is to obviously use the XNA framework, as shown in the code samples, the framework is easy to use, and creating a simple scene with different meshes is straightforward, therefore more time can be allocated to develop the physics engine and library.

**2.4 Vehicle Dynamics**

**2.4.1 Introduction**

The physics and dynamics that act on a car whiles moving or stationary, can affect the cars acceleration and velocity. The forces acting on the car can dynamically change the cars direction and flow. Material of the road and density of the air can also affect the cars drive force. The car’s engine and transmission efficiencies can affect the cars performance. These concepts all come under the topic of Vehicle Dynamics, and this is the core research that will help develop the application to demonstrate real world physics for a racing simulator.

**2.4.2 Vehicle Specifications**

The key information to help calculate the vehicle dynamics for a specific car is to understand the facts and figures of a cars specification, as the values from the specification may be used in some of the equations and formulas. Table 6 shows the specification for a Mercedes SL500 [JBCar], the specification data shown in Table 6 on reflects the most common details that are used in calculations.

|  |  |  |
| --- | --- | --- |
| Name | Value | Description |
| Curb Weight | 1835 kg | The curb weight is a description of mass, can be used to calculate weight, which is mass \* gravity |
| Wheelbase | 2560 mm | The wheelbase is from the centre of the front wheel to the centre of the rear wheel length; this is used to calculate the distribution of weight. |
| Weight Distribution Front | 51 % | This is the front centre wheel to the centre of mass of the car, also used to calculate the weight distribution. |
| Weight Distribution Rear | 49 % | This is the rear centre wheel to the centre of mass of the car, also used to calculate the weight distribution. |
| Tires | 255/45 ZR17 | This is a description of the tires used by the car, the information can be compiled to give the tires radius, therefore gives the wheel radius |
| Max Power | 302 Bhp | This value determines the maximum amount of power the car can reach |
| Max Power at RPM | 5600 rpm | This is the value of the RPM when the car is at max power, this can be used to calculate the power equation |
| Max Torque at RPM | 2700 rpm | This is the value of the RPM when the car is at max torque, this can be used to calculate the torque equation |
| Gear Ratios 1st | 3.59 | Ratio of the cars first gear, which helps calculate the cars torque. |
| Gear Ratios 2nd | 2.19 | Ratio of the cars second gear, which helps calculate the cars torque. |
| Gear Ratios 3rd | 1.41 | Ratio of the cars third gear, which helps calculate the cars torque. |
| Gear Ratios 4th | 1.00 | Ratio of the cars forth gear, which helps calculate the cars torque. |
| Gear Ratios 5th | 0.83 | Ratio of the cars fifth gear, which helps calculate the cars torque. |
| Final Drive | 2.82 | Ratio that determines the final push of the car’s engine. |
| Drag Coefficient | 0.29 | The cars body drag coefficient, which helps determines the cars drag resistance. |

**Table 6 Specification of an Mercedes SL500 [JBCar]**

**2.4.3 Basic Calculations**

Basic calculations look into the equations and formulas to make the car move from one position to the next. The key calculation is to determine the cars speed which then determines the cars distance, and this is done by calculating the acceleration, velocity and displacement of the car.

Newton’s [Newton] second law of motion states:

* F = ma;

This can determine the acceleration of a car by rearranging the equation to:

* a = F / m;

By using the specification of the SL500 the mass of the car is 1835kg, and by applying a force to the car would establish the acceleration of such car. For example, if the force of the car was set to 5000 N.

* a = 5000 N / 1835 kg
* a = 2.724 ms²

Once the acceleration is calculated the velocity of the car can be determined as Marco Monster states “*The car's velocity (in meters per second) is determined by integrating the acceleration over time*”, this indicates that the velocity is determined by applying the acceleration over a certain amount of time to the initial velocity of the object.

* v = u + at;

Where v is the new velocity, u is the initial velocity, a is the acceleration and t is the time.

From this equation, the velocity can be determined by using the acceleration calculated from the previous calculation, so for this example the initial velocity is at 10 ms and t is 1 second.

* v = 10 ms + 2.724 ms² \* 1 second
* v = 12.724 ms

Once the velocity is calculated the displacement can be calculated, as Brian Beckman [Beckman1991] states “*Just as acceleration is the rate of change of velocity, so velocity is the rate of change of distance over time*”, as this indicates that to calculate the displacement of an object is to integrate the velocity over time on the initial displacement of the car.

* d = d + vt;

Using the velocity calculated and setting the time to 1 and the initial displacement to 0, the displacement can be calculated.

* d = 0 m + 12.724 ms \* 1 second
* d = 12.724 m

This shows the displacement of the car from 0 to 12.724 M at a velocity of 12.724 ms, and this was calculated from the acceleration. Therefore the force acting on the car, which causes the car to move, is important to calculate as it can dynamically change the acceleration.

**2.4.4 Torque**

Torque is used to calculate the rotational power that causes the car to move. The calculation uses a torque equation to establish what the cars torque is at a specific gear ratio. Below shows an equation from [Jazar2008] which indicates the equation to calculate the Torque.

* Te = P1 + P2we + P3 we²

The values for P1, P2 and P3 can be calculated using the power equation as shown below [Jazar2008].

* P1 = Pm / Wm
* P2 = Pm / Wm²
* P3 = - Pm / Wm³

Where Pm is the max power (W), and Wm is the angular velocity (Rad/s)

Applying this to the specification of the SL500, the Torque equation can be created. Pm for the SL500 is 302 bhp and Wm is 5600 rpm.

* P1 = 225201.4 / 586.43 = 384.02 W/s;
* P2 = 225201.4 / 586.43² = 0.6548 W/s²;
* P2 = - 225201.4 / 586.43³ = -1.1167 x 10 ˉ³ W/s³;
* Te = 384.02 + 0.6548 we + -1.1167 x 10 ˉ³ we²;

The equation for torque is now defined, and by giving the equation a engine angular velocity (we) can determine the Torque. The engine angular velocity can be calculated by using the formula as shown below, this is known as the “*speed equation*” [Jazar2008].

Where ni is the current gear ratio, nd is the final drive, Rw is the radius of the wheel, and vx is the current velocity of the car. From the spec for the SL500, the first gear ratio is 3.59, the final drive ratio is 2.82 and the velocity is at 12.724ms. The wheel radius is calculated by using the tyre information, the details on how to read the tyre information is shown [CarBible2010].

* Rw = ((255 \* 0.45) + (431.8 / 2)) / 1000= 0.33 M
* we =( (3.59 \* 2.82 )/ 0.33 M ) \* 12.724 ms = 390.35 ms

By setting we to the result, the Torque of the car can now be found.

* Te = 384.02 + 0.6548 \* 390.35 + -1.1167 x 10 ˉ³ \* 390.35²;
* Te = 384.02 + 255.6 + -170.155
* Te = 469.5 Nm

**2.4.5 Forces**

From the previous section, to calculate the acceleration a force is needed, and this force acts like the drive force on the car. This force is created by a force known as traction force, “*the force delivered by the engine via the rear wheels*” as stated by Marco Monster [Monster2010], as torque is delivered to the rear wheels, the force on the tyres and road causes the wheels to push backwards, and stated by Newton’s [Newton2010] third law, there is always an equal and opposite force, and it’s the opposing force that causes the car to push forwards. Traction force is created by calculating the engines torque, as calculated in the previous section. The equation of for the drive force is shown below.

* Fdrive = Te \* xG \* xD \* n / Rw

Where Te is the engine torque, xG is current gear ratio, xD is final drive ratio, n is transmission efficiency and Rw is the wheel radius.

The transmission efficiency indicates the energy lost when changing gear, as Marco Monster [Monster] states “*As much as 30% of the energy could be lost in the form of heat*”, which indicates the efficiency of a typical engine is at 70% so n = 0.7 factor.

From using the information and calculation from the previous section the drive force can be calculated as shown below.

* Fdrive = (469.5 Nm \* 3.59 \* 2.82 \* 0.7) / 0.33 M
* Fdrive = 10082.4 N

By applying this force to Newton’s second law, the acceleration can be determined more accurately.

* a = F / m
* a = 10082.4 N / 1835 kg
* a = 5.49 ms²

Marco Monster [Monster] also stated “*If this were the only force, the car would accelerate to infinite speeds*”, this statement indicates that there is always an opposing force to limit the cars velocity and acceleration, these forces are known as the Resistance forces. The two key resistance forces are known as Air resistance and Rolling resistance.

Air resistance is scaled by velocity and depends on the cars speed, the greater the speed the greater the resistance, as Marco Monster [Monster] states “*This force is so important because it is proportional to the square of the velocity*”. The calculation for air resistance seems to be a constant formula, as Marco [Monster], Brian [Beckman1991] and Ted [Zuvich] all state the same equation to calculate air resistance, which is shown below.

* Fdrag = 0.5 \* cD \* A \* rho \* v²

Where cD is the coefficient of friction for a specific car, A is the frontal area of the car, rho (ρ) is the density of air, and v is the velocity of the car.

From the specification the frontal area of the SL500 is 2 m², the coefficient of friction is 0.29 and the velocity is 12.724ms. The typical value for air density is 1.29 kg/m³ as calculated by Beckman [Beckman1991]. From this information and data the drag resistance can be calculated.

* Fdrag = 0.5 \* 0.29 \* 2 \* 1.29 \* 12.724²
* Fdrag = 60.6 N

This force would oppose the direction of the car, so if the direction is always a positive value, then this force is a negative value.

Rolling resistance is caused by “*the tire’s resistance to deformation, scrubbing losses in the contact patch, tire slip, and air drag on the tire*” as stated by Ted Zuvich [Zuvich] this statement indicates what properties on a car can alter the rolling resistance. The typical main cause of rolling resistance is “*by friction between the rubber and road surface as the wheels roll along and friction in the axles*” as stated by Marco Monster [Monster].

The calculations for the rolling resistance seems vary between the documents and research papers found for vehicle dynamics. Marco Monster [Monster] states “*This means Crr must be approximately 30 times the value of Cdrag*”, so therefore Monsters rolling resistance equation is:

* Frr = 30 \* CDrag \* v;

Where CDrag is the constant factor for air resistance, and v is the velocity of the car. Marco Monster states “*I couldn't confirm its value anywhere*” which indicates the 30 multiplier could not be proven, so this shows an inaccurate way of calculating the rolling resistance.

If the velocity is at 12.724ms and CDrag is 0.3741.

* Frr = 30 \* 0.3741 \* 12.724
* Frr = 142.8 N;

Ted Zurvich [Zurvich] states his equation for rolling resistance to be:

* Rx = fR \* W;

Where fR is the rolling resistance coefficient and W is the weight of the car. The rolling resistance coefficient is then calculated by using some of the factors of the car that can affect the coefficient of rolling resistance. Zurvich [Zurvich] states “*Once estimate such treats velocity as the significant variable, and provides linear speed dependence*” which shows how the velocity is used as the factor of rolling resistance.

* fR = 0.01(1+v/100);

Where v is the speed in mph

If v is at 12.724ms (28.5 mph) and the weight is 1835 kg \* 9.81 ms² = 18001.35 N.

* fR = 0.01(1 + 28.5 / 100);
* fR = 0.01285;
* Rx = 0.01285 \* 18001.35;
* Rx = 231.3 N;

The final way to calculate rolling resistance is by going back to basics, where the resistance is caused by the coefficient of friction depending on the surface of the road [ETB2010]. The formula for the resistance is:

* fR = cW;

Where c is the rolling coefficient and W is the weight of the body, the typical value for rolling coefficient for a car on concrete road is 0.01 – 0.015. Using these values and the specification for the car, the rolling resistance can be calculated.

