**Computer Science**

**Programming Project Report**

Focus of Project

**Educational Planetary Motion Simulator and Space Game**

Produced by

**Gregory Wright**

Currently Studying at

**Southend High School for Boys**

Project Offset

**September 2016**

**Notes for Nzemeke:**

**There is one mention where it suggests looking at the design section. This hasn’t been completed yet and so is a meaningless reference. The mentions of numbering systems can be ignored for the time being. This is because the sources of the requirements and game features are not yet annotated. The two additional resources that could be collected is an interview with the head of department and teacher questionnaires. Would doing this be worthwhile?**

# **Table of Contents**

[Table of Contents 2](#_Toc462965170)

[Chapter 1 Analysis 4](#_Toc462965171)

[Problem Identification: 4](#_Toc462965172)

[The Client: 4](#_Toc462965173)

[The Current System: 4](#_Toc462965174)

[Research: 4](#_Toc462965175)

[Defining the Problem: 4](#_Toc462965176)

[Brief Description of Proposed Solution: 4](#_Toc462965177)

[My Solar System Simulator: 5](#_Toc462965178)

[Finding the Required Solution: 5](#_Toc462965179)

[Current Solutions to Similar Problems: 6](#_Toc462965180)

[Approaches to solve problem: 7](#_Toc462965181)

[Kepler’s Laws of Planetary Motion: 8](#_Toc462965182)

[The Roche Limit: 10](#_Toc462965183)

[Choice of language: 10](#_Toc462965184)

[OpenGL and SDL2 APIs: 11](#_Toc462965185)

[Stakeholders: 12](#_Toc462965186)

[Access to System: 12](#_Toc462965187)

[Teachers: 12](#_Toc462965188)

[Students: 13](#_Toc462965189)

[Server Host: 13](#_Toc462965190)

[The Proposed Solution 13](#_Toc462965191)

[Description of selected solution: 13](#_Toc462965192)

[User Requirements: 14](#_Toc462965193)

[Hardware & Software Requirements: 15](#_Toc462965194)

[Game features: 15](#_Toc462965195)

[Limitations: 16](#_Toc462965196)

[Limitations and Workarounds: 16](#_Toc462965197)

[The success Criteria: 18](#_Toc462965198)

[Chapter 2 Design 19](#_Toc462965199)

[Objectives of game: 19](#_Toc462965200)

[Chapter 3 Development 19](#_Toc462965201)

[Chapter 4 Evaluation 19](#_Toc462965202)

[Appendices 19](#_Toc462965203)

[Appendix 1: 20](#_Toc462965204)

[OpenGL & SDL2 research sources: 20](#_Toc462965205)

[OpenGL & SDL2 tutorial program: 20](#_Toc462965206)

[Appendix 2: 22](#_Toc462965207)

[My Solar System: 22](#_Toc462965208)

[Appendix 3: 22](#_Toc462965209)

[Numbering Systems: 22](#_Toc462965210)

[The Problem: 22](#_Toc462965211)

[Stakeholders: 23](#_Toc462965212)

[User Requirements: 24](#_Toc462965213)

[Appendix 4: 25](#_Toc462965214)

[Image references for Kepler’s Laws Research: 25](#_Toc462965215)

[Appendix 5: 26](#_Toc462965216)

[Success Criteria table: 26](#_Toc462965217)

# Analysis

## **Problem Identification:**

### The Client:

The client is one of the teachers in the physics department at Southend High School for Boys. Dr Middlemast, the client, teaches all years throughout the school (ranging from years 7 to 13). He teaches many sixth form classes (year 12 and 13). The issue is that he has only few interesting resources at his disposal for orbital mechanics in the A Level physics course. The resources also do not interest the students in a further career in physics. The client requires a system that can engage his students within this topic and educate them on many aspects in an interactive environment. The teacher has the idea that if students understand a topic to a higher level than required, it will make the actual questions on the topic seem easier. If the new resource works well for his classes, other teachers would be able to use the tool as well. Particularly circular motion is the region within orbital mechanics that is taught for the A level course.

### The Current System:

The client currently uses worksheets and a PowerPoint presentation to teach the students the appropriate knowledge on circular motion. There are also occasional experiments. The issues with this being that some students find it more difficult to understand the concept. This extends the time required to study the topic which otherwise could be used to cover other content. This method makes it difficult for the students to quickly visualise what is going on and is why there is a need for a new system. The PowerPoint presentation provides the appropriate knowledge for the topic (such as facts and notes). After going through the presentation, the knowledge gathered is reinforced with questions from the worksheets of ranging difficulty. The client also occasionally uses the My Solar System simulator. This simulator is useful but has many limitations. This approach would still be used but extended and altered to include the new resource developed as required by the client. The new system resource, proposed by the client, would be a space game as well as being an improved solar system simulator. This would replace the My Solar System simulator

## Research:

### Defining the Problem:

After meeting with the client for the first time, he briefly described his problem. It was asked of him to complete a questionnaire on the matter (look at appendix 3) so that he had time to define specific sections of the problem.

### Brief Description of Proposed Solution:

A solar system simulator and space game was the solution chosen. This is to be an improvement on the current simulator that is occasionally used by the client.

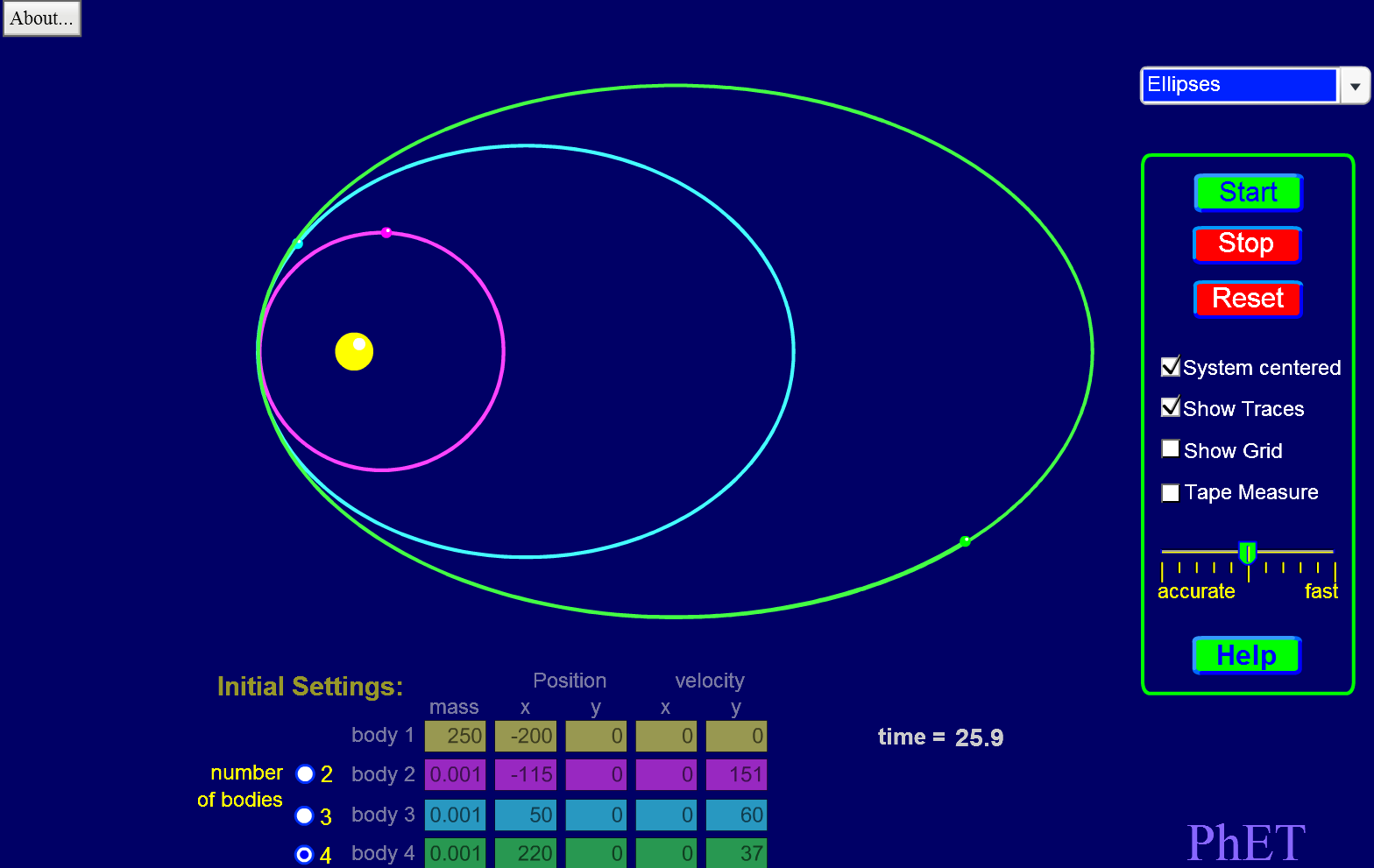
### My Solar System Simulator:

(Source in appendix 1)

The simulator is the current resource used by the client and has many limitations. It contains many features including the orbit tracings, the ability to change orbital speed and to start and stop the simulation at any time. The client wishes to have many of the same features as this simulator and so many of the requirements determined are similar to the aspects of this program. These were discussed with the client, as he understood the simulator well, during the interview stages. Some of the issues are stated below.

#### Some Limitations and Missing Features of Simulator:

* A maximum number of 4 bodies can be modelled and so more complex systems cannot be represented.
* Orbits are limited to two-dimensional space.
* The time interval is to an accuracy of only 1 decimal place.
* The initial values are provided/inputted but not current values.
* Doesn’t allow creating a solar system from scratch instead modify defaults.
* Uses arbitrary units so actual values cannot be calculated.



#### Image showing My Solar System simulator

### Finding the Required Solution:

Many research methods were used to find the required solution. This started with a simple discussion with the client to get an idea of what he expects. This happened shortly after receiving the client problem questionnaire.

For the client questionnaires, any questions that the client didn’t understand (may be due to their ambiguity) were discussed with the client face to face to get the appropriate responses the first time before going back to the client to collect the results.

#### First Interview with client:

To initially determine the requirements, a face to face interview was carried out with the client. This is so that the best opportunity was available to gather information on the system and make any queries that were found necessary.

First the client was asked about the game design requirements. He stated that the game had to be interactive such that his students can be engaged in the game. He also stated that it should be in 3D. Later on in the interview he mentioned that it was vital the windows 7 school computers could use the game. This would be so that the students could gain access to the game during and outside of lesson. There were many other requirements also discussed. He also talked about some of the issues with the current system in more detail.

Notes taken from the interview showing the user requirements gathered at this stage are available in appendix 3.

#### Game Features Client Questionnaire:

The questionnaire was used to gather an idea of what the client expected to be within the space game and simulator. This is because a game is a very broad description of what the user expects. The client was helpful in determining many of these features and are described within the *Game Features* section. what would happen in the game and so knowledge of what would need to be programmed in the later stage.

The questionnaire can be found in appendix 3.

#### Second Interview with Client:

During the second interview with the client, some additional requirements were gathered including those considered non-essential. The main purpose of this interview was to clarify that the approach chosen to simulate the model was acceptable. This meant making sure with the client that the equations derived from Kepler’s laws was the best method over the alternative options discussed in the *Approaches to Solve Problem* section.

Notes taken from the interview showing the user requirements gathered at this stage are available in appendix 3.

### Current Solutions to Similar Problems:

Research was carried out on existing simulators and space games to see if the approach was feasible and be used for the solution.

#### Kerbal Space Program:

Kerbal Space program is a mixture between a space exploration game and a simulator. It allows the player/user to build a variety of spacecraft (rockets, space planes, satellites, etc.) and send them into orbit around other planets and moons in the solar system. It can show how planets orbit a star as well as how spacecraft orbits can be changed. This is very relatable to the A level physics course circular motion topic. This fun space game however is far too complex for what is required by the client. It requires a large amount of processing which would mean an expensive computer system would be needed. The majority of computer systems (like the often cheap school systems) would not be able to run the game at a high enough frame rate to have a joyous experience. The next issue being that it is very difficult (and time consuming) to produce any successful crafts. Even though there is the ability to interact with the system, there is no explanation on how the calculations are performed. The necessary data is provided to carry out calculations on orbital mechanics although there is no explanation within the program on how to do so. This would require researching online to find tutorials on such calculations which would also take a significant amount of time.

### Approaches to solve problem:

There are many methods to correctly simulate a model of a solar system. The approaches researched are discussed below.

#### Simple Equation of the ellipse:

The simple mathematical equation of an ellipse could have been used. This would be able to then correctly simulate the orbital path of the satellite. The primary would simply be placed at one of the two foci. The issue with this method being that time couldn’t be considered. The speed at which the satellite would orbit around the primary would appear constant. This isn’t the case with orbits where the further away the satellite is to the primary, the slower it travels. Another issue being then simulating the different positions of the satellites when there are more than one orbiting a single star. The satellite that is closer to the primary should travel faster (not the same speed which would be the case) as the satellite further away. The equation of the ellipse is shown below.

#### Circular Orbits:

The other option is to use only circular orbits. These are far simpler to calculate than the proposed solution. The issue with this method being that orbits of planets around stars are not often circular. They are instead slightly elliptical although in many circumstances near enough to use this method. This approach would limit to circular orbits. This would be an issue if a highly elliptical orbit is desired such as that for a comet. It is also desired that the program goes beyond that of the A level physics. This cannot be the case if an A level physics topic is solely used. The equations for this method are listed below.

#### General Relativity:

Einstein’s theory of general relativity can be used to calculate the position of planets. It is however far too complex and heavy processing would be required. Also it isn’t relatable to the A level physics course. This makes this approach unsuitable.

#### Newton’s law of gravitation:

This method could correctly simulate the elliptical orbits of multiple objects; these laws are expressed in the A level physics course. This means that another method should be used because the user doesn’t want a system that uses only content from the A level physics course. An equation for this method is shown below.

#### Equation’s derived from Kepler’s Laws:

*Look at Kepler’s Laws of Planetary Motion* section to understand what the laws are.

Kepler’s laws go one step further at representing orbits over that described in circular motion for the A level course. Also a circular orbit can be produced with this method as well as elliptical. This means the calculations within the A level course can be used to calculate the positions that the satellite will be in relative to the primary. It isn’t a too complex or simple solution which are the main downfalls with the other approaches. Kepler’s laws are looked at in more detail on University courses. This makes the approach relatable to what students would study at a higher level and is what the client requires. This concludes that the equations derived from Kepler’s laws should be used.

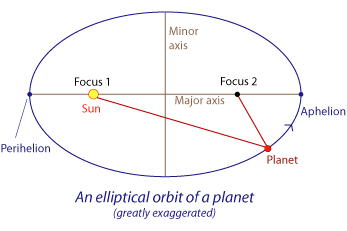
### Kepler’s Laws of Planetary Motion:

*(Look at appendix 4 for image references)*

Kepler’s laws were originally used to determine planetary motion however can still be used to determine orbits of objects with insignificant mass (a satellite) compared to what they are orbiting (the primary). Also the laws originally were only used for our solar system.

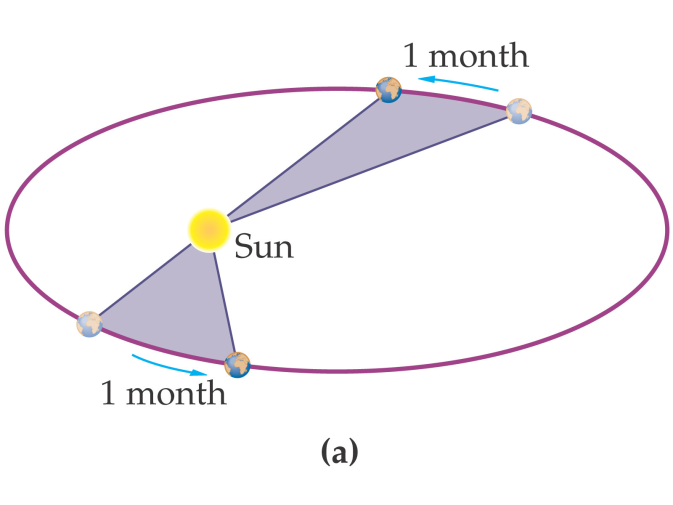
#### Kepler’s first law:

A planet rotates around the sun in an elliptical orbit. The sun is at one of the two focal points of the ellipse.



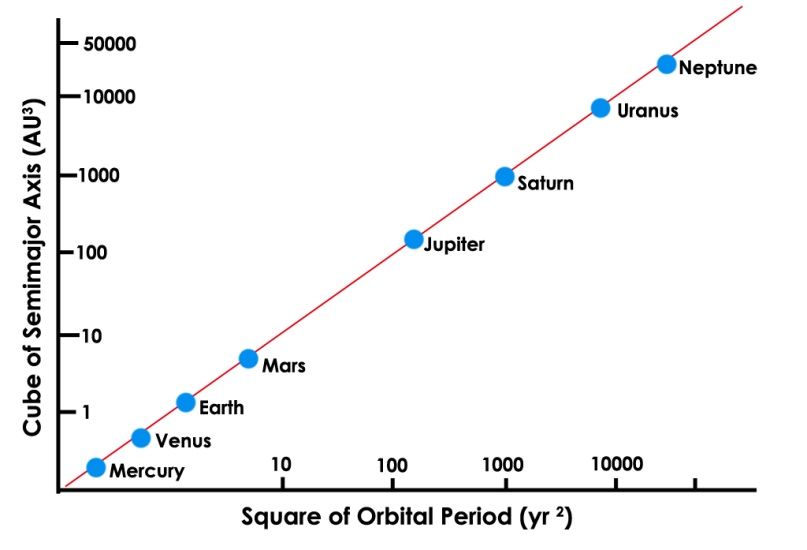
#### Image showing Kepler’s first law

#### Kepler’s second law:

That over a fixed length of time, the area inside the initial and final position of a planet and the sun at one of the foci is constant.

#### Image showing Kepler’s second law

#### Kepler’s third law:

The time period squared is directly proportional to the semi-major axis cubed.

#### Image Showing Kepler’s third law

#### Implementation:

After many hours of online research, a method to display the position of a satellite (i.e. a planet) relative to a primary (i.e. a star) was found. These are derived from Kepler’s laws of planetary motion. The relative position can be calculated as a function of time. This is through 4 stages of calculations. The first calculates the mean anomaly. The next calculates the eccentric anomaly of an orbit. The next converts that angle into the true anomaly. The final calculation is used to calculate the distance between satellite and primary. The true anomaly and distance is then used as part of a spherical coordinate system relative to the star.

Equations for implementation:

* *(calculation for mean motion)*
* *(calculation for mean anomaly)*
* *(iterative approach required to solve for E/eccentric anomaly)*
* *(Rearrange to solve for theta/true anomaly)*
* *(calculates heliocentric distance)*

### The Roche Limit:

Other factors occur that could limit or affect the possible orbits of planets. The one specified by the client is the Roche limit. This is the distance within which the gravitational field strength is strong enough to prevent another smaller object from being held together by gravity. This can be implemented by calculating the distance of the Roche limit and not allowing orbits to be simulated within the region.

### Choice of language:

The languages being compared were chosen by the amount of access to knowledge required on the language. By this referring to already knowing how to program using the language (in the case of Python & C++) or ease of learning the language (Visual Basic being taught in lesson). These languages are not the only languages likely to be used for the project but being the primary language to make up the solution.

#### C++:

* Very efficient and fast to run code.
* Complex language making it more difficult to program and may require more lines of code (increasing file size before compilation) and can extend the time it takes to be developed.
* The complexity allows for the production of programs with more specific functionality.
* Compiled so more difficult to debug at runtime.
* External Libraries are required to produce graphics unlike Visual Basic.
* A mixture of higher and lower level code gives the programmer many ways to achieve the same result and can be chosen depending upon the circumstance.
* The lower level aspect allows for advantage over other languages as these features make for more efficient code. These include the use of pointers and direct memory addressing. These features provide alternative faster access of memory.
* Compiled language so is not as portable due to the translation to object code is highly dependent upon the computer system specification.

#### Visual Basic:

* Use of window processes allows for ease to produce graphics in fewer lines than C++.
* Not as efficient as C++ or as fast to run.
* Easier to produce and less complex than C++.
* Designed for easy production of graphical user interfaces.
* Not cross compatible with operating systems other than Windows.
* Compiled into intermediate code so has many of the benefits of being compiled yet still contains many disadvantages from then being interpreted at runtime.

#### Python:

* Lack of complexity creates limitations and becomes less efficient than C++.
* External libraries are required to produce graphics unlike Visual Basic.
* Simplistic nature makes it easy to understand what is going on and there are only a few alternative methods of producing the same results so easier to decide which is most efficient.
* Interpreted language so slower at runtime than C++.
* Being interpreted allows for easier debugging.

#### Preferred language:

* The optimal language to use has to be C++. This is because it is highly complex allowing for the greatest amount of choice upon how to approach a task. This means the most efficient approach can be chosen. As it is usually translated to machine code (unlike the other two), it would likely have the fastest runtime. The ease of errors arising in the compiled code means that more thorough debugging is required.
* In addition, C++ is optimised for using classes and so makes object oriented programming ideal. This should help improve the clarity and efficiency (i.e. reusability) of code and such make it easier to debug and help through the design and testing stages (i.e. each class would be designed and tested separately).

### OpenGL and SDL2 APIs:

#### OpenGL:

OpenGL (Open Graphics Library) is an application programming interface that allows for the design and production of game engines. The main purpose being to produce graphics (usually 3D graphics).

#### SDL2:

SDL2 (Simple DirectMedia Layer 2) libraries are used to provide low level abstractions for multimedia hardware components. This being used in this case to produce a window and for event handling.

#### Cross-Compatibility:

These two libraries are used over alternatives (such as DirectX) because of their ability to be cross-platform allowing a greater number of users access to the software. The content is wide spread and will be able to run on most computer systems. Look at the *Limitations* section on OpenGL for why this may not be always the case.