* fR = 0.015 \* 1835 kg \* 9.81 ms²
* fR = 270 N;

From the three calculations of the rolling resistance, the three results were different, and these results can change the effects on the car, using [ETB] calculation is basic, but it will remain constant, as speed or other dynamic effects on the car is not taken into account. Zurvich [Zurvich] calculation is taking speed into account, so therefore the resistance will vary depending on how fast the car is going, and from the equation the faster the car goes the higher the rolling resistance. Finally Marcos Monster [Monster] formula takes the speed and the drag resistance into account, but ignores the weight of the car, and also has an unexplained multiple factor of 30.

The overall force that determines the cars acceleration and velocity is the sum of all three forces, where traction force is a positive value, and the resistance forces are negative.

* Flong = Fdrive + Fdrag + Frr

Using some of the values calculated, the total force can be determined, so Fdrive is at 10082.4 N, Fdrag is at -60.6N and Frr for this example is set to -270 N.

* Flong = 10082.4 + -60.6 + -270;
* Flong = 9751.8 N;

Therefore the acceleration is:

* a = F / m;
* a = 9751.8 N / 1835 kg
* a = 5.31 ms²

**2.4.6 Braking**

When a car brakes, a large force acts on the car opposing the drive force, therefore the car begins to decelerate and slow down. Marcos Monster [Monster] states “*When braking, the traction force is replaced by a braking force which is oriented in the opposite direction*”, which means the calculation in the previous section for the total force acting on the car is used to determine the brake force, instead of adding the traction force, the calculation adds a negative brake force.

* Flong = Fbraking + Fdrag + Frr
* Fbraking = kW;

Where k is the coefficient of friction and W is the weight of the car. The coefficient of friction is set to 0.62 and the weight of the car is 1835 kg \* 9.81 ms². Using this information the braking force can be calculated.

* Fbraking = 0.62 \* 1835 kg \* 9.81 ms²
* Fbraking = 11160.8 N

Applying this force to determine the full net force will help calculate the deceleration rate of the car when the user applies the brakes.

* Flong = -11160.8 + -60.6 + -270;
* Flong = - 11491.4 N

Using this force can calculate the deceleration rate.

* a = F / m;
* a = -11491.4 N / 1835 kg
* a = -6.26 ms²

**2.4.7 Weight Transfer**

The weight transfer can help give a visual effect on games, but can also determine the cars slip ratios. Marcos Monster [Monster] states “*When braking hard the car will nosedive.  During accelerating, the car leans back*”, this effect is caused by the transfer of weight as the down force on the front wheels will be greater than the rear wheels when braking, and during acceleration the rear wheels down force would be greater than the front wheels, which causes the effect of the car to lean back or forwards.

The calculation for the front and rear wheel down force is shown below [Jazar2008].



Where a2 is the rear track distribution, a1 is the front track distribution, l is the wheelbase, h is the height from the centre of mass to the ground, and a is acceleration. The information needed to calculate the forces, can be taken from the cars specification. For the SL500:

* l = 2560 mm;
* a2 = 49% of l;
* a1 = 51% of l;
* h = 1.0 m;
* m = 1835 kg;

If the acceleration is set to 0 the dynamic part of the equation is also 0, and only leaves the static part of the equation, which means there is no extra forces acting on the car to change its original down force.

* Ffront = 0.5 \* 1835 \* 9.81 \* 0.49 = 4410.33 N
* Frear = 0.5 \* 1835 \* 9.81 \* 0.51 = 4590.34 N

If acceleration is set to 5.31 ms², so the car is accelerating forwards.

* Ffront = 4410.33 – 0.5 \* 1835 \* 9.81 \* (1.0 / 2.56) \* (5.31 / 9.81);
* Ffront = 2507.23 N;
* Frear = 4590.34 + 0.5 \* 1835 \* 9.81 \* (1.0 / 2.56) \* (5.31 / 9.81);
* Frear = 6493.44 N;

From the calculations the rear down force is greater than the front wheel down force when the car is accelerating forwards. If the car brake’s it causes a deceleration, so if the deceleration is -6.26 ms²:

* Ffront = 4410.33 – 0.5 \* 1835 \* 9.81 \* (1.0 / 2.56) \* (-6.26 / 9.81);
* Ffront = 6653.9 N;
* Frear = 4590.34 + 0.5 \* 1835 \* 9.81 \* (1.0 / 2.56) \* (-6.26 / 9.81);
* Frear = 2346.77 N;

As shown by the calculation, during braking or deceleration the front wheels now hold a greater down force than the rear wheels.

**2.4.8 Cornering**

For a car to turn round a corner, the car must apply a lateral force and steering angle, which will determine the cars angular acceleration and angular velocity. Marco Monster [Monster] states “*This force depends on the slip angle (alpha), which is the angle between the tire’s heading and its direction of travel*” which indicates how the lateral force can be calculated when the steering angle is changed.

There are three factors that determine the slip angle, which are the “*the sideslip angle of the car, the angular rotation of the car around the up axis (yaw rate) and, for the front wheels, the steering angle*” as stated by Marco Monster [Monster]. The formula to calculate the sideslip angle is shown below.

* β = arctan(Vy / Vx);

Where Vy and Vx is the velocity vector

The angular rotation of the car is calculated by using the formula below. The rotation can be different for the front and rear wheel, but it depends where the centre of mass point on the car. Yaw speed uses the angular velocity, which is shown later on.

* yawspeedRear = aV \* l \* a1 / l;
* yawspeedFront = aV \* l \* a2 / l;
* rA = arctan(Vy / yawspeed);

The steering angle is just the angle of the wheel, and this only applies to the front wheels, as they are the only wheels on the car that changes its orientation and direction.

Once the three variables are determined, the next stage is to combine them to calculate the slip angle. The final equation for the slip angle is shown below.

Once the slip angle is determined the lateral forces can be calculated by using the formula stated below.

* Flateral = Fnlat \* Fz;

Where Fnlat is the normal lateral force and Fz is the load on the tyres, which is the weight of the car, multiplied by the weight track distribution.

The angular acceleration is calculated by calculating the body torque first, which is determined by the lateral forces. The formula is shown below.

* Tb = b \* Ffront – c \* Frear;

Where b and c is the wheelbase weight distribution ratios

Then using Newton’s [Newton] second law F = ma, the angular acceleration can be calculated. Using the same concept from previous calculations, the angular velocity, is the determined by incrementing the angular acceleration over time.

**2.4.9 Conclusion**

Overall the research has shown how a car can react with certain forces and dynamics acting on it, and these forces can all be calculated using complex formulas and equations, so therefore this will help the process of developing the application, as these formulas will be implemented. The research helped test some of the equations and theories behind the formulas, to ensure the results were accurate enough to be included into the implementation of the application, so that the formulas used can demonstrate the applications realism for racing simulators.

**3. Analysis & Design**

**3.1 Analysis**

**3.1.1 Introduction**

The research discovered in the previous chapters can help plan and design the implementation and process of the project. The project is focusing on developing an application that simulates real world physics in a racing game, from the different results found can help distinguish the different approaches that can be taken to achieve the final goals and objectives. This chapter is to discuss how the results helped in different ways, and analyze the different results found to form different approaches to be taken to target the end objective.

**3.1.2 Racing Simulators**

From the results found from what people actually preferred in terms of realism, the majority of people do prefer unrealistic racing simulators, but there is a market for realistic racing games, and that’s the market the project is targeting to demonstrate realism in racing simulators.

The results showed what people actually wanted in a realistic simulator and obviously real world physics is a major feature in a realistic racing game and is what people demanded, therefore the project will focus on the physics engine and an implementation of a physics library so the physics engine can employ it to perform complex calculations.

Since the focus is on the physics of the vehicle, the user interface will be a simple design to demonstrate the physics of vehicle dynamics applied to the vehicle’s 3D model, so using a simple graphics API will be the best approach to take to develop and implement this application.

People demand realistic racing games to be similar to real life racing and wants the experience to be as real as possible, so therefore the application will implement the use of a steering wheel as input to control the vehicle. The steering wheel will need a form of input controls and state management to help control what the user has inputted into the application, so the application knows how to process and execute the commands that needs to be performed when a certain state is changed.

**3.1.3 Software Methodologies**

The research for different software methodologies used in industry today showed how this project can adapt to these methodologies to assist the process and development of the project. The research showed how the majority of the methodologies used in industry today are group and team methodologies, but this project is developed by an individual, therefore none of the present methodologies were suitable, but were adaptable, so that the project can manipulate some of the practices to adapt for a single person project.

However the research showed that some researchers have implemented a new methodology known as PXP, as explained in the research chapter. The methodology was an implementation of XP, but for a single person, which is perfect for this project, as it will be an iterative and an individual team methodology. The project will use this methodology to assist in the planning of the projects implementation and testing by following the methodologies practices.

**3.1.4 Platforms**

There are many different platforms and frameworks used in the computing industry, from the research the current games industry shows a popularity of using C++ as its main coding language and uses either DirectX or OpenGL as its graphics API. The research indicated how complex C++ can be and the use of DirectX or OpenGL is a major learning curve, but C# was the other option, which indicated a simpler approach, but isn’t what the industry is looking for.

Due to the time scales of the project, the best approach would be the use of C# and the XNA framework, as the research indicated how C# is growing within the industry. Learning the XNA framework is simpler than DirectX or OpenGL, as it’s a managed graphics API and provides a simple interface to interact with the XNA library, also previous use of the XNA framework will help the implementation of the project.

From other researches, they indicated the purpose of the project, as its main focus is the physics engine, the models and game designs are not the main focus, so a simple graphics API to import different models would be the best for this project, so the time to develop is not taken away to implement and design the graphics. XNA is also capable to read different input using the DirectX input library to read the input for a generic gamepad, therefore the steering wheel can be used with the XNA framework.

**3.1.5 Vehicle Dynamics**

Vehicle dynamics is one of the core sections of this project, as it is the focus of using the formulas and physics from the research to implement the equations into a physics engine for a car racing simulator. The project is to demonstrate at realism in racing games, and to make the game realistic, is to use real world physics that affects the cars momentum and drive line.

This research was executed to look into the formulas and equations from vehicle dynamic concepts, which can assist the implementation of real world physics into a racing simulator. The research shown positive results, as the equations found had shown a significant amount of detail on how they are used, therefore the formulas were simple enough to test. The majority of the equations showed how one result from a calculation was linked to another to produce the next result, this indicates how the physics engine will need to be able to process these links and calculate new results as linked results change during runtime.

The research showed some of the formulas being compared to similar formulas which gave the same output, but the execution of the equations sometimes resulted in different values being outputted. Therefore the best formulas need to be chosen that best suits this project, some of the equations gave logical reason why one equation is chosen over the other, and is obviously best used. The key point to this project is to make the game as realistic as possible, so the best equations are the dynamic ones which are linked by other equations, so if one value alters, many equations will execute, due to one value, this shows a dynamic change in the physics, therefore more realistic affects.

The physics engine will have to include as many equations that can help increase the realism within this application, as the research showed how the calculations worked, so the implementation of the formulas should be fairly simple to develop, but a clean implementation would help keep the process of calculations low, so that the game is optimized to run at is best performance. By using simple test plans will help test these equations within the application and also help demonstrate the applications performance levels.

**3.1.6 Conclusion**

The analysis has shown how relevant the material that has been researched is and how it can be adapted to the project. From researching data about the market, has helped understand the reasons for the demand of realistic simulators and why this project can demonstrate why realism in games is important. The methodologies help plan and structure the flow of the project and to ensure that the project is following steps to guide the project to success. The platforms comparisons help distinguish which platform is best used for this project, as technology grows the amount of platforms and frameworks increases, so choosing which one that best suits this project and is relevant to industry demands is important. Finally the vehicle dynamics and physics is important to understand real world physics, so physics researched can be adapted and implemented into the application. By using real world calculations would allow the application to demonstrate realism in games. Overall the project analysis has shown the importance of the individual research topics and how they will be adapted and developed into the project.