#### Suitability of APIs:

To certify that OpenGL is suitable to produce the graphics for the project, a significant amount of time was spent going through many tutorials in learning how to use the API. Many of these tutorials were in video and document formats. The main tutorials used are linked appropriately in appendix 1. After going through many tutorials it was determined that OpenGL is suitable. The programs produced graphics with enough complexity for the scope of the project. Also it was found that OpenGL wasn’t too complex to still be able to produce a program within the appropriate timeframe. The “jump” between OpenGL 2D to 3D graphics was also insignificant at increasing the complexity of the project. Producing 3D graphics only adds a few extra challenges which are easy to work around. This means that this requirement suggested by the client is achievable within the timeframe.

#### Additional Challenges for 3D graphics in OpenGL:

* The added dimensions required for the matrix transformations as well as other calculations.
* The slower runtime due to carrying out depth tests. This is where an additional buffer is used to store depth information. OpenGL compares the depth of each pixel when there is multiple wanting to be located at the same location on the screen. The closest pixel (lowest Z value) is displayed. If this didn’t occur, OpenGL would overlap objects depending upon the order that they are drawn.

#### Further Reading On OpenGL:

After learning the capabilities of OpenGL, A book was found on the matter. The book (look at appendix 1) covers the basic ideas of OpenGL that was covered in the online tutorials and then goes into far more depth into these ideas as well as alternative approaches. The book also covers many advanced features of OpenGL.

#### The Need of SDL2 with OpenGL:

OpenGL is limited because it can produce graphics but doesn’t have the infrastructure to display those graphics. By this, the most noticeable aspect is the lack of a window to display the graphics. To do these features SDL2 can be used. It is used because it is considered simple to setup and compatible with OpenGL. It has many other features as well like event handling. SDL2 can be used to produce simple 2D graphics but isn’t capable of producing high performance 3D graphics. This is a specified requirement and as such means it cannot be solely used to produce the window as well as the graphics to display.

## Stakeholders:

Look at appendix 3 for research and analysis on the stakeholders.

The stakeholders are the groups or individuals that have an influence on the solution. These include the users of the proposed solution (teachers and students) as well as the server host.

### Access to System:

Both the teachers and students would have the same access level to the system. This is due to each user being able to play the game and use it as an educational tool. There is no requirement to not allow full access to the system. The only exception being user scores stored on a database.

### Teachers:

Teachers would want to use the game as an educational tool at their disposal. This would allow them to help educate their students to better understand the topic. This resource could potentially become valuable in helping students grasp concepts. This is similar to the needs of the client due to the fact the client himself is a teacher. Other teachers would also not have many good resources available for this topic within the school. This is because the majority of resources are shared within the physics department. This resource should also help make teaching the content easier for the teacher.

### Students:

There are a variety of students with varying levels of interest and needs for this solution. These students are limited to those that are studying A level physics. This is because the resource is for students studying that course according to the client. The resource should enable the students to visualise what is going on and how everything works within the topic. The students are more likely to enjoy the use of the simulator rather than just going through a textbook on the matter. This should make learning for the student quicker and easier and also help them decide whether they wish to go into further education in physics.

### Server Host:

For the online aspect of the game, a server is required. This would then mean a server host would be needed to maintain the server. This host would grant access to the stored information online. As peer to peer networking would primarily be used, relatively little processing is required by the server. This is also the case for the leader board where each user would only download the information on the user scores and upload new scores. As the scores would be updated for each user, there is a limited amount of data required to be stored per user. This means that the storage capacity required isn’t likely large as a classroom full of students’ information wouldn’t take up much space. During the development the server would be hosted locally as there wouldn’t be many processes being carried out and testing would be more efficient and controlled. As the program becomes more popular with more users, an external server would become more appropriate. The server host would then be the organisation running the server. This may be the school on the school server or a company that hosts servers in another location. This being the stakeholder. In the case of a company running the server, the only concern is payment. The payment is dependent upon many factors but mainly the amount of processing and data traffic going through the server. These factors would be the main concern with the school server as their servers are far less performing.

In the case of this game, like mentioned already, little processing would be needed by the server so should only be an insignificant impact on the server. If the program becomes even more popular another situation would need to be considered but that is outside the scope of what is expected by the client at this stage.

## The Proposed Solution

### Description of selected solution:

A space game and solar system simulator where the aim is to get from one orbit to the next. The user has to get to the location by correctly calculating trajectories. The game would be educational such that teaching the user valuable knowledge and understanding of the physics behind space travel. There is also a demonstration mode which only includes the simulator element.

### User Requirements:

Look at appendix 3 for research methods on the user requirements as well as understanding the numbering system.

#### The Users:

There are two main user groups. The students and the teachers. The client is also a teacher and so the requirements are similar for both the teachers and the client. Research carried out on these users included interviews and questionnaires with the client and an online questionnaire with the students.

#### Categorisation:

The requirements are split up into three categories (game design, non-essential and general user requirements). The game design requirements specifically affect the behaviour of the new system (i.e. what it should do) whereas the general requirements do not. The non-essential requirements are those which the client has specified as being useful but not a necessity for solving the problem. Brief reasoning of each requirement for the system are also included, abbreviated from the discussions with the client and other users.

#### Game Design Requirements:

1. An interactive system that engages the students in the topic. This involves being able to input into the system to change in-game properties.
2. A game that goes into more advanced orbital mechanics above the scope of the A level physics syllabus. This is to give the students an idea of higher education in physics such as at university. The game should use the equations derived from Kepler’s laws of planetary motion to determine the orbits of planets around stars as concluded in the *Approaches to Solve Problem* section*.*
3. Should include the game features specified in the appropriately named section on page X. These features should be interesting and fun for the students which should help them learn and understand the content easier.
4. The game would be produced in the three dimensions. This would allow the user to have interactions in a more complex and interesting environment.
5. There should be a task or challenge for the game to keep students interested for longer.
6. The user (teacher and students) can alter the interval at which game time progresses. By this allowing the user to change the “speed” at which planets orbit a star and everything else in the game world. This would then also include “freezing” the game at different points in orbit allowing for easier demonstrations.
7. To influence the educational aspect of the game, different modes are required. One mode would have the full aspects of the game. A demonstration mode would be used to simplify the simulation and such make it easier for use within the lesson.
8. Randomly generated solar system. This is to make it different for each use and keep interest.
9. In demonstration mode the user can create their own solar system. This is for use of displaying specific orbits in class.
10. Hints explaining briefly what terminology used means. This to further influence the educational aspect.
11. Values of the calculations being carried out and other information. This is so the students can understand the process of calculating the orbits or use the information for their own calculations and make predictions of what will occur.
12. A scoring system to accompany the challenge of the game.
13. A simplistic user-friendly graphical user interface due to ranging skill level of users for a computer system.
14. A setting to use different textures, this could help in rendering the game on lower end computer systems.

#### Non-Essential Requirements:

1. A multiplayer element to the game that would allow different players to connect with each other.
2. A leader board to accompany the multiplayer aspect to record scores of the users. (dependent upon one above)
3. The ability to play the game in both offline and online modes. (dependent upon two above otherwise only offline required)
4. Implement the Roche limit.

#### General User requirements:

1. Be able to run on the school computer systems. The school systems use windows 7 and have a GPU. If the school systems couldn’t run the program, it would be difficult for the user (students and teachers alike) to have access to the game.
2. Be able to be used on multiple operating systems so that the student can gain access to the resource from home. This is so they can use the tool for revision.
3. A game that could run on the majority of lower end computer systems. This would be the case for the cheap school computer systems. This would allow access to more users.

### Hardware & Software Requirements:

Some of these requirements are discussed in more depth in the limitations section. The requirements shown are for the windows 7 computer systems as that is the primary machine the solution is targeted at.

#### Hardware requirements:

* GPU
* Hard disk space: 50MB overestimation, look at design stage for more accurate value.
* Ram: 1 GB for x86 / 2 GB for x64 processors
* CPU: 1 GHz single core x86/x64 processer
* Keyboard and mouse
* Monitor/display
* Network Interface Card (for using network features)

#### Software requirements:

* Operating system (windows 7 for school systems, Linux systems also supported)
* C++ compiler
* GPU Drivers (includes modern OpenGL: versions 4. 0 +)

### Game features:

These are similar to the user requirements but are the features which describe what the game is about and what you can do within the game. These are also determined by the user (client and other stakeholders). Look at appendix 3 for research methods and about the numbering system.

#### Essential Features:

1. The game would simulate a solar system.
2. The user starts of orbiting a random planet.
3. The solar system has a random number of planets.
4. The user controls a vessel (i.e. a spaceship).
5. The user changes orbits to go from one large mass to the next.
6. Random start points.

#### Non-Essential Features:

1. Within the solar system there are also a number of comets.
2. Each planet would have a number of moons.

### Limitations:

There are many limitations within the project. These are due to user preference, whether complexity is too great due to the timeframe or processing power being not adequate. These are summarised below however more are described in detail in the next paragraph.

#### User Preference:

* The spin of the objects can be ignored as it doesn’t affect the orbital motion.
* The objects modelled are limited to that within a solar system. This excludes some bodies such as black holes from the model.

#### Complexity and Performance:

* Collisions of objects to be ignored.
* Simplistic textures for faster runtimes (if used at all).
* Only a single solar system due to the extra mechanisms required too complex to simulate transfer of objects from one solar system to another.
* The loss of energy due to space dust can be ignored because of added complexity and wouldn’t make the orbit a closed loop.

### Limitations and Workarounds:

#### A Computer System:

The most obvious limitation would have to be the availability of a computer system. Without a computer system at hand, the program wouldn’t be reachable and so the user couldn’t use the system.

#### A Graphics Processing Unit (GPU):

The approach chosen would require the use of the GPU because of the significantly reduced amount of time to render objects within the game. There are many concerns with this which include:

* Many of the same calculations would be carried out upon the objects loaded. Complex objects would be represented using many vertices and requires each to be translated at the same time using calculations.
* For the game to run in real time, the central processing unit (CPU) may not be able to carry out enough computations within a set time to be possible. Each computer system would have different hardware specifications and the range of CPU processing power would be significant in being able to run the game at an acceptable frame rate.
* A GPU can carry out computations much more quickly, when they are in parallel, over a number of CPU cores.
* For rendering graphics, parallelisation is very high. This is because of the many vertices used to represent objects and the large number of pixels for the display.
* The game would depend upon a significant amount of physics and hence calculations. Some of these calculations would be carried out continuously for the existence of an object such as a planet orbiting a star. For this example, there would be many vertices representing points on the surface of the planet. Each vertex would contain the information required so that it can be drawn to the screen. This would include the colour, the co-ordinates, etc. For the sphere (planet) there would be many vertices required to represent it and would all be transformed using the majority of the same calculations.
* Using a GPU makes it more difficult to run within a virtual machine as they don’t often come with support for using the GPU.
* Another issue is that the computer systems without a GPU wouldn’t be able to run the game. This isn’t much of a concern as nearly all modern computer systems have a GPU.
* OpenGL uses the GPU to easily render graphics for three dimensional objects wanting to be displayed. Hardware acceleration significantly improves the performance of graphics rendering.