**3.2 Design**

**3.2.1 Introduction**

The design is used to distinguish the implementation of the application from the analysis, it assists in the planning of the project, so the implementation process can determine the different components needed in the application and organize them in terms of priority and importance.

Design patterns and object oriented concepts can be used to help design the architecture of the application. The use of UML diagrams can be helpful to show potential architecture designs before implementing the application, so the developer understands how classes and objects are linked, which will therefore make the implementation and testing simpler.

This chapter is to look at some of the design patterns and best practice coding techniques that can be used in terms of a game engine and a physics library. By using UML diagrams will assist in demonstrating how the architecture can be implemented for the game engine and physics library.

**3.2.2 Game Engines**

Game engines are always being developed or used in games development and they always follow certain rules and practices to develop and implement a game engine. Game engines were designed to help solve many problems in game development, and one of the major problems is computational cost, as scenes need to be updated every frame.

The whole game engine uses simple object oriented (OO) concepts to implement, as keeping objects and classes separated can help computational costs, therefore only objects that are needed to be created at initialization are created and other objects are created during runtime, therefore object and class references are not kept in the heap wasting memory space.

The game engine should consist of a game loop to calculate variables every frame, so the game can update its values and render the games interface. The engine should consist of two main functions within the game loop, which are Update which calculates the algorithms and behaviours of the game object and Render which uses the variables after the update pass, to draw objects onto the screen, dealing with graphic algorithms within the content pipeline.

The use of Game Objects will help manage what is being created and what needs rendering, so the game object will have its own Update and Render function, so the game object manages its own algorithms and drawings. The game objects are then stored in a list so the game manager can access the different game objects and check for state changes.

Other designs like Input, Sound, Resource managers can help keep the game clean and easy to develop, as the different classes will manage its own behaviours, so for example a Input manager will manage and detect different inputs from a user or player and will then trigger events to let the game manager deal with the action and behaviours that needs to take place.

The game engine will be designed using the game engine concepts and practices as discussed, this will allow the development to be controlled and planned into different components that are needed in this application, which therefore allows easy debugging and refactoring during these stages within the methodology.

**3.2.3 Object Oriented Concepts**

Object Oriented (OO) programming is a set of fundamentals and practices of concepts used in the development industry. The concepts assist the organization and planning of a development team and the projects implementation and testing phase. Using objects and classes can help the debugging and refactoring phase as it is easier to find bugs within independent objects. This section is to discuss how this application will be designed using some of the object oriented concepts.

*Inheritance*

Inheritance is used for specializing objects and classes with subclasses; this will allow subclasses to hold more information about the object, but will still be the same type as the base class therefore other classes with instances of the object can still access its base class functionality.

For this project the games engine will use a base class for its game objects called GameObject and objects like CarObject will be a subclass of a GameObject, so the CarObject can access the core information and functionality using the super class functionality and store extra information for the CarObject that is not necessarily needed in a GameObject. The GameObject class should be a clean and basic object where common attributes and variables are common in all the GameObjects needed for the application, for example, a Texture or a Model, or Positions and Centres Positions are all common attributes a CarObject or a TreeObject may have.

*Encapsulation*

Encapsulation is used to control how the class allows objects access its core functionality, this is created by defining the members of the object, which are specified by either public, private or protected. Encapsulation can also be used to hide public access to variables, so other objects can access them by using getter and setter functions.

This application can use this concept to hide functionality from the manager classes, for example the physics manager can only access commands and functions that use algorithms to calculate values and variables for the game object. The GameObject only accesses the physics manager to perform these calculations to return values, but has no access to how the calculations are made, which are hidden in interfaces and command classes.

Using protected members within GameObject’s will only allow subclasses of a GameObject to access them member functions, so for example a GameObject can hold a base functionality to calculate its centre so a CarObject can access this functionality within the object when it needs to calculate its centre, but other classes cannot access this functionality.

*Polymorphism*

Polymorphism is used to make objects data type appear as its subclass data type, this allows a generic way of developing an application, therefore the objects can be of data type A and can still access its functionality from its subclass B.

In the game manager’s update and render, the manager doesn’t want to be checking what the subtype of the object is, so the list can store only GameObject data types, therefore the list allows any subtypes of GameObject to be added, so the CarObject and TreeObject can be added into the list but will have a data type GameObject. When the Update or Render in the game manager calls the GameObject functions, it will use the subclass functionality even if the data type is a GameObject, this can only work if the subclass override’s its base functionality or it will just call the base class functionality as default.

**3.2.4 Object Oriented Techniques**

There are many different object oriented techniques that can be used to help optimize the applications implementation and also ensure the object oriented concepts are adapted, therefore the application will become dynamic and efficient.

*Events*

Events can be used as a message system for the game input class, as keys are being pressed the class can detect when keys are pressed and fire an event so the main game class can handle the event with its own functions. This functionality helps create message systems between objects and classes, as event are being triggered, the relevant object or class will pick up the message and execute the functionality.

*Abstract*

Abstract classes are used similar to a base class with inheritance, but an abstract class doesn’t allow instances of itself to be created. The GameObject or CameraObject can be an abstract class, as a simple GameObject or CameraObject, doesn’t really need to be instantiated, as they hold very little information. Its class that inherits the GameObject and CameraObject that needs to be instantiated. Also abstract classes also give the option of using abstract functions, which states that any functions that are abstract does not hold any functionality within its bas class, classes that inherit from the base class have to implement the function and give it relevant functionality.

*Interface*

Interfaces are similar to abstract classes, but are stricter, as they only hold signatures of method calls in the interface, so if a class implements the interface the class also needs to implement the functionality for functions that need implementing. This project can use interfaces to help hold type objects for the physics library. The physics library is created by using a set list of command classes that execute behaviour and algorithms. The command classes can implement an IExiaCommand class, where there is a execute() function that needs implementing in the individual command classes. This will help keep the command classes separate and also minimize the amount of calculations performed at once, and only the commands that are needed are executed

**3.2.5 Design Patterns**

Design patterns are important to use when implementing a system, as they help optimize the applications performance. Design patterns are a set list of solutions to implement and solve common problems within the programming. The design patterns help determine how code is implemented and design, which gives the developers an easy way of looking on how to plan their implementation.

This project can use some of the design patterns to help implement and design the application, as the Singleton design pattern would be useful to classes where they can only be instantiated once, as some of the game core classes don’t need multiple instances, for example the GameInput, GameSound or the ResourceManager class need more than one, this therefore helps manage what is being created and efficiently saving memory space and improving performance.

Other design patterns like the Flyweight can be used to help manage the Resources being loaded into the game, as the design pattern helps store the resources only in one repository so multiple loads of the content is not needed. The resources can be accessed when the application needs it.

The physics engine can also use the Strategy pattern, where the commands can be treated as a behaviour or a strategy, and it only calls the command it needs when the strategy is being executed. If the physics manager needs to change its behaviour, it can change it during runtime, this behaviour is similar to the process of the State pattern, as state changes behaviours and algorithms for the current selected state is executed, so for the current selected command the physic algorithm can be execute.

**3.2.6 UML Diagrams**

UML diagrams are used to help design the architecture of the application before starting the code implementations. The diagrams help determine the plan and strategy to tackle the problems and issues that may arise during the implementation. There is much different type of diagrams that can help this project, but the main ones to focus on are the Class and Sequence diagram.

The class diagram shows how the architecture of the application is to be implemented, with the use of objects and classes. The diagram shows the relationships between the classes and how they are linked using object oriented concepts and techniques. The physics library shows a physics manager that holds and stores command objects, which are implementing the IExiaCommand interface. These objects are executed in the physics manager update function, and the car object stores an instance of the physics manger and calls its update function.

The sequence diagram shows how objects are created and how functionality is executed within the threads time line, the main focus is the sequence of the physics library, as the core order of calculations, needs to be designed before the implementation of mathematical calculations occur, as wrong calculations, or values not being calculated first can cause the application to crash. The sequence and class diagram are shown in Appendix E.

**3.2.7 Software Methodology**

The first phase of the PXP methodology is to design a plan which contains a list of features ordered by priority. The features are based on user stories and requirements of the project. The design of the plan is shown in Appendix D, where this projects requirement are listed into user stories and broken down into feature lists, where the features are then placed into phases and stages. Each of the phases is sorted in order by priority and importance. The phases are designed as a development stage, so features within the current phase is implemented and tested before moving onto the next phase. The phases will have iterations if problems are encountered or if new features within the phase are required.

**3.2.8 Conclusion**

The design and planning has shown how the project will be developed and implemented. As the methodology describes each of the phases of implementation and the UML designs indicate the architecture and structure of the application. The best practices and fundamental concepts of development and programming are taken into account, to help design the best architecture that suits this project. The overall design has shown new concepts and new ways of development that will be adapted and implemented into this project.

**4. Implementation & Testing**

**4.1 Implementation**

**4.1.1 Introduction**

The implementation section will indicate how the project is implemented by following the chosen methodology, and how the each of the sections discussed in the previous chapters are used and adapted to implement the artefact. Problems encountered and solved will also be discussed within this chapter as tests were executed as features of the artefact were developed.

The report on the implementation will be based on iterations of phases and stages within the development plan, each phase has a set of features and is used to show what needs implemented and tested within that phase. The phases are shown in the Appendix D with its list of features.

As discussed in the research section, the language and API used to implement this artefact will be based on using C# and the XNA library, therefore Visual Studios 2008 will be used to develop this artefact.

**4.1.2 Phase One - Models**

This is a simple phase, where the use of 3D models, designed and created in 3D Studio Max, are imported into the XNA framework to test which models are best suited for the project and also shows enough detail to demonstrate the main artefacts features.

XNA supports DirectX (.x) and FBX (.fbx) files as 3D models, so therefore any model found will need to be in this format. The models are available from some websites where designers publish their models for other people to use; some of the known sites are [TurboSquid] and [the3DStudio]. After retrieving a few models to test and work with, the next stage is to create a simple program using the XNA library to import these models.

XNA models are loaded by using the Content manager that the XNA framework provides. Each model loaded is a type object of Model, so instances of Model are created to store the 3D models information. The line below shows how the use of Content is used to load a Model object type, “SKYLINE” is the first model to be loaded and viewed.

Model carModel = Content.Load<Model>(“SKYLINE”);

The rendering of the model is done by using a basic effect that XNA provide to define the models world, view and projection matrixes. This is all shown on the MSDN library, [MSDN].

There are three models to be tested and viewed, to see which one is best for this project and the results of this is shown in Table 7.

|  |  |  |
| --- | --- | --- |
| Name | Description | Image |
| Skyline | Model loaded fine, but small errors on model has caused the model to lose some features of colour and vertices |  |
| SL500 | Model rendered with detailed texture materials, high detailed features on the interior of the car, very small model. |  |
| BMW330 | Model rendered with high detail, and a very large size model, but no details of textures included on the model. |  |

**Table 7 Results of the loading and rendering of models**

*Phase Conclusion*

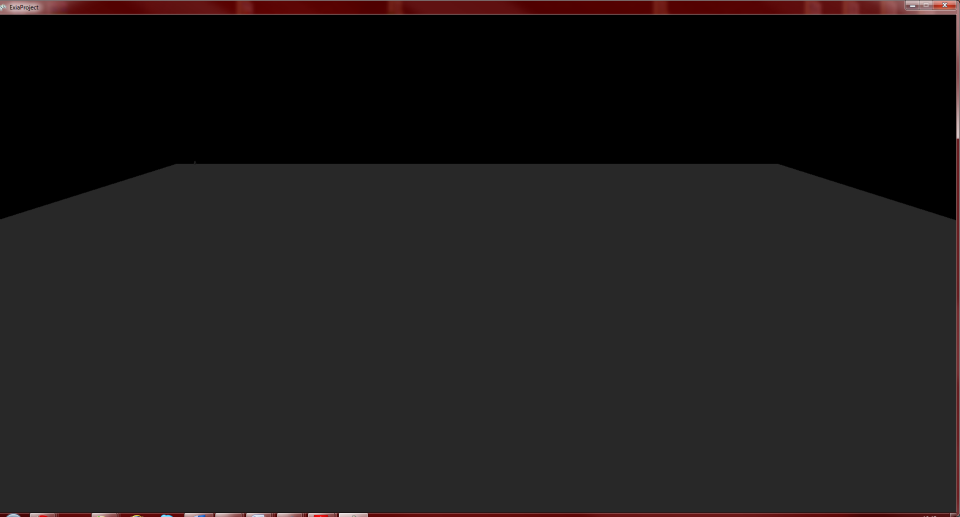
The results of the model loaded and tested indicates the SL500 to be the best suited model for this project, as the size is small, which therefore lowers processing time during the render phase of the application. The model also includes texture images to map on to the model, to give the model advance details and therefore makes the artefact look and feel more realistic.