#### Alternative Without a GPU:

A work around can be achieved using software rendering to run OpenGL executables. This is an option for systems without a GPU or without drivers available to run the more recent versions of OpenGL (i.e. 4. \*). This can be achieved using Mesa 3D for example. Although the programs can still be run, the frame rate would be significantly reduced as the CPU is used to render the graphics over the GPU.

#### A C++ Compiler:

* The program produced will be written in C++. This is a compiled language and hence requires a compiler to translate the high level source code into object code. This means a compiler would need to be shipped with the program for it to be run. This would mean a larger file size as well as taking a considerable longer time to run as it needs to be compiled first by the user (the time of installation).
* It only needs to be compiled once so the executable can be run thereafter. This then means that the compiler is no longer needed and hence reducing the file size. This is considerably faster than the alternate approach of using an interpreter. For large programs (like this game would be), it is better to compile as executable files run faster than interpreted program files. This is because the executable is already translated whereas the interpreter has to translate each line in the program before it is executed every time the user wants to run it. This would considerably slower performance. This is especially the case in producing graphics as different commands would need to be processed for every frame. An interpreter would bottleneck this.
* Another solution would be to produce many different versions of the program that are precompiled before being shipped. This is inefficient as there would need to be many different versions produced for every type of computer system dependent upon hardware and software specification.

#### Drivers for OpenGL:

* The latest drivers are required to use OpenGL. This is because there are many different versions of OpenGL. The latest versions (i.e. 4.0 +) have increased capabilities. It has more features and more improved and efficient features as well as bug fixes.
* Modern OpenGL is significantly different to the older versions. They have improved many features with this. The older versions are supported by modern OpenGL and such allows for backwards compatibility. The issue is that the supported features for the older versions of OpenGL are inefficient. This is why the newer features would be used that is not compatible with the older OpenGL versions.
* As new versions are released, the GPU drivers need to be updated such that the newer version features can be supported. The developers for older GPUs may no longer update their drivers as they are becoming outdated. This causes some older GPUs to not support the latest versions of OpenGL and hence run the game being developed.

### The success Criteria:

There are two aspects to the success criteria. These would be whether the requirement has been completed as well as the efficiency and standard at which it is met.

The success criteria and user requirements are considered to be one of the same thing. The user requirements specified are then used as the success criteria. The user, test groups and stakeholders then determine whether the success criteria have been met.

A simple draft table for meeting the success criteria is shown in appendix 5.

#### Completed Requirements:

This would be assessed by the end user and by testing groups at different stages of the project lifecycle. Once all the requirements are met, the project is completed and is a success. If not all requirements are able to be met within the timeframe, the project will be considered a success if at least the necessary requirements are met. The necessary requirements are determined by the user (stakeholders) at the beginning of the project and may change throughout the process. These will be chosen according to the importance to the user and whether a requirement is dependent upon another being achieved.

#### Efficiency and standard of Requirements:

To achieve this, two methods would be used. To assess the efficiency of the solution, testing is carried out to determine that an acceptably efficient method is chosen. This is also determined by assessing the code while being written to see that it is to an efficient standard. This is because the programmer should be able to understand how the code works and whether there is a better approach. An issue with this being that it would be easier to overlook a mistake if the same person assesses the code. To determine whether each requirement met is user friendly, testing is carried out by the user.

#### Scoring the requirements:

The success criteria are scored by the user. If the score is high enough it would suggest the requirement has been met to a high enough standard. If the score is not high enough, the requirement is reworked until the user finds it is or the fixed timeframe has been reached. The timeframe is considered fixed due to the limited time that is available until the project deadline is reached for the A Level programming project to be submitted for grading by OCR.

#### Comparison with My Solar System:

To make certain the new simulator is an improvement to the old one, they can be compared. This would be done through research from students.

# Design

JUSTIFY DECOMPOSITION

JUSTIFY DECISIONS, only described so far i.e. testing.

### Objectives of game:

* Use of OpenGL to render the world to the screen. OpenGL has the capability to produce the 3D graphics expected in the user requirements.
* Use of SDL2 to create the application structure including the window the game runs in. SDL2 can be used to produce a graphical user interface. This should let the user have the desired interactivity stated in the user requirements.
* A leader board that shows the user the top in the world against their highest score. This would be implemented through linking the program to a database server that simply stores the highest score. This would need to use a primary key (quite possibly the device IP address or mac address) to recognise a user. This is so it can update a specific user’s score after playing the game again. The leader board scores have to be held on a database server to store the information on each user. The server would also be needed in connecting to separate users not on LAN.
* Another feature would be to introduce multiplayer. Most likely peer to peer networking would be used to allow for users to play with or against each other without requiring heavy server processing.
* The game would be produced in the three dimensions. This would allow the user to have interactions in a more complex environment (which should then be more interesting).

## Overall System Design



### Input Process Storage Output:

#### Demonstration Mode:

|  |  |  |  |
| --- | --- | --- | --- |
| Inputs | Processes | Outputs | Storage |
| Initial body properties (i.e. mass, a, b, etc.).  User pre-set model selection.  Pre-set database data. |  |  | xxxxxxxxx |

#### Game Mode:

|  |  |  |  |
| --- | --- | --- | --- |
| Inputs | Processes | Outputs | Storage |
|  | Simulating spacecraft around orbit. | User scores to display.  User scores to database. |  |

#### Both Modes:

|  |  |  |  |
| --- | --- | --- | --- |
| Inputs | Processes | Outputs | Storage |
| User settings (i.e. rendering quality).  Game speed (i.e. freeze and reduce). | Simulating the position of bodies within the solar system. | Graphics being displayed (i.e. solar system model). |  |

## Usability Features

## Definition of Record Structure

## Modular Design

The resource will be designed using modular programming. The program is split into many modules. Each module will have a unique purpose. Each module will have separate files. Each module is a class or the main function. Each class module will have two files, a header file and a source file.

### Object Oriented Design:

Use Cases & Object models

Class inheritance diagrams

Interactions between objects

The approach will be object oriented as mentioned within the analysis. The user defined classes to be used for this are listed below.

### Class Inheritance:

There is only little class inheritance. This is the inherited properties of the star class. The star class is used to only store the properties of the star and so has no methods. As the star is the only body that doesn’t undergo orbital motion (as everything else is orbiting it) it doesn’t require the attributes of all the remaining bodies. All bodies have certain properties such as mass, radius, etc. The remaining bodies (i.e. the body class) therefore inherits the attributes of the star class as well as having additional attributes that the star class doesn’t have. There are also the inherited properties of the game class from the simulation class.

### Application class:

Handles the application as a whole. Includes taking user inputs and then making manipulations to the display.

#### Attributes/Variables:

#### Methods:

* User input

### GUI class:

Handles many features of making the graphical user interfaces.

#### Attributes/Variables:

|  |  |  |  |
| --- | --- | --- | --- |
| Name/identifier | Description | Data type | Access specifier |
|  |  |  |  |

#### Methods:

### Simulation Class:

### Game Class:

### Player Class:

### Graphics class:

The graphics class handles the operations required to display the graphics. This mainly includes the operations carried using the OpenGL library.

#### Attributes/Variables:

#### Methods:

|  |  |  |  |
| --- | --- | --- | --- |
| Name/identifier | Description | Data type | Access specifier |
|  |  |  | Private |

### Star class:

This class used to store the properties of the star and has no methods as described in the *Class Inheritance* section.

#### Attribute/Variables:

|  |  |  |  |
| --- | --- | --- | --- |
| Name/identifier | Description | Data type | Access specifier |
| Mass (m) | Stores the mass of the body. | Double | Private |
| Radius | Stores the radius of the body.  (Assumes a spherical body) | Double | Private |
| Body type (name) | Stores the type of texture to be used. The enumerator also distinguishes the type of body. | Enumerator | Private |

### Body class:

A body refers to, in this circumstance, an entity in the solar system model such as planets, moons and comets. For this class it excludes the star. The inherited attributes are also listed. Compare the attributes of the star class and the body class to determine the inherited attributes.

If the identifier going to be used in the program is not the same as its name, the identifier is listed in brackets.

Look at orbit diagram in appendix X that shows what many of the attributes represent.

Many of the identifiers are determined from the symbols used in mathematical equations.

#### Attributes/Variables:

(These are the initial attributes required by the object before calculations can be carried out)

|  |  |  |  |
| --- | --- | --- | --- |
| Name/identifier | Description | Data type | Access specifier |
| Mass (m) | Stores the mass of the body. | Double | Private |
| Radius | Stores the radius of the body.  (Assumes a spherical body) | Double | Private |
| Semi major axis of orbit (a) | Stores the value for this distance. | Double | Private |
| Semi minor axis of orbit (b) | Stores the value for this distance. | Double | Private |
| Z-plane angle (thi) | Stores the value of this angle. | Float | Private |
| Body type (name) | Stores the type of texture to be used. The enumerator also distinguishes the type of body. | Enumerator | Private |

(These attributes are calculated once from the initial attributes for the object lifetime)

|  |  |  |  |
| --- | --- | --- | --- |
| Name/identifier | Description | Data type | Access specifier |
| Time period of orbit (P) | Stores the time it takes for the body to complete one orbit. | Float | Private |
| Eccentricity (e) | Stores how elliptical the orbit actually is. Calculated from a and b. | Float | Private |
| Mean motion (n) | Stores the mean motion of the body. | Double | private |

(These attributes are calculated from the other attributes many times during the lifetime of the object)

|  |  |  |  |
| --- | --- | --- | --- |
| Name/identifier | Description | Data type | Access specifier |
| Spherical coordinates (spherCoords) | Stores the relative coordinates of the body. Made up of polar coordinates to be calculated and z-plane angle as constant. | 3-dimensional vector | Private |

#### Methods:

|  |  |  |  |
| --- | --- | --- | --- |
| Identifier | Description | Parameters | Access specifier |
| prelCalculations | Carries out preliminary calculations. This includes computing values for the calculated once attributes. | Those listed in initial attributes table | Public |
| calcPolarCoords | Calculates the relative polar coordinates Uses the functions in physics class to get here. | a, b, m | Public |
| calcRelPos | Calculates the relative position of the body with that it is orbiting. | polarcoords and z-plane angle | Public |
|  |  |  |  |
|  |  |  |  |

### Physics class:

Handles all physics and maths calculations for the simulation. Separated into one location for easier use and manipulation.

#### Attributes/Variables:

|  |  |  |  |
| --- | --- | --- | --- |
| Name/identifier | Description | Data type | Access specifier |
| Mean motion (n) |  | n, M, ms | Private |
| Mean anomaly (M) |  |  |  |
| Eccentric anomaly (E) |  |  |  |
| True anomaly (theta) |  |  |  |
|  |  |  |  |

#### Methods:

|  |  |  |  |
| --- | --- | --- | --- |
| Identifier | Description | Parameters | Access specifier |
| calculate\_n | Calculates the mean motion. Returns n. | bigMass, m, a | public |
| calculate\_e | Calculates the eccentricity. Returns e. | a, b | public |
| calculate\_M | Calculates the mean anomaly. Returns M. | n, t | Public |
| calculate\_E | Calculates Eccentric anomaly. Returns E. | e, M, x | Public |
| calculate\_theta | Calculates the true anomaly. Returns theta. | E, e | Public |
| calculate\_r | Calculates the heliocentric distance. Returns r. | E, e, a | public |
|  |  |  |  |

## Data Dictionary

Approximate size of program

## Data validation, security and integrity

### Data Integrity:

### Data Security:

### Data Validation:

Validation is required for user input. This is because errors could arise if certain inputs are made. This could lead to corruption of data or crashes.

## File Organisation & Processing

## Database Design

## Sample of planned SQL queries

## Identification of Storage Media

It is estimated that the program will require 10MB to be stored. This only includes the raw program and excludes any textures or images. The program is going to be run on the school computer systems as mentioned in the *analysis* section. This would likely be during lessons in the computer rooms where each student can use the program or in class by only the teacher (i.e. client). Every classroom has a computer system for the teacher to use. The program will be accessible on the school network distributed drive called *RMshared*. Each student can then access the program from any school machine. For access from home, the student can make a copy of the program and run it from home.

## Detailed Description on Implementing Kepler’s Laws

## Algorithms

### Selecting Algorithms for Solving Kepler’s Equation:

As stated elsewhere, the eccentric anomaly *E* is calculated through iteration. This is essential as the angle arrived at is needed to calculate the true anomaly (part of the spherical coordinate system). The equation, known as Kepler’s equations, to be solved is:

Rearranging for *E* is not possible so a numerical method would have to be used. This is referring to repeatedly computing an estimate of the value of *E* until it is of a high enough accuracy (i.e. decimal places) to be used to calculate the true anomaly. There are many considerations that have to be made when deciding the most efficient numerical method to implement. The method needs to be considered as the algorithm produced will be carried out every frame on every body in the model and hence requires to be as efficient as possible because of the significant impact on runtime.

#### Factors affecting the efficiency of method:

* The number of iterations required to get to an accurate solution. Different numerical methods converge to the actual value at different rates so require more or less iterations.
* The number of arithmetic operations needed per iteration. This being obvious that the more operations carried out, the longer it will take.
* The type of arithmetic operations being carried out. This is because different operations require more or less processing than one another. For example, multiplying and adding is usually significantly faster than division and division is significantly faster than calculations with trigonometric functions.

#### The Newton-Raphson Method:

The Newton-Raphson iterative formula is listed below:

By rearranging Kepler’s equations to be a function in terms of *E* leads to as follows:

1. (*where M & e are constant*)

Substituting these equations into the iterative formula gives:

An algorithm that simply iterates, or uses recursion, of the above equation can now be implemented.

One issue with this method being that *ecosE* cannot equal 1. This is because division by zero is undefined. This wouldn’t occur often yet is possible. One example of this would be if the initial estimate of *E* () is equal to *0* (in radians) and e is equal to *1*. This would cause an error.

#### A Simple Fixed Point Method:

Kepler’s equation can be rearranged to give:

This can be put into an iterative format as follows:

This can then be used to estimate a value for *E.* Even though this formula is simpler, it would usually require more iterations to get to an accurate estimate of *E* when compared with Newton’s method.

#### Other Potential Numerical Methods:

There are a large number of numerical methods that can be used to solve the equation. Similar to the methods described above, they have different advantages and drawbacks. Other possibilities include the bisection and false position methods.

#### Example Algorithm:

Listed below is a simple pseudocode to implement a numerical method.

*Define function called E (e, M, x) // Where x is the number of iterations*

*E = M // Let the initial estimate be M*

*For i=1 to x*

*/////////// (iterative equation) ////////// (i.e. E = M + e\*sin(E))*

*Next i*

*Return E*

#### Testing for Most Efficient Method:

The best numerical method is dependent upon many factors and may vary depending upon the input. It is very difficult to predict at this stage which numerical method would be most efficient so a specific algorithm shouldn’t be selected yet. A description on determining the best method is discussed in the *Test Strategy* section.

## User Interface Design

The game runs in a single window. Within the window, there are a number of user interfaces.

### Navigation diagram:

Below is a diagram showing the relationship between the screens the user can get to in the program.

C:\Users\Gregory\AppData\Local\Microsoft\Windows\INetCacheContent.Word\modular design pages.png