**4.1.3 Phase Two - Graphics**

This phase of the implementation is to look at how to create and generate the graphical interface of the artefact. The features included in this phase are terrain generation, sky box, and camera orientation. Each of the features is important to demonstrate the artefact physic dynamic features and give the users a sense of realism. This phase will be a basic creation of standard graphics, to allow the features to be created as a base and then re-factored into advance graphics later on.

*Terrain*

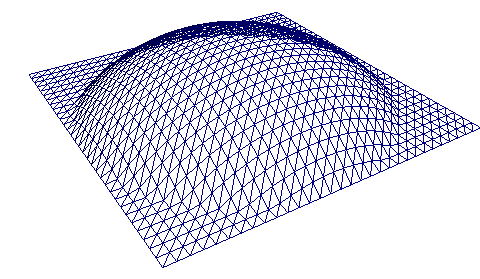
The terrain is a basic flat pane in this phase, this is going to give the car model a terrain to drive and move on. The implementation of this terrain uses a tutorial from a website by Riemer [Riemer2008]. Riemer uses basic indices and vertices logic to generate the terrain, so using Vectors, the points can be stored into an array buffer, but for a flat terrain the Y value for the Vector will be set to 0 at all times. The colour of the terrain will be set to a grey colour which represents the road. After following the tutorial the terrain was created and generated as a flat pane, as shown in Figure 4.



**Figure 4 Flat Pane Terrain**

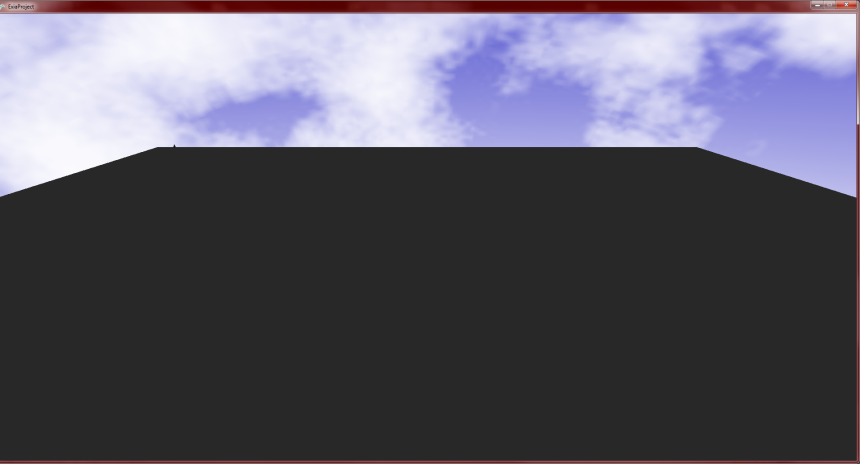
*Sky Dome*

The sky dome is also created using one of Riemer [Riemer2008] tutorial. The dome is a DirectX (.x) 3D Model, and this is imported to create the sky dome for this application.



**Figure 5 dome.x model for the Sky Dome [Riemer2010]**

The dome also needs to be textured with a cloud map, and this is done by using effects in XNA, where XNA can create a basic effect for models, but for the realism of the sky dome, a simple custom effect file is used to help map the texture on to the dome, this effect file is a custom made file by Riemer [Riemer2008]. For realism the sky domes position is always relative to the camera’s position, this gives the graphical interface a sense of depth. After completing the tutorial from Riemer the application now has a sky dome with a textured cloud map, as shown in Figure 6.



**Figure 6 Application with the texture Sky Dome and Terrain**

*Camera*

The camera is created and place at certain positions, which allows the scene to be rendered within the camera’s view and projection. Currently the camera is placed high up in the sky which shows the terrain and the sky dome. The camera should be placed behind the car at all times and follows the cars, like a chase camera, also other views should be available for the user to select, like the in car view.

Placing the camera behind the car is simple, as an offset will be applied to the camera from the cars current position, and by using the cars position and applying the offset the camera will follow the car like a chase camera.

\_cameraPosition = \_offset;

\_cameraPosition += \_CarPosition;

The code shows, \_offset as the initial offset position of the camera, and is then applied to \_CarPosition, which is the position of the car, to give the final position of the camera behind the car.

Also the orientation is important so when the car turns left or right the car object will rotate, so therefore the camera must also rotate. By having an orientation value that indicates the cars rotation, the same value can be used to calculate the cameras rotation and position. The rotation is based on a Quaternion matrix, which helps calculate the values of rotation and world matrixes easier.

\_cameraPosition = Vector3.Transform(\_cameraPosition,

Matrix.CreateFromQuaternion(\_cameraRotation));

The code above indicates how the position of the camera is transformed relative to the rotation of the car, where \_cameraRotation is a Quaternion matrix and is created by the cars rotation value.

Overall the cameras position is given an offset, then a transformation is applied to give the rotation of the camera, and finally the position of the car is applied to this value to give its final position of the camera. Figure 7 shows the camera behind the car.



**Figure 7 The rear view Chase Camera, relative to the cars position and orientation**

The offset should be a designed variable that the user or developer can change, so therefore a camera for the inside view can be created, and also be treated as a chase camera. Therefore by applying a different offset, the cameras position can be placed within the car’s interior on the driver’s side, which should therefore give the player or user a good sense of realism. This view is shown in Figure 8.



**Figure 8 Inside car view for realism**

*Phase Conclusion*

This phase has shown how the graphical interface is implemented and has given the application the graphical looks it needs to show and demonstrate realistic vehicle dynamics in a racing game. The application currently creates the objects, like the terrain, sky dome, car and the cameras within the Game1.cs file, and runs within the relevant game states, this will need further work to re-factor the objects into different classes, which shall be implemented at a later phase.

**4.1.4 Phase Three - Advance Graphics**

This phase looks into implementing complex graphics. The main features for this phase are height map generation, so the terrain can have a sense of depth to give the application a 3D feature. Also camera delays are introduced in this phase to give the camera a realistic view when driving the car on the terrain map and finally basic collision detection is introduced so the car object does not go through the terrain walls.

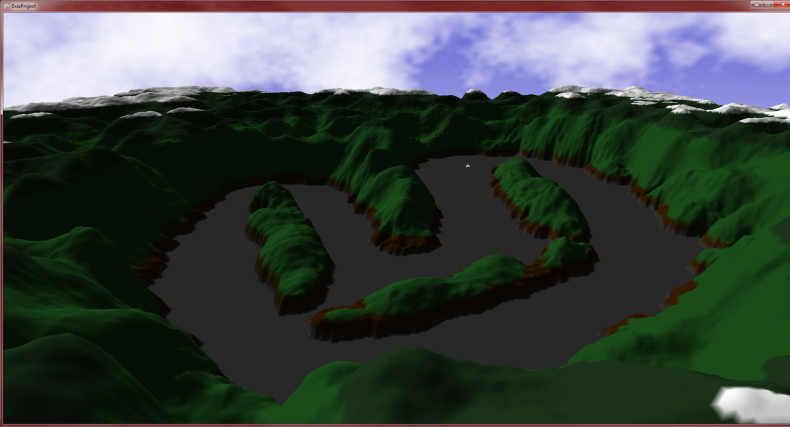
*Height Map*

The height map is generated by altering and giving the terrain, indices and vertices on the Y axis, which therefore gives the terrain some height. For a smooth terrain, the values of Y are generated by applying a height map to the terrain, which tends to be a black and white image, where the colours and shades on the image determine how high the points should be. Figure 9 shows an example of a height map that will be used in this artefact, the dark shades show low height and the light shades show high height values.



**Figure 9 Height Map of a small race track**

The application loads the height map and does a pixel check, to test the colour and shade, and then works out a value for the Y axis. Once the height is determined the terrain is then coloured dependent on what the height of the points are, so anything that is zero is given a dark grey colour, anything slightly higher than ground level is brown, then higher points are set to green and dark green, and then points towards the highest peak are set to white. Using normalization and determining vertices and faces normal vectors, the terrain can use light sources to determine shades, to give the terrain a dynamic effect.



**Figure 10 Outcome of applying a height map to the application**

These techniques and effects of creating and using height maps were shown from a tutorial by Riemer [Riemer2008]. Figure 10 shows what the outcome of applying a height map to the application.

*Camera Delays*

Camera delays is a common effect on a chase camera within games, it provides a sense of realism and also gives the game a more dynamic view. This effect is created by using the calculations previously for the cameras position and rotation, and applies an interpolation technique to the values which therefore adds a delay on how the camera changes it position and rotation.

\_cameraRotation = Quaternion.Lerp(\_cameraRotation,

\_followObject.CarRotation, 0.2f);

The \_cameraRotation is not just set to the car objects rotation this time, but uses the Quaternion.Lerp() function to interpolate through the car rotation points. The 0.2f is a value that sets the steps of interpolation through the rotation points; this value is between 0 and 1 where 1 is the maximum rotation point of the car, therefore for every 0.2 step between 0 and 1, the camera rotation changes.



**Figure 11 Camera delay when the car changes its orientation**

*Collision Detection*

The collision detection is implemented to stop the car object from going through the terrains walls. This is implemented by checking the objects position against the points on the height map where the value of Y is greater than 0, so therefore when the car object position intercepts the terrains height positions.

GetHeight(\_carPosition) > 0.0f

*Phase Conclusion*

Overall this phase has shown a major improvement to the graphical interface which creates a realistic game, and therefore provides a good demonstration to the physics of the vehicle when driving round a track. Re-factoring is still needed to clean the code of the project, so dynamic classes can be created for specific objects.

**4.1.5 Phase Four - Refactoring**

This phase is where all the current code and objects written to create the application is re-factored into its own class and dynamic object, where instances of the class are created when it is needed. The classes will be created based on the design of the applications game engine. The class information is all shown in the design section of this project and in the UML diagrams in Appendix E.

The main features of this phase are to re-factor the graphical features. The terrain generation will be created and initialised in a Terrain object, a SkyDome class will be created to deal with information of the sky dome, the Camera will be changed to a base class where other cameras, like Chase and Stationary inherit from the base Camera class, and finally the CarObject will be created to deal with all the information the car model may hold.

The objects will be created in the Game class and are used within the process stages of the XNA framework. The Game class will deal with the standard load, update and render stages and use the objects that are relevant to them stages. The Game class will also deal with the content and resources loading, so the resources can then be given to the relevant objects.

*Terrain Class*

The terrain class will contain all the processes and functions that are needed to generate and calculate all the indices and vertices needed to create the terrain. The class will hold its own vertex and index buffer to store this information. When the terrain class is created, parameters are passed into its constructor, which are Effect, Size and Heightmap. This information will be from the Game class.

ColoredTerrain(Effect effect, Vector2 size, Texture2D heightMap)

\_terrain = new ColoredTerrain(effect.Clone(device), new

Vector2(500, 500),

Content.Load<Texture2D>("TEXTURE//heightmap6"));

There are three core functions where the Game class will need to access it to perform the processes it needs. These functions are the Initialise, Update and Render, where initialise will begin the generation and information storing of the terrain data and height map data. Update processes any information that needs changing over time during the process of the Game, and finally render is to render the terrain on to the screen by using the buffer information.

The GetHeight() function will also be stored in the terrain class, as the function needs to know the information from the terrains height buffer, so this function is static so that other classes or objects can access this function without creating a instance of this class.

*SkyDome Class*

The SkyDome class is implemented to create the sky dome for the scenery in the application. The core information this class stores is the Model of the sky dome and a Texture2D of the texture image it needs to map onto the dome model. The effects are also needed for the model to understand how to map a texture to its model with the specified image. The constructor of the class is shown below.

SkyDome(Model skyModel, Texture2D skyMap, Effect effect)

\_skyDome = new SkyDome(

Content.Load<Model>("MODEL//DOME//dome"),

Content.Load<Texture2D>("MODEL//DOME//cloud"),

effect.Clone(device));

The class functionality is very basic, as the update and initialisation of this class is not needed; only the render function is core to this class, as it needs to be able to draw itself on to the screen.

*Camera Class*

The camera class is an abstract base class, with function signatures where classes that inherit the camera class need to implement these functions. The camera class cannot be instantiated, so classes need to be derived from it to be created. This class is used to store the information about the camera’s view and projection, and cameras can vary their properties to alter its view.

The base camera has a constructor that needs a set of parameters for the class to store; these parameters are the base position, the viewport and the rotation angle. These parameters help determine the cameras orientation and positional offsets.

Camera(Vector3 position, Viewport viewport, Vector2 rot)

There are two main cameras types that are needed within this project, and they are ChaseCamera and StationaryCamera, where the chase camera follows a specified object and in this case the car, the stationary camera is a placed camera which does not move from its position.

The chase and stationary camera extends the Camera base class and uses the base constructor to pass the parameters the camera constructor needs. The chase camera also needs its own parameters and properties to store in its class.

ChaseCamera(CarObject follow, Vector3 position, Viewport

viewport, Vector2 rotAngles, bool realView)

: base(position, viewport, rotAngles)

The follow object is a reference to a car object where the camera will follow and a realView is the type of view the chase camera is, either a realistic view or standard view, this is triggered by a Boolean either true or false.

The stationary camera does not need any more information that it needs to store as the camera doesn’t need to know about other information in the application as the camera just stays stationary at one point in the world.

*Car Object*

The car object is an important object within this project, as it’s the core class that creates the model of the car and generates all the physics it needs to allow the car to move. The object has three main functions, initialise, update and render, similar to the terrain and sky dome classes.

The update function needs a GameTime parameter so that the update can process its functionality at a given time, and the game time can be used to calculate the time it took from one call to the next call.

update(GameTime gameTime)

The render function needs a GraphicsDevice and a ViewTransformation parameter so the render function can draw the model of the car using the device given, and to position it relative to the camera the view information is needed.

render(GraphicsDevice device, ViewTransformations viewTrans)

The car object class has a constructor where it needs a Model, Specification and Steering Wheel Input parameters. The model gives the object what to render on to the screen, the specification is used to determine the facts and figures of a specific car, and then this information can be used in the Exia library to perform calculations. The steering wheel input, is the input device the car object will need to use to change values of the cars orientation, velocity and acceleration.

CarObject( Model carModel, Specification carSpec,

SWInput swInput)

This class is a derived class of PhysicsObject, which is a base class that holds variables and properties for the physics calculations; this base class is part of the Exia library. The base PhysicsObject class does not have a constructor or any functionality, the class is just a data holding class and also gives the CarObject a PhysicsObject type reference, so the Exia library can still use the information it needs, but doesn’t specifically need a Car type object.

*Phase Conclusion*

This phase has shown how code can be extracted from one class and be made into separate classes, so therefore the object has its own independence and can be dynamically created when it is needed. The main game class now creates the instances of a Terrain, Skybox, Camera’s and the Car objects, and uses the game process stages to call individual functionality from these classes.

**4.1.6 Phase Five - Exia Engine**

This phase is the core implementation of this project as this is where the physics library is developed for the calculations to be executed and managed in the application. This phase has many features to it, these features are the core commands for the calculations, the base PhysicsObject class to store the information, the PhysicsManager which creates the communication with the CarObject to the commands, and other utility classes to store physic constants and unit convertor functionality.

The Exia is the physics library that will be used by the application to apply vehicle dynamics to the car. The library will use the formulas and equations, which are shown in the research section, to perform these calculations for the car object. The calculations will determine the car objects position and orientation.

*Engine Core & Utils*

The core parts for the engine are the PhysicsManager, PhysicsObject, and Specification classes, and also the IExiaCommand interface. Each of these features is used to help either store information or assist in the communication between two classes.

The IExiaCommand is an interface with one signature method called execute. The interface is implemented by the command classes so the calculation can be performed within the execute function when called. The function also takes a PhysicsObject parameter, which assists the commands with the calculation.

execute(PhysicsObject pObj);

The physics manager is used to create the communication from the CarObject to the physic commands. The physics manager has a simple interface to it, as it has one main function called update where GameTime and PhysicsObject is passed into the function as parameters.

update(GameTime gameTime, PhysicsObject pObj)

As discussed in the previous phase the PhysicsObject is a data class that holds a large amount of variables, for example position, rotation and velocity. These variables are used to help perform the calculations it needs to apply the realistic physics to the car object. The physics manager and the commands take this object through method parameters and sets variables and data to values that are calculated within the commands, as instances of an object is treated as a reference the CarObject will pick up the changes to the variables.

The PhysicsManager holds references to all the command classes that implement the IExiaCommand interface, and within the update function these commands are executed to perform their calculations. The CarObject holds a reference of the PhysicsManager and calls the update function within its own update function, and because car is derived from a PhysicsObject the CarObject jus passes itself to the PhysicsManager.

\_physicsManager.update(gameTime, this);

The Specification class is implemented to hold constant information about a certain car, for example the cars wheel base or the cars mass. This object is treated like a data object, which stores a large amount of functions, but also has small functions to assist how the information is stored. The PhysicsObject has a reference to a Specification object so then the commands can access the specification data to execute its calculations.

Other classes that are in the Exia library are the utility classes that help execute calculations. The PhysicsConstant class is implemented to store static variables of values used around the world today within calculations, for example gravity, g = 9.81, so this value is stored in this class. Since the variables are static any other object or class can access the variable without creating a instance of the class.

PhysicsConstants.g;

The other utility class used is a UnitConverter class, where values can be passed into this class and can change from one unit to the next for example meters per second (ms) to miles per hour (mph), which is a measurement of speed and velocity. Again these unit conversions are static ways of converting from one unit to the next which are well known around the world. Also this is a static class where all the functions are accessible without creating an instance of the class.

*Acceleration, Velocity and Displacement*

The command classes for Acceleration, Velocity and Displacement are fairly simple, as its basic physics. From the research the Acceleration is based on force against mass, so the mass of the car is known from the specification, which is 1835 kg, and the force is calculated, but this is shown in a later section, but for testing purposes, the value for force was given 9000 N which gives a constant acceleration.

* a = F/M
* a = 9000 N / 1835 kg = 4.9 ms²

The acceleration and force variables were given a Vector value, as the car needs to be able to deal with the displacement change in the Z and X direction, where Z axis is based on forwards and backwards direction and X axis is based on sliding left or right.

pObj.\_force = new Vector3(0,0,9000)

public void execute(PhysicsObject pObj)

{

pObj.\_acceleration.Z = pObj.\_force.Z / pObj.\_spec.CurbWeight;

pObj.\_acceleration.X = pObj.\_force.X / pObj.\_spec.CurbWeight;

}

From working out the acceleration the velocity can be calculated, as velocity is based on integrating the acceleration over time, which is determined in the research. To work out the increment of time in seconds from the XNA frameworks process in the update stage, is based on the GameTime parameter on the update process function in the CarObject. As shown below \_delta is the incremented time, and by using the gameTime parameter the update process knows the time it took from one call of the update method to the next.

\_delta = (float)gameTime.ElapsedGameTime.TotalSeconds;

Once the incremented time is worked out, using the equation shown below can determine the current velocity of the car. Where a = 4.9 ms², dt is set to 0.5 and v is initialised at 0 for testing purposes, the velocity can be determined.

* v = v + dt \* a
* v = 0 + 0.5 \* 4.9 ms² = 2.45 ms

The velocity is also a vector value, so the velocity can be determined within the Z and X axis of the car, which will be useful when the Lateral Forces are taken into account.

public void execute(PhysicsObject pObj)

{

pObj.\_velocity += pObj.\_delta \* pObj.\_acceleration;

}

Similar to calculation the velocity of the car, the displacement can also be determined by integrating the velocity over time. The same time calculation for dt is used, and replacing the acceleration with the velocity in the calculation can determine the displacement position of the car.

* p = p + dt \* v
* p = 0 + 0.5 \* 2.45 ms = 1.225 m

The implementation of this function is slightly different compared to the Acceleration and Velocity, the orientation of the car has to be taken into account to calculate the direction vector the car is going. A simple matrix multiplication for the velocity and rotation is needed to determine the directional velocity as shown below.

public void execute(PhysicsObject pObj)

{

Vector3 v = pObj.\_velocity;

Quaternion r = pObj.\_rotation;

pObj.\_speed = Vector3.Transform(-v, r);

pObj.\_position += pObj.\_delta \* pObj.\_speed;

}

The negative v sets the direction of the car to forward, so going into the board, this is due to the notation used in the XNA framework.

*RPM & Torque*

The next command class that needs implementing is the RPM and Torque class, as the Forces acting on the car is dependent on these two calculations so that the force can be integrated into the application to calculate the forward acceleration. The formula for RPM is shown below, as discovered from the research.

* rpm = wR \* gR \* fR \* 60 / 2pi

The gear ratio and final drive ratio are from the Specification class, where the PhysicsObject will store a current gear, so that the gear ratio can be looked up from the Specification class. The gear ratio of the car during first gear is at 3.59 and the final drive ratio of the car is 2.82. The wheel angular velocity is based on the cars current velocity and the cars wheel radius in mm, so for this car the wheel radius is 0.33 mm and the current car velocity is at 2.45 ms for testing purposes.

* wR = 2.45 ms / 0.33 mm = 7.42 rad/s
* rpm = 7.42 \* 3.59 \* 2.82 \* 60 / 2Pi = 717 rpm

Cars need to rotate an approximately 1000 rpm for the car to keep its engine going or else the car will end up stalling, so a condition statement is needed in this command to stop the car from going below 1000, as shown below.

public void execute(PhysicsObject pObj)

{

float rpm = 0;

rpm = ((pObj.\_velocity.Z/ (UnitConverter.CMtoMM(pObj.\_spec.WheelRadius)))

\* pObj.\_spec.Gears[pObj.\_currentGear]

\* pObj.\_spec.FinalDrive

\* 60 )/ MathHelper.TwoPi;

if (rpm < 1000)

rpm = 1000;

pObj.\_currentRPM = rpm;

}

The torque is a complex formula that uses a quadratic equation to solve its current torque at a specific gear as shown in the research. The basic torque formula is shown below, which calculates the drive torque. The values of the transmission efficiency, gear and final drive ratios are stored in the Specification object.