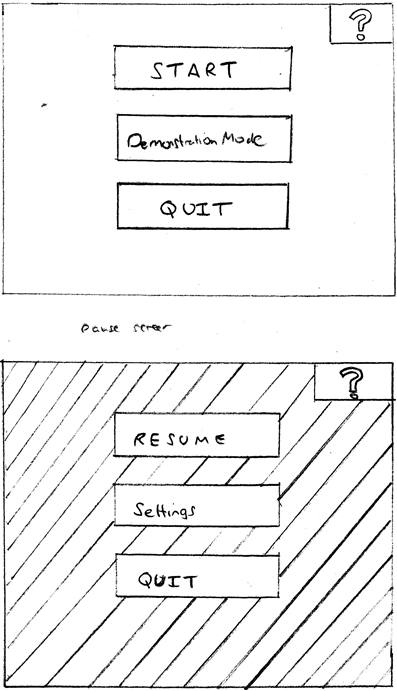
Add help screen and scoreboard above

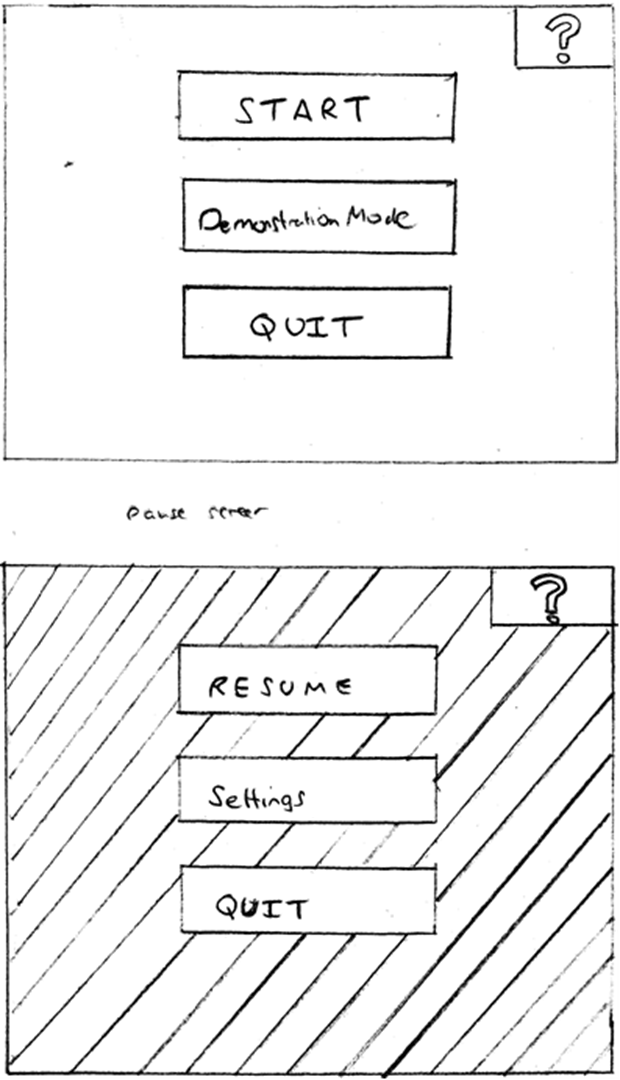
Score board design

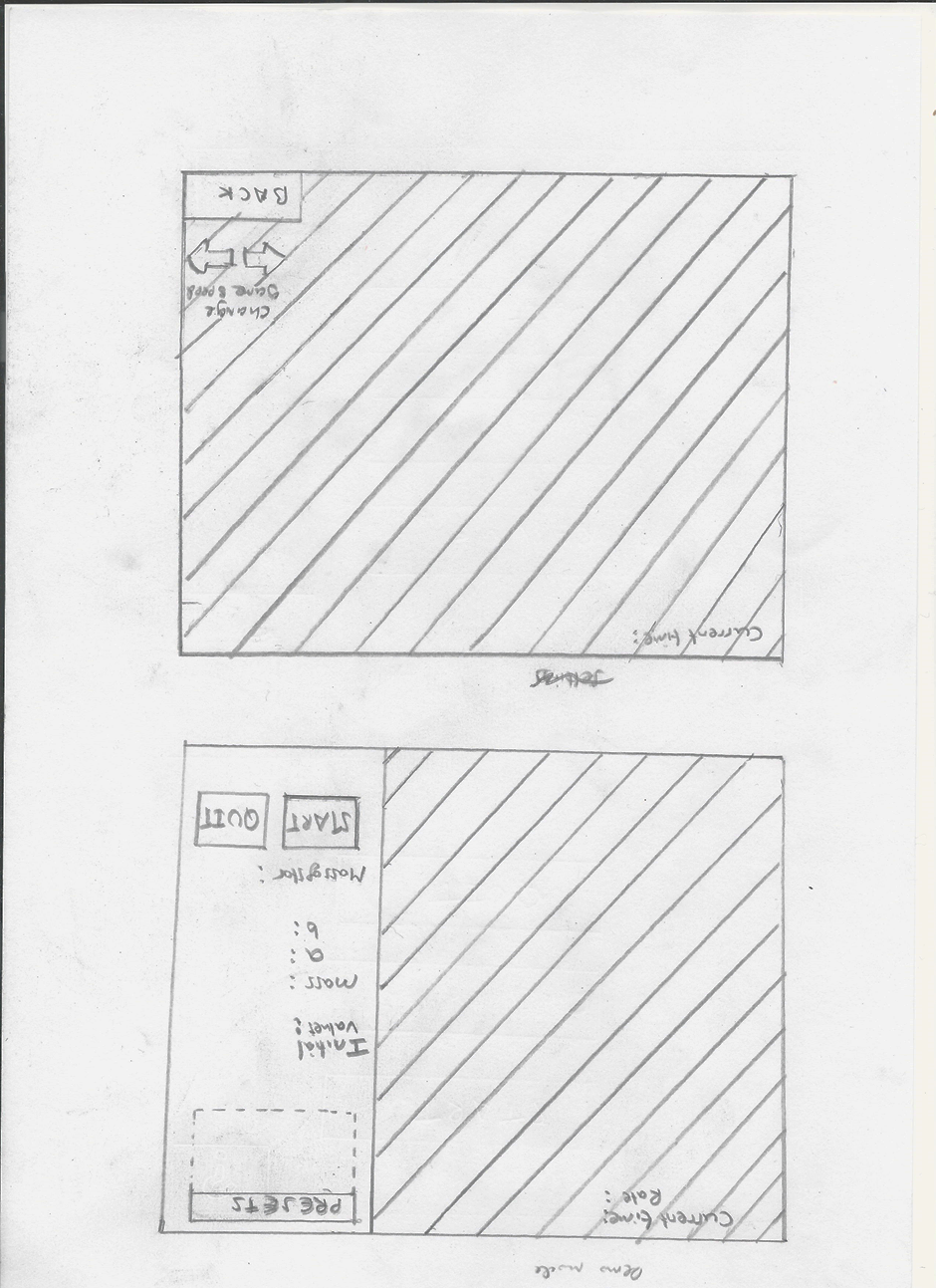
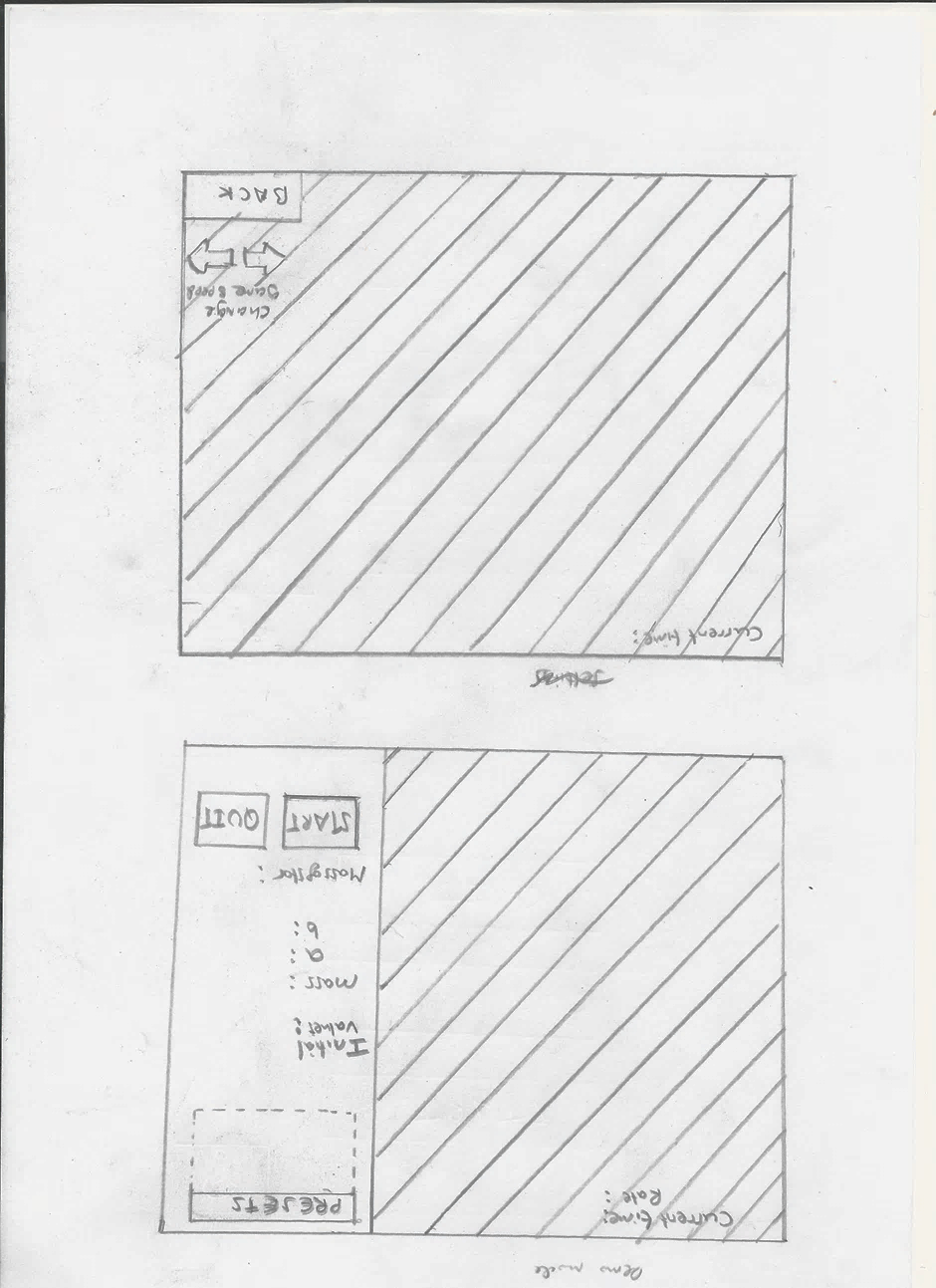
Help screen design

Settings design

### Start Menu interface:





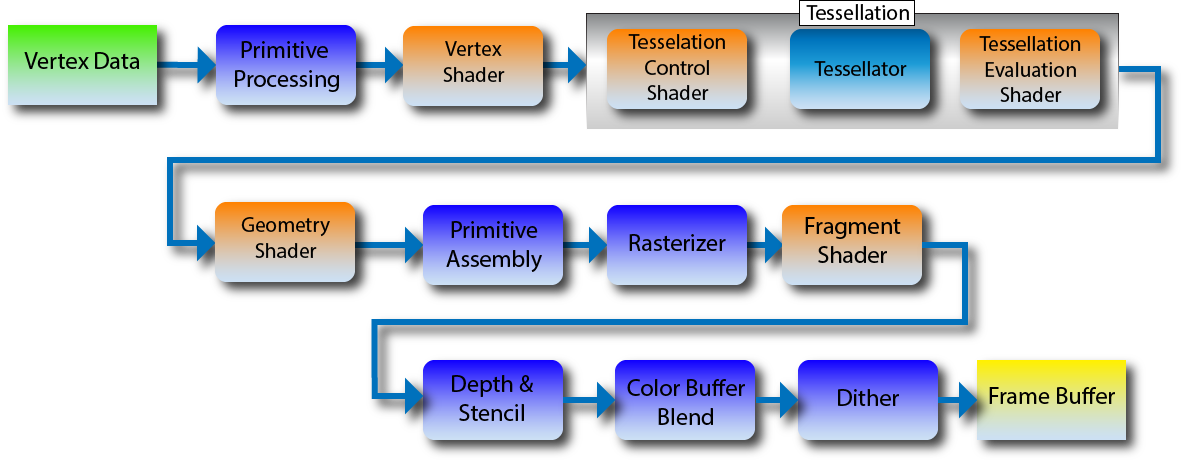


## The OpenGL Pipeline:

OpenGL uses a pipeline that decomposes the graphics processing into manageable stages. Many of the stages are fixed and cannot be altered (and automatic). Some stages (the shaders) can be altered. These stages are changed depending upon the complexity and features of the graphics being produced.

#### OpenGL Pipeline Diagram:

This diagram shows all the stages of the OpenGL pipeline.



<http://www.opengl2go.net/wp-content/uploads/2015/11/OpenGL-4.0-Programmable-Shader-Pipeline1.png>

#### Alterable stages:

* The vertex shader
* The tessellation shaders (control & evaluation)
* The geometry shader
* The fragment shader

#### Shaders:

A shader is a small computer program that is run on the graphics processing unit instead of using the central processing unit. In OpenGL, shader code can be written in GLSL (OpenGL Shading Language).

#### Stages to be altered:

Only the vertex shader and fragment shader are to be altered for the program. This is because there is a limited time frame and the added complexity would make it more difficult to program. Also the other shaders do not make the program more efficient for simpler, lower-end graphics that is expected by the client. If there is time during the development stages, the additional shaders may be used. The vertex and fragment shaders are essential in producing graphics.

#### Shader Programs:

Shader programs are considered to be the combination of the shaders at each stage of the pipeline into a single program.

#### Shader Program Diagram:

This diagram shows the steps necessary to implement shaders within the C++ program. There are appropriately named OpenGL functions for this. For each type of shader, functions are carried out to create the shader. The shader then has to appropriately sourced. This is referring to the memory locations and formatting of the GLSL code held describing the shader program. The shader is then compiled. After this, the shader program is created. The shaders are then attached to the program. The shaders can now be linked to the program. The program can now be used. This diagram explains the shader process well and so algorithms on this process will not be produced, or at least not in as much detail.

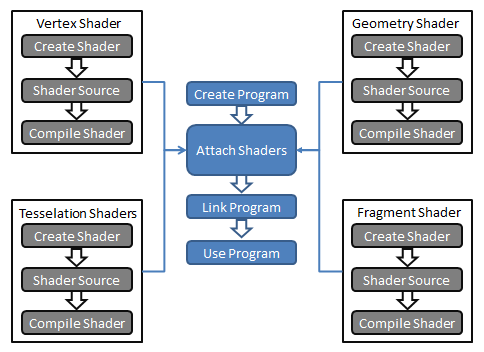
#### Vertex Shader:

The vertex shader is the first programmable stage of the OpenGL pipeline. This shader is provided the vertex attribute data from the draw command written in the C++ code. The vertices are the position of important points needed to draw shapes. They also include other properties such as the colour at that point. For examples the three vertices of a triangle. The vertex shader is run once for each vertex provided to it. The vertex shader carries out calculations and passes the vertex down the pipeline.

#### Fragment Shader:

The fragment shader is carried out once per fragment. A fragment is all the data necessary to draw a single pixel and is also used to determine whether that pixel should be drawn. The fragment shader determines the colour of the pixels to be displayed.

#### Shader Program Design:



<http://www.lighthouse3d.com/wp-content/uploads/2011/06/glslprogram.png>

## Test Strategies

Black/white box testing. Destructive testing. Trace tables. Alpha and Beta testing? One other I remember not sure yet.

Test for best numerical method. Compare Newton-Raphson method with others. Test efficiency/speed.

For each test

### Stages of Testing:

Some testing will be carried out in a particular order. This is because it is a waste of time to carry out certain tests when others fail. For example, white box testing shouldn’t be used to test the efficiency of a function when that function doesn’t return the correct values in black box testing or uncertain whether it does.

#### Categorisation of Testing:

1. Unit Testing
2. Integration Testing
3. System Testing
4. Validation Testing

### Unit Testing:

A unit is to be considered a method for each class or a small section of code.

### Integration Testing

This is where multiple units are ‘integrated’ or combined to test how they work together.

### System Testing

This is where testing is carried out on a functioning machine.

### Validation Testing

This is testing where

### Testing Numerical Method Algorithms (System Testing):

To decide which numerical methods should be used, performance testing is to be carried out. To implement each algorithm requires little code. This means each method runtimes can be compared without spending significant amounts of time coding. It is assumed that the relative speed of each type of arithmetic operation is similar upon different machines, this means that testing each algorithm runtimes can be done on a single machine and the pattern can be assumed for all machines. Each algorithm will be tested on their own as an independent piece of code. A test for each algorithm is carried out a large number of times and averages taken. For each algorithm, different values of *M* and *e* are inputted for a set number of iterations. The accuracy of the outputted value for *E* is also recorded. All the values are recorded on a table. The choice of inputs is determined using boundary testing. It will be also calculated and recorded the values that would return an error. An alternative method would then be used for those values if that method is selected. The algorithm with the fastest and most accurate value for E is the most efficient. It is likely an algorithm would be faster for a range of inputs over another. This means a selection of algorithms would be used depending upon the value of *e* and *M.*

### GUI Testing:

### Dynamic Testing:

### Static Testing:

### Black Box Testing:

Black box testing will be used on each function within each class. The desired output of each function is known. A range of values are inputted into each function. If the expected output is received for each input then the function has passed the test, else it fails and needs to be reworked. This test is used to determine if each function is working correctly. A table will be produced recording whether each function has passed or failed for given inputs. This type of testing can only be used for functions; it isn’t used for the procedures for each class as they do not return a value.