* Tdrive = Tengine \* gR \* fR \* tE

The engine torque is calculated using the torque performance equation which is shown in the research. The engine torque is based on the velocity of the engine, we. These equations are shown below.

* Tengine = (p1 + p2 \* we + p3 \* we²)
* we = (gR \* fR / wR) \* v;

From these equations, the formula can be combined to give the torque of the car at a specific gear as shown below. The torque is also based on user input, as the user may alter the amount of torque depending on the throttle position. The throttle position is between 0 and 1, so therefore when the user puts his foot down the throttle position is set to 1 so the torque is at its maximum point. This throttle position will determine if the car is moving or not.

public void execute(PhysicsObject pObj)

{

float p1 = UnitConverter.HPtoW(pObj.\_spec.MaxPower) /

UnitConverter.RPMtoRADs(pObj.\_spec.MaxPowerAtRPM);

float p2 = UnitConverter.HPtoW(pObj.\_spec.MaxPower) /

(float)Math.Pow((double)UnitConverter.RPMtoRADs(

pObj.\_spec.MaxPowerAtRPM), 2);

float p3 = -UnitConverter.HPtoW(pObj.\_spec.MaxPower) /

(float)Math.Pow((double)UnitConverter.RPMtoRADs(

pObj.\_spec.MaxPowerAtRPM), 3);

float we = 0;

we = ((pObj.\_spec.Gears[pObj.\_currentGear] \*

pObj.\_spec.FinalDrive) / (pObj.\_spec.WheelRadius/100))

\* pObj.\_velocity.Z;

float tE = p1 + (p2 \* we) + (p3 \*

(float)Math.Pow((double)we, 2));

pObj.\_we = we;

pObj.\_currentTorque = tE \* pObj.\_throttlePosition;

}

*Forces - Traction & Resistance*

This command determines the forces acting on the car in the forward and backwards direction, therefore this can determine the acceleration and the velocity of the car. From the test for the acceleration the force was set to 9000 N, this force provides the traction force the car needs to make it move forwards. The formula for the traction force is shown below.

* Ftraction = Tengine \* gR \* fR \* tE / wR

The Tengine was calculated in the Torque component, and the other information is from the car’s specification, therefore the traction force can be calculated as shown below.

public void execute(PhysicsObject pObj)

{

float tractionForce = 0;

tractionForce = (pObj.\_currentTorque \*

pObj.\_spec.Gears[pObj.\_currentGear] \*

pObj.\_spec.FinalDrive \*

pObj.\_spec.TransmissionEfficiency) /

pObj.\_spec.WheelRadius / 100);

pObj.\_tractionForce = tractionForce;

}

Traction force isn’t the only force that will be acting on the car, so the resistance forces are taken into account, which causes the car to slow down, and goes against the traction forces. The first resistance force is drag resistance which acts are air resistance.

* Fdrag = -Cdrag \* v²
* Cdrag = 0.5f \* dC \* fA \* rHo

Where the drag coefficient = 0.29 and frontal area = 2.0 are from the cars specification, and the rHo is the density of air, this is kept constant to standard dry air, which is at 1.29; therefore the value is stored in the PhysicsConstant class. This value can be changed depending on the environment and the weather.

* Cdrag = 0.5f \* 0.29 \* 2.0 \* 1.29 = 0.3741
* Fdrag = -0.3741 \* v²

The negative sign gives the drag force an opposing force to traction force.

public void execute(PhysicsObject pObj)

{

float Fdrag = 0;

float CDrag = 0;

CDrag = 0.5f \* pObj.\_spec.DragCoefficient \*

pObj.\_spec.FrontalArea \* PhysicsConstants.rho;

Fdrag = -CDrag \* pObj.\_velocity.Z \* pObj.\_velocity.Z;

pObj.\_cDrag = CDrag;

pObj.\_dragForce = Fdrag;

}

The other resistance force is the rolling force of the car, as this is created by friction and traction. The formula is shown below, which was explained in the research. The parameters are simple to find, as c is a constant coefficient, and for a standard concrete road the value is 0.015 and the W is the weight which is the mass of the car multiplied by gravity.

* Frolling = c \* W
* Frolling = 0.015 \* 9.81 \* 1835 = 270 N

This again is set to a negative value as it’s an opposing force to the traction force. Also as this can only affect the car when the car is moving forwards a check is added to ensure this only happens when the velocity in the Z direction is greater than 0.

public void execute(PhysicsObject pObj)

{

float rr = 0; float Frr = 0;

if(pObj.\_velocity.Z >0)rr = 1.0f;

Frr = -rr \* PhysicsConstants.rcoef \* (pObj.\_spec.CurbWeight)

\* PhysicsConstants.g;

pObj.\_rollingForce = Frr;

}

The brake force is similar to the rolling resistance, but this time the coefficient of friction is taken into account. The equation below shows how to calculate the braking force, the coefficient of friction for brakes is normally 0.62.

* Fbrake = u \* W
* Fbrake = 0.62 \* 9.81 \* 1835 = 11160 N

This force is a value that is determined by user input, as users apply the brakes in a car, so there must be a value to multiply by between 0 and 1, so 1 is for full brake force applied and 0 for no brakes applied.

public void execute(PhysicsObject pObj)

{

float Fb = 0;

if (pObj.\_velocity.Z > 0)

Fb = ((PhysicsConstants.cof \* PhysicsConstants.g \*

pObj.\_spec.CurbWeight))\*

-pObj.\_brakingPosition;

pObj.\_brakingForce = Fb;

}

The check is to stop the car from going backwards when a brake force is applied and the car is stationary. The brake force is also a negative force as it opposes the traction force to slow the car down.

Overall once all these forces are calculated the net force can be summed up by adding all the forces together, and since the values of the three resistance forces are negative they will be deducted from the traction force to act as a resistance.

public void execute(PhysicsObject pObj)

{

Vector3 netForce = new Vector3();

netForce.Z = pObj.\_tractionForce + pObj.\_brakingForce +

pObj.\_rollingForce + pObj.\_dragForce;

if (pObj.\_currentRPM > 6000)

{

if (netForce.Z > 0)

netForce.Z = 0;

}

pObj.\_force = netForce;

}

The force is a vector, so the Z value is affected for the forwards and backwards motion, later on the X axis will determine the lateral forces when the car turns left or right. Also the force is set to 0 when the current RPM count is over 6000 rpm, this will stop the car from passing the redline on the revs.

*Weight Transfer*

The weight transfer command is calculated to determine how the weight is transferred from the front tires to the rear tires; the use of this calculation will basically create the camera effects, so when braking the camera tilts forwards and during acceleration the camera tilts backwards, and the amount of tilt is affect by the amount of down force acting on the front and rear tires.

There are two formulas that are used to calculate the front wheels down force and the rear wheels down force, as exposed by the research. The formulas are shown below, where b and c are the wheel base distribution from the centre of mass, W is weight, h is the height from the centre of mass, l as wheel base, m as mass, and finally a as acceleration.

* Wrear = 0.5 \* b \* W + 0.5 \* h/l \* m \* a
* Wfront = 0.5 \* c \* W - 0.5 \* h/l \* m \* a

By applying this equation the weight distribution for the front and rear wheels can be determined, but when the acceleration is at 0 therefore the car is at constant speed the rear and front down force would depend on the values of b and c.

public void execute(PhysicsObject pObj)

{

float Wf = 0;

float Wr = 0;

float b = (pObj.\_spec.WeightDistributionRear / 100);

float c = (pObj.\_spec.WeightDistributionFront / 100);

Wf = (0.5f \* b \* pObj.\_spec.CurbWeight \* PhysicsConstants.g)

- (0.5f \* (1.0f / (UnitConverter.MMtoM(pObj.\_spec.WheelBase))) \*

pObj.\_spec.CurbWeight \* (pObj.\_acceleration.Z));

Wr = (0.5f \* c \* pObj.\_spec.CurbWeight \* PhysicsConstants.g)

+ (0.5f \* (1.0f / (UnitConverter.MMtoM(pObj.\_spec.WheelBase))) \* pObj.\_spec.CurbWeight \*(pObj.\_acceleration.Z));

pObj.\_Wf = Wf;

pObj.\_Wr = Wr;

}

*Lateral Forces*

This command determines the forces acting on the car when the car is going round a corner. From the research the lateral force is based on the cars angular velocity and forward velocity, by calculating what the rotational velocity the slip angles of the car can be calculated. Firstly the velocity at the specific orientation is calculated first, as shown below.

Vector3 velocity = new Vector3();

velocity.X = cs \* pObj.\_velocity.X + sn \* pObj.\_velocity.Z;

velocity.Y = -sn \* pObj.\_velocity.X + cs \* pObj.\_velocity.Z;

The velocity uses a transformation multiplication from the cars velocity against its rotation angle to work out its directional velocity. The yaw speed is then calculated to determine the rotational angles of the car.

float yawspeed = pObj.\_spec.WheelBase \* 0.5f \*

pObj.\_angularVelocity;

float rot\_angle = (float)Math.Atan2(yawspeed, velocity.Z);

The slip angle is to determine the maximum angle before the car begins to slide sideways. Once the slip angle is calculated the front and rear tyre slip angle are calculated, as the front wheels can rotate left and right so the slip angle would be different to the rear wheels, therefore the rear and front forces would be different.

float sideslip = (float)Math.Atan2(velocity.X, velocity.Z);

float slipangleFront = sideslip + rot\_angle - pObj.\_steeringAngle;

float slipangleRear = sideslip - rot\_angle;

Then applying these calculations can determine the lateral force on the front wheels and the lateral force on the rear wheels, therefore the car can slide on turning corners once it reaches the slip angle point.

Vector3 flatf = new Vector3();

Vector3 flatr = new Vector3();

flatf.Z = 0;

flatf.X = (float)(-5.0f \* slipangleFront);

flatf.X = (float)Math.Min(2.0f, flatf.X);

flatf.X = (float)Math.Max(-2.0f, flatf.X);

flatf.X \*= pObj.\_spec.CurbWeight \* 0.5f \* PhysicsConstants.g;

flatr.Z = 0;

flatr.X = (float)(-5.20f \* slipangleRear);

flatr.X = (float)Math.Min(2.0f, flatr.X);

flatr.X = (float)Math.Max(-2.0f, flatr.X);

flatr.X \*= pObj.\_spec.CurbWeight \* 0.5f \* PhysicsConstants.g;

Once the lateral forces are calculated the angular acceleration can be calculated, therefore the angular velocity can be determined, and so the velocity would have a velocity on the X axis if the car turns too quick causing a high lateral force and therefore cause the slip on the road.