#### Type of Inputs and expected Outputs:

table

### White Box Testing:

#### Dry Run (Static) Testing:

### Destructive Testing:

This test method will involve trying to ‘break’ the program. This is referring to attempting to make it crash or behave unexpectedly. There are different aspects where this testing will be used. Destructive testing will be used to test the user inputs. The user can enter inputs using the graphical user interface. Each input is tested to see the limit at which the system would break. Below is a list of test data that would be collected for the GUI.

### GUI Tested Inputs: (System Testing)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Type of Input & Description | Type of Testing | Expected inputs | Expected Values | Ways to break System |
| Button Press | Destructive |  |  |  |
| Button Press | Destructive |  |  |  |
| Mouse movement | Destructive, boundary |  |  | Move mouse as quickly as possible back and forth.  Keep moving mouse in one direction as quickly as possible. |
| Mouse Click | Destructive, boundary |  |  |  |
| Keyboard press |  |  |  |  |
| Textbox input |  |  |  |  |

### Acceptance Testing:

This is where the system is given to the client to see if all the user requirements have been met before completion. At this stage the *Success Criteria Table* illustrated in *Appendix 5* is given to the client. The client then looks and uses the software going through each requirement to see that it satisfies expectation. If it does, it has been completed otherwise it is reworked.

### Diagram Showing Testing Each Module:

This diagram appropriately shows the testing method for each class.

# Development

# Evaluation

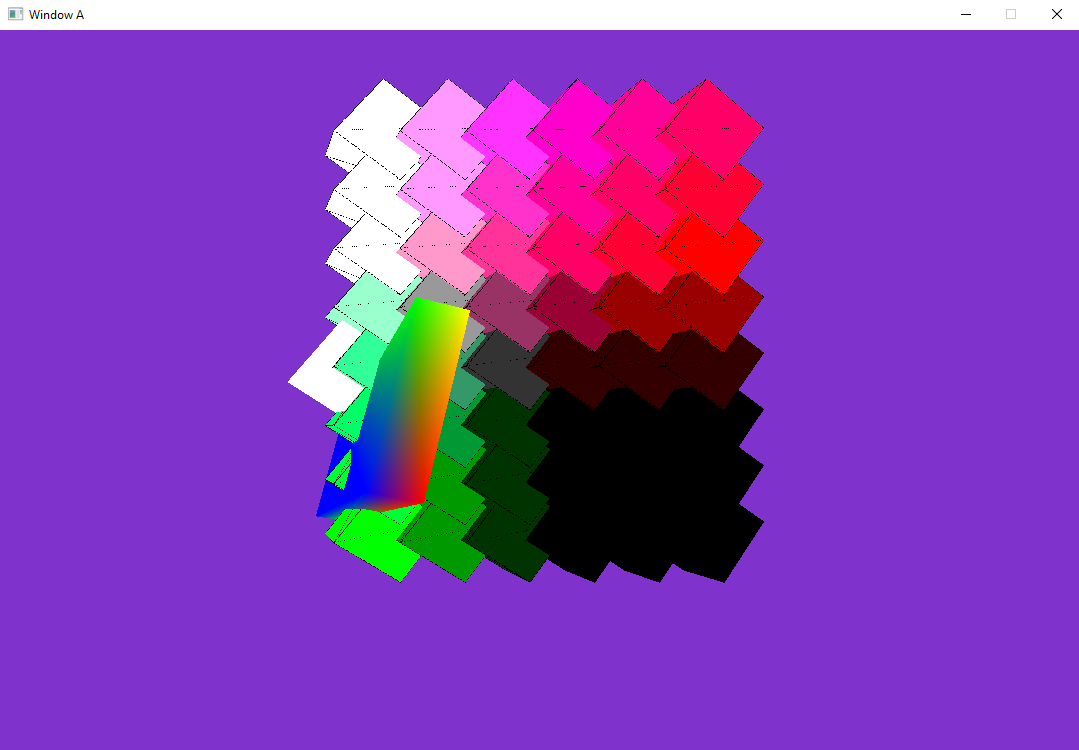
# Appendices

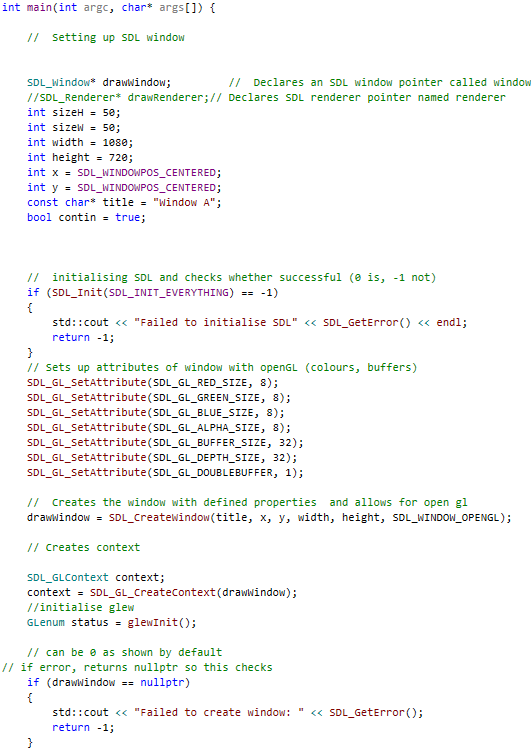
## Appendix 1:

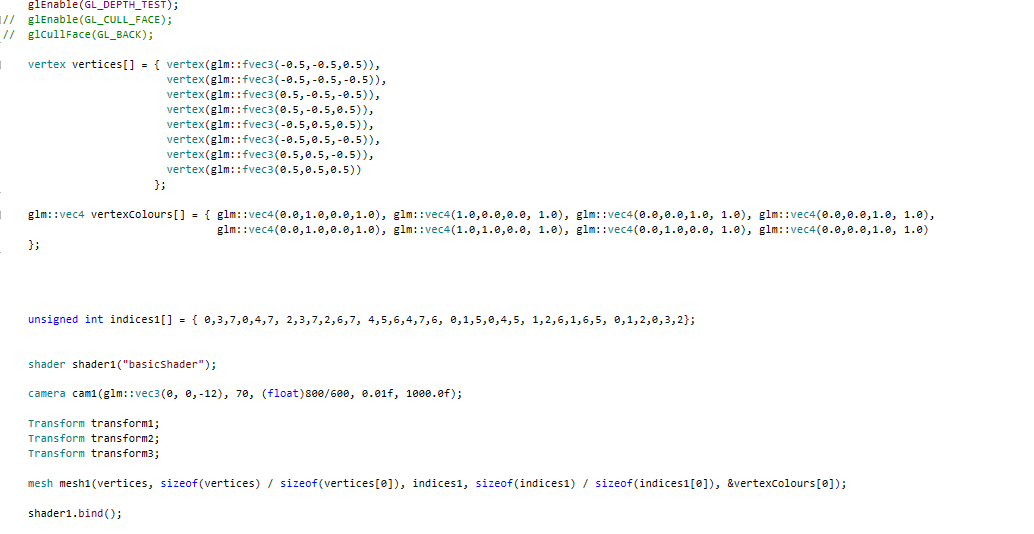
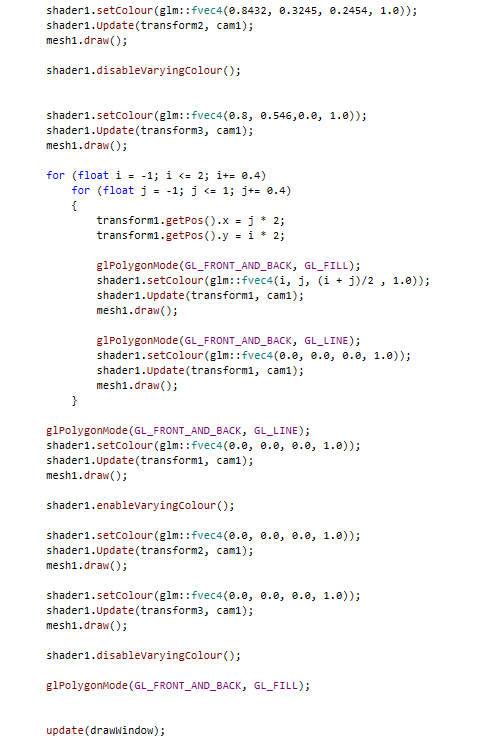
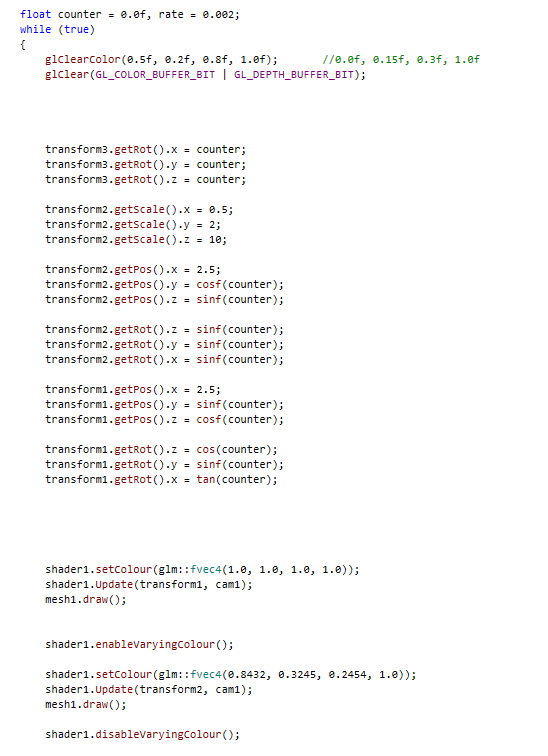
### OpenGL & SDL2 research sources:

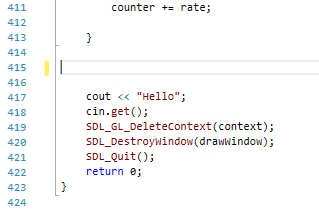
* [https://www.**youtube**.com/watch?v=ftiKrP3gW3k](https://www.youtube.com/watch?v=ftiKrP3gW3k) (over 4 hour video series – source of first video provided using OpenGL and SDL2).
* <http://headerphile.com/sdl2/opengl-part-1-sdl-opengl-awesome/> (word tutorial on OpenGL and SDL2).
* OpenGL SuperBible 7th Edition (a textbook on OpenGL).

### OpenGL & SDL2 tutorial program:

The tutorial shows a program that displays a number of rotating cubes and an oblong of differing colours on a purple background. The program code (written in C++) shows the main function and doesn’t include the code for the user defined classes.