*Angular Orientation, Velocity, & Acceleration*

The angular orientation, velocity and acceleration commands calculate and determine how the car turns left and right. The velocity and acceleration determine the speed of the car at turns on the track, which will lead to how the car slides and bends at a turn. The orientation determines the angle and direction of the car. Below shows how the angular acceleration command is implemented using the formulas from the research.

public void execute(PhysicsObject pObj)

{

float angularAccel = 0;

float b = (pObj.\_spec.WeightDistributionFront / 100) \*

(UnitConverter.MMtoM(pObj.\_spec.WheelBase));

float c = (pObj.\_spec.WeightDistributionRear / 100) \*

(UnitConverter.MMtoM(pObj.\_spec.WheelBase));

float bodyTorque = b \* pObj.\_flatF.X - c \* pObj.\_flatR.X;

angularAccel = bodyTorque / pObj.\_spec.CurbWeight;

if (pObj.\_velocity.Z <= 0)

angularAccel = 0;

pObj.\_angularAcceleration = angularAccel;

}

The angular acceleration uses the lateral forces implemented earlier to work out the body torque acting on the car. The condition check is used to ensure the car does not jus drag itself sideways even if the car velocity going forward has stopped. Once the angular acceleration has been calculated the angular velocity can be calculated, the angular velocity is similar to the calculation for forward velocity and is determined by time.

public void execute(PhysicsObject pObj)

{

float angularVel = 0;

angularVel = pObj.\_delta \* pObj.\_angularAcceleration;

pObj.\_angularVelocity += angularVel;

pObj.\_angularVelocity \*= 0.8f;

}

The orientation of the car can now be determined from the angular velocity. The steering angle is calculated to determine the maximum angle the wheels can turn, and then the angle is determined by incrementing the angle by the angular velocity. The angle then needs to be converted into a Quaternion matrix for the transformation to calculate the models rotation.

public void execute(PhysicsObject pObj)

{

float factor = 1;

if (pObj.\_currentGear == "Reverse")

{

factor = -1;

}

pObj.\_angle = 0;

pObj.\_steeringAngle = pObj.\_steeringPosition \*

MathHelper.Pi / 6.5f;

pObj.\_angle += pObj.\_delta \* pObj.\_angularVelocity \* factor;

Quaternion additionalRot = Quaternion.CreateFromAxisAngle(

new Vector3(0, -1, 0), pObj.\_angle);

pObj.\_rotation \*= additionalRot;

}

The factor variable is used to determine the direction of the turn, so either left or right this is determined by the gear the car is in, so when the car is in reverse the gear the direction of turn is opposite.

*Phase Conclusion*

This overall phase has shown how the research of the physics is implemented and tested in the application. The calculations were converted from maths to code, and some of the calculations did have some varied problems but were fixed after further research and reiterations on specific calculations. The main physics manager creates and holds a reference to the instance of all the commands that has to perform calculations and calls the execute function in the physics manager update function.

**4.1.7 Phase Six - Interface Information**

This phase is based on the interface and how some of the calculations are displayed on the screen for the user to see. Information that is relevant to the user is the Speed of the car in miles per hour (MPH), the current gear and the rev count. All this information has been calculated and can be accessed publically, so the game class can render this information on to the screen.

To draw the information onto the screen a sprite batch is needed, where the information for the application is given to the sprite batch, so it understands how to render the information on to the screen.

\_spriteBatch.Begin(SpriteBlendMode.AlphaBlend,

SpriteSortMode.Deferred,

SaveStateMode.SaveState);

There are three main functions that render the specific information, one for the speed, one for the gear, and the last one for the rev counter. The speed uses a string to display the information:

string speed = "" + UnitConverter.MStoMPH(\_car.\_velocity.Z);

string unit = " mph";

\_spriteBatch.DrawString(\_spriteLargeFont, speed,

posSpeed, Color.White);

\_spriteBatch.DrawString(\_spriteRegularFont, unit,

posSize, Color.White);

The gear uses several images, depending on the gear

\_spriteBatch.Draw(\_gears[\_car.\_currentGear], pos,null,

Color.White,0,Vector2.Zero,1.5f,

SpriteEffects.None,0);

The rev uses an image of a bar and a string to display the current rev counts.

string rpm = "" + (int)\_car.\_currentRPM;

string unit = " rpm";

\_spriteBatch.Draw(\_revBG, pos, null, Color.White, 0,

Vector2.Zero, 1.0f, SpriteEffects.None, 0);

\_spriteBatch.Draw(\_revCount, pos, new Rectangle(0, 0,

(int)(\_revBG.Width\*(\_car.\_currentRPM/6000)),

\_revBG.Height), Color.White, 0, Vector2.Zero,

1.0f, SpriteEffects.None, 0);

\_spriteBatch.DrawString(\_spriteRegularFont, rpm,

posRPM, Color.White);

\_spriteBatch.DrawString(\_spriteRegularFont, unit,

posSize, Color.White);

From these draw methods the UI is created to display the information for the user to see, as shown in Figure 12.



**Figure 12 Interface information**

*Phase Conclusion*

This phase has shown how the information calculated using the Exia engine can be displayed as user feedback. The information gives the user an understanding of the general mechanics of the car and gives them an idea of the speed and force is applied to the car, therefore testing the realism is easier.

**4.1.8 Phase Seven - Camera Effects**

This phase is to implement realistic effects with the cameras inside view. These effects are created by the forces acting on the car. When a car accelerates the front of the car lifts, and when the car brakes or decelerates the rear of the car lifts, this is caused by the weight transfer, which was calculated in the previous phase. By using the weight transfer the camera can be altered and tilted in directions that show an acceleration and deceleration of the car.

float totalWt = \_followObject.\_Wf + \_followObject.\_Wr;

percentageWf = (\_followObject.\_Wf / totalWt);

percentageWr = (\_followObject.\_Wr / totalWt);

if (percentageWf > percentageWr)

accelRot = (percentageWf) \* 0.15f;

else if (percentageWr > percentageWf)

accelRot = (percentageWf) \* 0.07f;

The code above shows how the differential difference of the front and rear down force to determine the direction of the tilt, then a multiplication factor is used to limit the amount of tilt to add realism to the effect, as shown by Figure 13 and 14.



**Figure 13 Forward Acceleration, camera tilts backwards**



**Figure 14 Braking Deceleration, camera tilts forwards**

Other camera effects include the turning force and angular velocity, so the camera tilts and rotates on the Z axis when the velocity on the X axis is greater than 0.

if (\_followObject.\_velocity.X != 0)

{

cornerRot = 0.001f \* \_followObject.\_velocity.X;

}

The velocity on the X axis can be negative or positive based on the direction the car is moving in, so therefore the rotational tilt will be between two points, so when the car turns left, the camera rotates clockwise, and when the car turns right the camera rotates anticlockwise. The factor of 0.001f is used to help limit the amount of tilt, so the camera view does not over spin left or right. This effect is shown in Figure 15 and 16.



**Figure 15 Camera Tilt anticlockwise on right turn**



**Figure 16 Camera Tilt clockwise on left turn**

*Phase Conclusion*

This overall phase has shown how the physics and dynamics of the calculations can be used to add extra effects to the application, which help demonstrate the realism in this racing simulator, and therefore give the user a better indication on what is happening in the application.

**4.1.9 Phase Eight - Input**

The application currently uses a generic GamePad to give the application input to test the physics and realism. For further realism the system will use a gaming steering wheel as its input device, the Logitech G25 Steering Wheel is currently not supported by the XNA framework but can be used by DirectInput which is part of the DirectX library. The device is installed on to the machine, and the SWInput and SWInputStates classes are implemented to handle the device and the input states.



**Figure 17 Logitech G25 Steering Wheel**

The SWInput class is created to help read the device information from the system by using DirectInput. The device is found by using the DirectInput Manager class to find what input devices are attached to the computer. Once the list is determined, an instance of a Device type object is created for the specific Logitech G25 device. Below shows the code used to find the device instance.

DeviceList controllers = Manager.GetDevices(

DeviceClass.GameControl,

EnumDevicesFlags.AttachedOnly);

foreach (DeviceInstance gameCtrlDev in controllers)

{

if (gameCtrlDev.ProductName ==

"Logitech G25 Racing Wheel USB")

{

\_controller = new Device(gameCtrlDev.InstanceGuid);

break;

}

}

From using the correct drivers for the device the input axis, triggers and buttons are given a specific instance name, so for example the brake pedal the instance name would be ‘Brake’. From this information, limits can be set for the input component, but this only works for components that are based on triggers or an axis. For the brake and accelerator the values of input would be between 0 and 1, the clutch would just be either a 1 or 0, and finally the steering wheel is based between -1000 and 1000, which indicates either the steering wheel is rotated left or right. The code for this implementation is shown on the next page.

if (\_controller != null)

{

\_controller.SetCooperativeLevel(handle,

CooperativeLevelFlags.Exclusive |

CooperativeLevelFlags.Background);

\_controller.Properties.AxisModeAbsolute = true;

foreach (DeviceObjectInstance doi in \_controller.Objects)

{

int rangeMin = 0, rangeMax = 0;

switch (doi.Name)

{

case "Accelerator":

rangeMin = -1;

rangeMax = 0;

break;

case "Brake":

rangeMin = -1;

rangeMax = 0;

break;

case "Clutch":

rangeMin = 0;

rangeMax = 1;

break;

default:

rangeMin = -1000;

rangeMax = 1000;

break;

}

if((doi.ObjectId & DeviceObjectTypeFlags.Axis) != 0)

{

\_controller.Properties.SetRange(

ParameterHow.ById, doi.ObjectId,

new InputRange(rangeMin, rangeMax));

\_controller.Properties.SetDeadZone(

ParameterHow.ById, doi.ObjectId, 200);

\_controller.Properties.SetSaturation(

ParameterHow.ById, doi.ObjectId, 10000);

}

}

\_controller.Acquire();

}

The key part to retrieve the information to detect input change is by the SWInputState class. The SWClass holds a get function to return a SWInputState, the function uses the JoystickStates object to read the devices input states, and from that information the SWInputState can convert the information into states for the game to pick up.

public SWInputStates GetState()

{

JoystickState state = \_controller.CurrentJoystickState;

inputStates.SetGear("First" , state.GetButtons()[8]);

inputStates.SetGear("Second", state.GetButtons()[9]);

inputStates.SetGear("Third", state.GetButtons()[10]);

inputStates.SetGear("Fourth", state.GetButtons()[11]);

inputStates.SetGear("Fifth", state.GetButtons()[12]);

inputStates.SetGear("Sixth", state.GetButtons()[13]);

inputStates.SetGear("Reverse" , state.GetButtons()[14]);

inputStates.Accelerator = state.Y;

inputStates.Brake = state.Rz;

inputStates.Steering = state.X;

inputStates.Clutch = state.GetSlider()[1];

return inputStates;

}

The SWInputState class just holds data and properties for the acceleration, brake, clutch, steering and the gears. The gears of the input are stored in a dictionary, which holds a name for the gear and a value which determines if the gear is active or not. Then the class has a get property which returns which gear is currently active.

public String CurrentGear

{

get

{

foreach (KeyValuePair<String, int> i in Gears)

{

if (i.Value != 0)

{

return i.Key;

}

}

return "Neutral";

}

}

*Phase Conclusion*

This overall phase has shown how other devices that are not XNA specific can be imported into the XNA framework by using the DirectInput library. The steering wheel now adds extra realism to the application, with dynamics clutch control, gear changing, and steering mechanisms. This will give the users to test the application, and judge the realism easier.

**4.1.10 Phase Nine - Sound Effects**

The sound effects are added and implemented by using the XNA framework to play sounds in the format of a wave (.wav) file. The sound can be loaded in by using the content manager, and the sound loaded is type SoundEffect as shown below.

Content.Load<SoundEffect>("SOUND//Gear1")

A SoundManager class is created to help store the sound effect data into a dictionary, so each of the sound effects have a name reference as its key. The SoundManager class is a singleton class as only one instance of this class is needed to be instantiated. When the sound effect is added, the sound effect is converted into a SoundInstance object by calling the CreateInstance function; this gives the sound effect a more dynamic way of playing the sound.

The SoundManager class also has the ability to play a sound, by calling the PlaySoundInstance function, which takes a set amount of parameters which are the key name, the volume and the pitch the sound should play at. Below shows the implementation of the PlaySoundInstance function, which triggers the sound to play.

public void PlaySoundInstance(string name, float volume,

float pitch)

{

if (volume > 1)volume = 1.0f;

if (volume < 0.1) volume = 0.1f;

if (pitch > 1) pitch = 1.0f;

if (\_soundboardInstance.ContainsKey(name))

{

\_soundboardInstance[name].Volume = volume;

\_soundboardInstance[name].Pitch = pitch;

if (\_soundboardInstance[name].State !=

SoundState.Playing)

{

\_soundboardInstance[name].Play();

}

}

}

The volume and pitch are given a limit so the sound is always playing but does not go out of bounds of what the XNA framework needs. The statements are used to check if the dictionary has a sound with the given key name, and if the sound of the specific instance is currently playing.

When the car object triggers the sound to play for the engine, the car object uses the rev counter to determine the pitch and the volume of the sound, also the current gear determines which sound should be played.

SoundManager.getInstance().PlaySoundInstance(\_currentGear,

(\_currentRPM / 6000), (\_currentRPM / 6000));

*Phase Conclusion*

This phase has shown how the sound effects are created in the XNA framework, and how the application uses the SoundManager to play specific sounds. When the sound effects are triggered or played, the specific sound played can be altered by changing specific values for the volume and pitch based on the physics of the engine for engine noise, and even tyre physics can give dynamic sounds when braking. These effects all provide the application a realistic effect.

**4.1.11 Conclusion**

The Exia engine was developed using basic object oriented development, by using the formulas and equations that were found from the research has helped develop the applications core physics engine. The algorithms for the physics engine were created and processed within its individual command class, as shown by the design, so therefore the classes did not have a large amount of calculations to process, but only its relevant calculations.

The Game engine was taken lightly compared to the Exia engine, as the main focus was the development of the physics engine, which was the core phase of implementation that demonstrated realistic driving in a racing simulator. The Game engine was developed using simple concepts, and no major game engines concepts were implemented, like the GameObject abstract class.

The overall implementation has shown how the application was developed from using a phase and iterative base implementation. Each of the phases was based as a feature list and story board, where each of the components in the list was developed and tested before moving on to the next phase. The testing is shown in the next chapter, where it shows the test plans created for each of the phases and also the results from the different tests executed.

**4.2 Testing**

**4.2.1 Introduction**

This section is to look at some of the ways used to test the application. The testing was implemented during the development of the application, as the methodology states that the testing is done during the process of development, so tests are being made as new code is being developed. The core method of testing that the methodology states is known as Unit testing, but due to time concerns and difficulties, Unit testing was not implemented, but standard regression and test plans were created in its place. This chapter is to look at how these two tests were implemented and created to test the applications fundamentals.

**4.2.2 Regression Testing**

Regression testing is done by testing the application is still functioning when parts of the code are changed, or when new code is added to the application. This helps ensure that the new code hasn’t broken or changed any previous code that was implemented. This also helps debug the application, as many times in development, fixing one bug can lead to others, but understand how bugs occur can make the process of bug fixing faster and quicker. This can lead to chains of bugs that started from one bug and have lead from one problem to the next. This style of testing can take time, as old tests are still being executed as the application grows. Therefore a test plan is created to give the application a set of tests that need to be tested when code changes. The list of tests helps assure that core functionality does not get broken, but if they are, they can be found easily with a test plan. The test plan for this application is shown in Appendix F.

**4.2.3 The Test Plan**

The test plan is split into different phases as part of the methodology; each phase of development is a list of core features that need to be implemented. Therefore this helps control what tests are needed in the current phase, so it can limit the amount of tests being executed and also unnecessary tests. This section is to discuss the tests plan for each phase.

*Phase One – Models Testing*

This phase was fairly simple as little errors were found. The tests are to assist in the choice of a suitable model for this application. Loading the models into 3D Max showed which models may cause some problems, and from the tests the results of each model were as expected.

*Phase Two – Graphics Testing*

During this phase the graphics were implemented, and due to this phase was base on the use of Riemer’s Tutorials, the results of the test were successful. The camera development used some core math’s basics to implement and the results of the camera test were all successful.

*Phase Three – Advance Graphics Testing*

This phase looks into testing the advance graphics, as the results of the tests, were successful. Problems were encountered when implementing the collision detection, as the collision was occurring from the centre of the car, as no boundaries were initialized for the car model.

*Phase Four – Refactoring Testing*

Refactoring the application so it was clean and also dynamic is a big change, so full tests of each of the phases were executed. The tests were based on checking if the core functionality of the objects were still working, as the refactoring was taking the code and moving the relevant parts into its own object class.

*Phase Five – Exia Engine Testing*

The physics engine is implemented to help create the dynamic effects in the application, as most of the physics were based on calculations; the results were outputted on the compilers command line. The results of the majority of the tests were successful, as the expected values calculated by hand, were outputted and shown in the command line; therefore the equations and formulas were being implemented correctly. As the majority of equations and calculations are linked together when a new command is implemented for an equation, other equations were re-tested to ensure the numbers outputted were still correct.

*Phase Six – Interface Information Testing*

This phase uses the calculation results to display them on the screen for visible feedback to the user. The results showed that the information can be seen on the screen, and the values displayed were being updated correctly when a calculation is executed.

*Phase Seven – Camera Effects Testing*

This phase showed how the camera effects were tested, and the results of the test showed that the effects worked and showed realism in games. The implementation of the camera effects were based on the physics calculations, so to ensure they worked correctly extra tests were performed as figures change depending on the forces and dynamics acting on the car.

*Phase Eight – Input Testing*

When the steering wheel device was adapted to the application, the input from the steering into the application needed to be tested as the XNA framework did not officially support the steering wheel device. The test plan for the input showed each of the components of the device and how they were used to test the input command to cause the car to react. As the input determined the majority of the physics calculations, the engine was retested to ensure the calculations were correct.

*Phase Nine – Sound Effects Testing*

This phase was to test the sound playing when the car moves, and when the car changes gears, but as this wasn’t a visual effect there were no images available to show the results, but the results were successful and the sounds did play when certain events occurred when the car was driving round the track.

**4.2.4 Conclusion**

Overall the testing was a success, as the majority of the tests showed positive results, the application was able to demonstrate some of the core features in real world physics in games. The test plan was a productive way to understand what needed testing, and gave the application a direction to head, instead of being slowed down due to bugs and problems. There were a few problems that occurred but were solved and tested to ensure bugs don’t reappear. The methodology structure has helped define some of the tests that needed to be executed, as the tests were based on phases feature lists.

5. Critical Evaluation

**5.1 Introduction**

This section is to look at how successful this overall project was, and to evaluate the key sections within this project. The evaluation will look at the objectives and verify if they were achieved, and also look into the core individual sections of the project to understand if requirements and specifications were implemented. Understanding the success of the project can also help evaluate what can be improved within the project, as key points will be made on changes that would happen if this process was to start again. The key success to this project is to be able to learn from what has been processed within this project, and use the new skills learned, to adapt it to industrial work and future projects. The overall methodology will also be taken into account, as some of the core moments were to able to adapt to new and different software methodologies when tackling a project of this scale.

**5.2 How successful was the project?**

The success of this project is to check if the aims and objectives were achieved and weather the research was able to declare the demand for this project to be implemented. The important point of this, was to develop an application that simulates realistic racing dynamics, and to show how and why realistic games are fun and in demand. The research was designed to gather information from the market, the industry, and the real world physics, which all helped determine the project’s success.

Information gathered helped understand what the market wants and demanded, also what the current state of the industry was, what skills are needed and what methodologies were being implemented into major projects. Finally the dynamics helped understand the scale of the project and how the project will be implementing the dynamics into a game, so that the application can demonstrate the core features of physics in a simulation. The overall research showed relevant and quality information which determined that this project needed to be executed to see if real world dynamics can be implemented into a form of a game.

The aims and objectives were achieved by processing through the Research, Analysis, Design, Implementation and Testing stages. Each staged guided the project further and gave the project its end goals. The research helped achieve some of the objectives, as the research determined the key concepts of vehicle dynamics, and how calculations were calculated in the real world, this was a large learning curve as understanding the concepts were difficult at times, but were necessary to understand how these dynamics can be implemented into the game. The research also determined what was in demand in terms of realistic simulators for racing games, which indicated what was lacking in the racing simulators market. The industry research showed what skills were needed in the gaming industry, which indicated how this project was to progress with its implementation, as core skills needed to be learnt for a successful development of the application. The overall research on vehicle dynamics and software methodology helped design the architecture and structure of the application. The integration of the steering wheel, gave the application a more realistic feel to the game, as it helped demonstrate how driving a car in a game can be similar to driving a car in real world.

Overall this shows how the aims and objectives were achieved by the information gathered from the research. Also the research helped determine that this project can make a difference in some realistic racing simulators. Overall the project is a success, as the final application was able to demonstrate vehicle dynamics in a game, and was made realistic.

**5.3 What improvements can be made to the project?**

There are always many improvements that can be made in a project, as processes can be executed differently when developing a different project. These improvements will help learn new skills and help improve the planning of the project. Some of the major improvements for this project are discussed in this section.

One of the major improvements that can be made is project management, as the project progressed; a lot of the tasks were delayed due to hold ups in other sections of the project as shown in Appendix B, which shows the changes of the project plan as development of the project continued. The project management can be improved by understanding the full process of a task; therefore the allocation of sufficient amount of time to certain task and better estimations of tasks can be more accurate, therefore the plan would have needed fewer changes.

Other improvement is to understand the full concepts of the physics that was needed for this project. The scale of the vehicle dynamics is large, and there is a lot to learn, as anything can affect the car. For example the brake discs materials can affect the amount of brake force, the engines valves and pistons tension and friction forces that is opposing the movement, the tires pressure, density, and materials, all of these extra concepts can all effect the cars drive line and dynamics. The overall physics on car is a complex topic, and researchers all over the world are still trying to understand the full concept of aero dynamics. The way to improve the learning step of vehicle dynamics is to lower the scale of the dynamics that are needed in the application, focus on a section of the physics, like the engine or just the tires, not everything. This will help improve the time scale of the research and also lowers the intense application that is needed to demonstrate so many dynamic concepts.

Improving the understanding of coding in a library that is suitable for the industry, as the point of this project is develop a racing simulating game, so why not use a coding language that majority of game industries use. At the moment there are no company that uses full time XNA development within their games development. Being able to develop an application using a coding language that is used in the industry today would help learn new skills that can be adapted into the industry easier. By learning skills and understanding the industry faster would help improve the choices made in terms of frameworks and languages, so the right choice is made to help improve development skills.

Overall there are probably several more improvements that can be made, but these three points are enough to take the next project to the next level, and from that project the next set of improvements can be determined, therefore there is a learning progression as more and more projects are developed.

**5.4 Following the methodology practices**

The overall methodology has shown the project a new way of developing applications, as the PXP methodology is an unknown methodology, after applying it to this project, has shown that this methodology can work with other projects. Following some of the stages were difficult, as stages may have been skipped or altered due to lack of time. The practice showed how the planning of the methodology was implemented and created to help process the development stages of the methodology. This helped within this project, as the plan set each of the iterations as set of features, which helped guide what needed to be implemented, and when it has to be done by. The feature list help keep a check list on the implementation but also the testing of the phase, so when the application is tested, the plan understood what needed to be tested so tests were executed easily. The one problem that was encountered with this methodology was the unit testing, as this was skipped and left out due to the lack of time, and also wasn’t really necessary as unit testing takes a while to develop the actual tests for a specific class, but also unit testing only helped if the development was more than one, so integrated testing can be applied, but once integrated the unit tests can be ran to ensure the integration has broken existing code and functionality. The overall methodology has shown a positive success as the methodology was used to its best, but as it was a new methodology, the methodology was not tested enough to state it’s an official methodology, so researching on information on this methodology was difficult to find relevant information.

**5.5 Conclusion**

Overall this evaluation has shown the good and the bad points within this project, but the project is still a success, as positive and negative evaluations have shown what can be improved, what problems occurred, and how to solve the problems. This evaluation has indicated the new paths that can be taken into new projects in the near future, and how development within this project can be ported and used in other projects. Overall this project has shown why realism is needed in games, and why people demanded it, and it was all because people want to escape reality and can therefore go into games. Being able to play games and not put people in danger is a bonus, and that’s why people want more from games to be more realistic.