## Appendix 2:

### My Solar System:

#### source:

<https://phet.colorado.edu/sims/my-solar-system/my-solar-system_en.html>

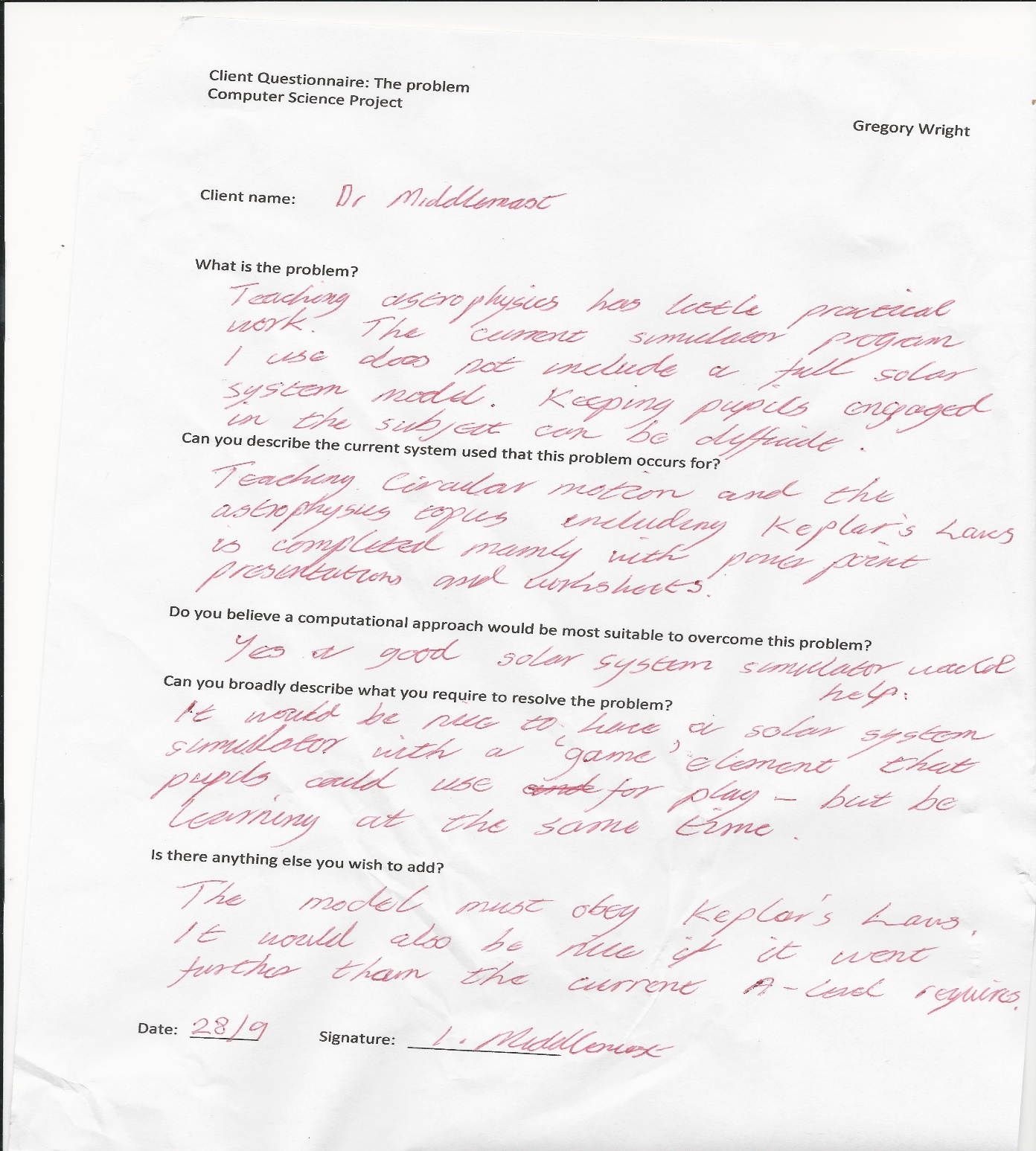
## Appendix 3:

### Numbering Systems:

The numbered points are used such that it is easier to match each aspect listed to the source that it came from. The numbering is **NOT** a ranking of priority. This is the case for both user requirements and game features.

### The Problem:

#### Client questionnaire:



### Stakeholders:

#### online Student questionnaire and statistics:

Due to the format of the questionnaire, instead of the Google Docs questionnaire being displayed, the questions that were asked are listed accordingly. There was a total of 33 responses to the questionnaire. This gives a large enough sample to give a representation of the students.

* *How enjoyable did you find learning circular motion in school? How enjoyable did you find learning circular motion in school? How enjoyable did you find learning circular motion in school?*

On a scale of 0 – 10 (10 being highest), 21 responses scored 7 or lower. This suggests that students do not find the topic enjoyable, or could be more enjoyable than it was.

The next set of questions were multiple choice. The options were: *None at all*, *only a tiny bit*, *some*, *lots*, *loads*. The values stated for each question are the number of responses given that were either some, lots or loads.On

* Did you require any further revision outside of school to understand any of the content in circular motion?*Did you require any further revision outside of school to understand any of the content in circular motion?* ***21***
* *Do you think that lessons could have been more interactive within class learning circular motion?Do you think that lessons could have been more interactive within class learning circular motion?* ***23***
* *Do you think that a solar system simulator could have made learning circular motion and orbital mechanics easier?* ***23***
* *Do you think that a space game could have made learning circular motion and orbital mechanics easier?Do you think that a space game could have made learning circular motion and orbital mechanics easier?* ***22***
* *Would you prefer learning about circular motion using a simulator and space game or from presentations?* ***25***

These responses suggest that there are improvements to the system and that the proposed simulator and game could achieve this.

The final question was a simple yes or no answer. 25 said yes.

* *Do you think that a simplistic user interface/display should be used for the simulator?*

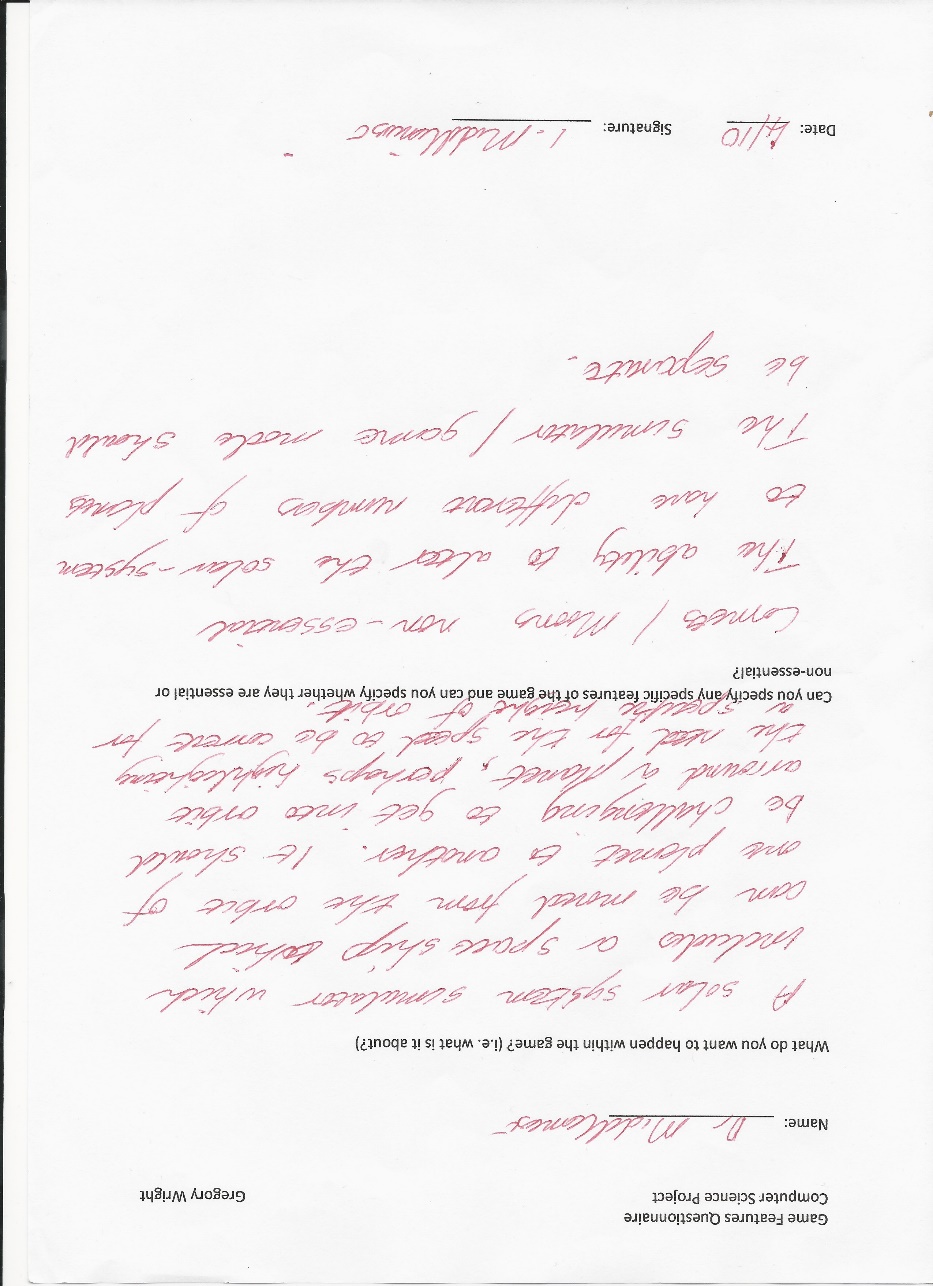
This tells me students prefer a display that isn’t overcomplicated.

### User Requirements:

The user requirements are determined by the client and other stakeholders. The information collected was through questionnaires and meeting with the client and stakeholders. Other research was used to help determine user requirements such as online research before approaching the client and stakeholders for further feedback. The information collected from the stakeholders were taken to the client and used to help determine further requirements.

#### First and Second Interview with Client:

#### Game Features Questionnaire:



## Appendix 4:

### Image references for Kepler’s Laws Research:

#### First Law:

<http://astronomer.wpengine.netdna-cdn.com/wp-content/uploads/2013/06/kepler1.gif>

#### Second Law:

<https://eloisechen.files.wordpress.com/2013/02/kepler2law.jpg?w=1600&h=1200>

#### Third Law:

<http://m.teachastronomy.com/astropediaimages/KeplerThirdLaw.jpg>

## Appendix 5:

### Success Criteria Table:

The table below shows a draft version of a form for the client to score the success criteria. It includes many fields on ways each requirement can be improved if not fully met. This table would primarily be used for user-friendliness as the efficiency of code is difficult to determine by the end-user. Also for determining whether the requirement met is what was expected by the client. It includes a field to indicate the efficiency of the requirement to the client.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| User Requirement | Ready to be checked by client | Programmer efficiency score | Score by client | Reasoning for score | Is anymore improvement necessary? | Ways to improve the requirement | Is there anything else to add? |
| … | … | … | … | … | … | … | … |

## Appendix 6:

### Module Algorithms:

Listed below are the algorithms for each module (i.e. class) produced during the design stage. The header files are excluded from each algorithm as they are a summary of the contents of each class which are already listed in the *Design* section.

#### The Main Function:

Pseudocode algorithm showing the workings of the solution within the main function. The pseudocode uses the appropriate method names of the objects rather than explaining what it will do. The method names should be straight forward enough but a description of what they do can be found in the *Design* section.



#### The Application Class:

#### The Game Class:

#### The Simulator Class:

#### The Graphics Class:

#### The Physics Class:

#### The Body Class: