



DO PROCEDURALLY GENERATED SPACE-CRAFTS FOR THE GAME DEMO "GALACTIC ARMS RACE" MAKE GAMEPLAY MORE ENJOYABLE?



UNIVERSITY OF
LINCOLN

By

Andrew Deathridge

DEA11209320

MComp Games Computing

The University of Lincoln

01/05/2014

DO PROCEDURALLY GENERATED SPACE-CRAFTS FOR THE GAME DEMO "GALACTIC ARMS RACE" MAKE GAMEPLAY MORE ENJOYABLE?

By

Andrew Deathridge

2014

Acknowledgements

I am absolutely grateful to my project supervisor, Dr. Patrick Dickinson. Without his advice and encouragement, it's fair to say that the quality of the project would have been nowhere near what it is now. I would like to take advantage of this opportunity and just say thank you for everything, I really appreciate all the help you gave to me and I will not forget it.

I am indebted to Erin Hastings and Kenneth O'Stanley who gave me the opportunity to use Galactic Arms Race for my project. This game is their own "baby" which they have won a best paper award at the IEEE. The fact that they let me try my procedurally generated player avatars/ space-crafts was unexpected and I am still elated by the opportunity they gave. To both of you, thank you so much.

Finally, I would like to thank my family for the constant support they have given me. The times where I have been up till four in the morning and they ask me if "I'm ok" and tell me "not too worry" when they have work in the morning have always stayed in my mind. The endless love that they have shared with me has always given me the motivation and the will to do my tasks to the highest capability possible. To my mother, father and brother, I love you all!

Abstract

The aim of the project is to see whether procedurally generated space-crafts or ships in Galactic Arms Race will make the experience of it more enjoyable as a game. Two versions of GAR have been made to show if this use of PCG works on its own, or if it still depends on another aspect such as randomly generated ship stats. A questionnaire was also made to test how players acted with their ships and the opinions of the participants on their space-crafts. The results eventually show that PCG does work with player avatars in games (such as GAR) but, the stat changes are vital for the players to believe that the ship they have is completely different.

Table of Contents

Acknowledgements.....	2
Abstract	3
Figures	7
1 Introduction	9
2 Literature Review	11
2.1 Introduction	11
2.2 Galactic Arms Race	11
2.2.1 What is Procedural Generation?	12
2.2.2 Uses of Procedural generation in games	13
2.3 Game enjoyment.....	14
2.3.1 What is Game Enjoyment?	14
2.3.2 What makes a game enjoyable?	14
2.4 Identify with the player Sprite	16
2.5 Procedurally Generated player sprites	16
2.6 Will different player sprites effect game enjoyment?	17
3 Methodology	18
3.1 Introduction	18
3.2 Aims and objectives	18
3.3 Project Management Methodologies	19
3.3.1 Types of methodologies and their cycles.....	19
3.3.2 Waterfall.....	21
3.3.3 Agile.....	22
3.4 Risk assessment and contingency plans.....	23
3.5 Gantt chart	26
3.6 Milestones.....	27
3.7 Research methods.....	28
3.8 Tools used	29

4	Design.....	31
4.1	Introduction	31
4.2	The ships.....	31
4.2.1	Body parts	33
4.2.2	Wing parts	33
4.2.3	Head parts	33
4.2.4	Engine parts.....	33
4.2.5	Weapon parts.....	34
4.2.6	Tail Parts	34
4.3	Questionnaire.....	34
5	Implementation.....	37
5.1	Introduction	37
5.2	Implementing Galactic arms race version 2	37
5.3	Implementing Galactic arms race version 3	43
6	Evaluation.....	47
6.1	Introduction	47
6.2	The test procedure and its participants	47
6.3	The Experiment and its Results	48
6.3.1	Question 1 & 2.....	49
6.3.2	Questions 3 – 6.....	50
6.3.3	Questions 7 & 8	52
6.3.4	Questions 9 – 12.....	53
6.3.5	Questions 13 & 14	54
6.3.6	Questions 15 & 16	55
6.3.7	Code Metrics Version Two	56
6.3.8	Code Metrics Version Three.....	58
7	Conclusion	64
8	Critical Reflection	66

9 References..... 69

10 Appendices..... 72

 10.1 Appendix 1: Gantt Chart..... 72

 10.2 Appendix 2: Black-Box Testing Data..... 72

 10.3 Appendix 3: White-Box Testing Data 74

 10.4 Appendix 4: Questionnaire 75

 10.5 Appendix 5: Spreadsheet Data..... 79

Figures & Tables

Figure 1: Table showing the spiral Development (Skumar, 2014a)	20
Figure 2: Table showing the prototype development stages (Skumar, 2014b)	20
Figure 3: Table Showing the Waterfall Development Stages (Skumar, 2014c)	21
Figure 4: Table Showing the Agile Development Stages (Skumar, 2014d)	22
Figure 5: Gantt chart	26
Figure 6: An example of changing ship to body	37
Figure 7: The loadModel method in GAR	38
Figure 8: LoadModel examples	39
Figure 9: The globals which store the ship parts along with other A.I. ships	39
Figure 10: The first wing added into the list in FileManager	40
Figure 11: The ChangeBody method	41
Figure 12: Comparison of the ship code and the body part code that randomly picks model	42
Figure 13: Randomly generated ship (1)	42
Figure 14: Randomly generated ship (2)	43
Figure 15: Wing list in FileManager with random value for rotation	44
Figure 16: Random rotation in GAR version 3	45
Figure 17: Random stats 1	45
Figure 18: Random stats 2	46
Figure 19: Question One & Two Results - Bar Chart	50
Figure 20: Question Three & Four Results - Bar Chart	50
Figure 21: Question Five & Six Results - Bar Chart	51
Figure 22: Question Seven Results - Bar Chart	52
Figure 23: Question Eight Results - Line Chart	52
Figure 24: Question Nine & Ten Results - Bar Chart	53
Figure 25: Question Eleven & Twelve Results - Bar Chart	53
Figure 26: Question Thirteen Results - Bar Chart	54
Figure 27: Question Fourteen Results - Line Chart	54
Figure 28: Question Fifteen Results - Bar Chart	55
Figure 29: Question Sixteen Results - Line Chart	55
Figure 30: Models Generated 1	56
Figure 31: Models Generated 2	57
Figure 32: Ship Stats 1	57
Figure 33: Models Generated 3	58

Figure 34: Ship Stats 2 58

Figure 35: Base Values 1 59

Figure 36: Question One & Two Results - Bar Chart..... 60

Figure 37: Question Three & Four Results - Bar Chart 60

Figure 38: Models Generated 4..... 61

Figure 39: Ship Stats 3 61

Figure 40: Base Values 2 62

Figure 41: Question Nine & Ten Results - Bar Chart..... 62

Figure 42: Question Eleven & Twelve Results - Bar Chart..... 63

Table 1: Risk assessment table at its contingency plans 25

Table 2: Scaling the models 32

Table 3: List of scale ratings for each models axis..... 32

1 Introduction

Throughout game history, companies have been trying to create innovative games that the world has never seen. Games that the market audience will buy because its unique selling point is so different from other competitors. From advances such as online multiplayer from DOOM and story driven role playing games from Star War Knights of the Old Republic, games have always had this sense of innovation and this is the same for modern day.

Procedural content generation (PCG) is a modern day advancement in games that can change the very design of future creations however, it is used very rarely in this industry. The basic description of PCG is the ability to generate content algorithmically rather than manually implementing it. This means creating content “on the fly” rather than basic manual distribution. An example of this is the game Minecraft. A game in which the player must survive against the monsters of the world generated using the terrain to create shelter and weapons. Dwarf fortress is uses this approach as well for its world. In this, every world that is created in a new game is generated in the PCG fashion So that every time a player involves themselves in this application the world is different and therefore the player has a new experience. Julian Togelius (an expert in this field), has tested PCG on games similar to Super Mario Bros to generate different levels. From the testing on this, a theory has emerged that PCG does not ruin a game like Super Mario Bros but instead improves it to attract new players. This is the case because of the positivity that was apparent when testing with the selected players (Togelius et al, 2009).

Amongst all the research done on PCG, there is one which focused on the idea of generating different weapons. This was the game called Galactic Arms Race (GAR) created and designed by Erin Hastings and Kenneth O. Stanley. The main overview of the game is for the player to pilot a spaceship and battle all sorts of enemies to gain new skills and levels while finding new weapon systems to upgrade the firearms of the ship. The PCG part of this is the weapon systems for when the player finds isotopes to upgrade the weapon system, the current system doesn’t change but rather evolves into a new weapon blast.

To say that PCG does make a game “fun”, one would have to see how a game is fun. Using the “8 kinds of fun theory” (Hunicke et al, 2004), it is possible to analyse the effect of PCG in the game and see what impact the change will have on the fun aspect of it. For example, in the Super Mario Bros research narrative could be a new fun element because of the persistent change of different levels causing each player’s first round experience to be different. This would cause different stories of the events that transpired in each individual’s play-through of the game. Expression

would be an example for GAR manly because of the different weapon patterns that can be achieved by the player's in-game.

So with PCG you can add an essence of randomness to any game environment or mechanic of a video game but this is not the limit. This project will look deeper into PCG and consider the possibility of using this method as a way of generating player avatars and see if this makes the game more "enjoyable" like the Super Mario Bros test devised by Julian Togelius. GAR will be used as a tool to test the project. The plan will be to generate two new versions of GAR, one will have new generated spaceships with the same stats as the original ship and the other will have the same new ships but with different randomised stats. With these two versions, it is hoped that an answer will be found that will reveal whether the procedurally generated spaceships make gameplay in GAR more enjoyable or not.

2 Literature Review

2.1 Introduction

This Section will go into further detail about Galactic Arms Race and what Procedural Generation is to GAR. Following this, game enjoyment will be discussed further to explain the definition of it and how this is possible in games. This will then lead into further discussions on how a player can identify with their sprite/model. While going into the reasons and possible reactions of players, when their models are procedurally generated. All of this will hopefully give a theory of whether different player sprites will effect game enjoyment based on the knowledge mentioned previously (see introduction).

2.2 Galactic Arms Race

This game was originally used as an experiment to show PCG in games. Hastings (2014) talks about the purpose stating;

“A key objective of this game is to explore the potential for automated content generation technology to intelligently create content for games. In GAR, all player weapons are generated by the cgNEAT algorithm based on weapon usage statistics. However, cgNEAT does not simply respawn weapons that people like. Rather, it creates new weapons that elaborate on the ones that have been popular in the past. ”

As mentioned previously by Hastings, the weapons are generated using the cgNEAT algorithm. This algorithm is based on a method of machine learning and pattern recognition called an artificial neural network. For GAR, a particular type of neural network was used called a compositional pattern-producing network (CPPN). The difference between this neural network and the standard is that the basic network can only contain certain function types while this one can hold an infinite amount of functions types. One of the co-workers of Hastings (Kenneth O’ Stanley) developed his own algorithm which allows neural networks to evolve with the neuroevolution technique (training neural networks to learn so they adapt to new challenges). This developed algorithm is called neuroevolution of augmenting topologies (NEAT). With CPPN and NEAT, you can merge the two together and create cgNEAT. cgNEAT stands for content-generating NEAT which evolves custom content based on user preferences in game.

There are many different methods of implementation that can be used to add procedural generation in games. GAR uses SBPCG (search based procedural content generation) as its main method to generate the content. This will be discussed further when talking about procedural generation later(see what is procedural generation). With SBPCG in place, GAR is able to change

the weapons in game so that they adapt to the style of a particular player. GAR was released five years ago in 2009 and one of its creators has stated that *“GAR is the first game to enable CCE of novel game content”* (Hastings et al, 2009b). CCE in this case is Collaborative content evolution. Hastings (2009b) explains this well by stating;

“In short, players begin the game with an initial set of content. If players use some of the content often, it is inferred that they enjoy that content, and the game produces new content that extends from or elaborates on that preferred content. However, if players are unhappy with certain content, they will not use it (or may discard it); thus the game will not produce more content of that type. The aim of this process is to continually evolve content based on the preferences of the player.”

GAR as a game has three game modes available to its player. One is a single player campaign in which a player can just follow the missions given to play the game. The second is a defence mode where the player is thrown against all sorts of aliens in which they must defend against. The final mode is a multiplayer mode in which players can either work together to defeat their enemies or just have the capability to kill anyone in a player versus player environment. For the project, it is hoped that the procedurally generated ships will work on all game modes. However, this is not an important feature, because as long they work on one the testing can be done.

2.2.1 What is Procedural Generation?

Procedural content generation has been touched on ever so briefly in the introduction to this whole document. This section will explain in more detail what PCG is and its definition. So as stated before procedural content generation is a programmable method to generate game content, using a random process in which the results would be unpredictable to anyone playing the game. The key factor of PCG is the randomness. This is because the whole concept suggests that from small numbers can a large number of possible generated content types be made. As mentioned previously, SBPCG is used by GAR as a method of PCG. SBPCG is where the program will search through the activities of the player in the game and find any common links. These are then sent to the key part of SBPCG which is the fitness function. The purpose of the fitness function is to “grade” the content that the player uses by either a vector which stores real numbers or an integer. When different types of fitness results are mixed, new content appears in the game. This process repeats itself while the game is running and all conditions are met. This also shows how PCG can give the illusion that the content in the game is evolving constantly.

2.2.2 Uses of Procedural Generation in Games

PCG is a very rare addition to games as it is currently still new research that is still being tested. According to Julian Togelius, this is still a new concept which still doesn't have its own textbook (Togelius et al, 2010). Although some modern games are using this technology, its uses are still small compared to other advances. A lot of tests and research have been done into the subject to see what effects it will have on games and the people who play them. Dr Julian Togelius has tested PCG in many different ways to see its potential with games. He has made a racing simulation which generates tracks procedurally (Togelius et al, 2010), a rouge-like maze game which the world is created using PCG (Togelius et al, 2010) and assisted in the PCG levels for the Super Mario Bros game mentioned previously (Pedersen et al, 2009). There have also been tests to change the rule sets of games using the "evolving" idea which was stated before when discussing GAR. One game this was tested on was Pac-man (Togelius and Schmidhuber, 2008) where the game objective(s), player position and AI sprites were all generated in the PCG fashion. Like SBPCG, a fitness function is used to determine which rules are bad and which ones should be kept. Another example of changing the rule sets happened with Dr Cameron Browne, who looked at the same idea but for board games such as chess (Brown, 2008). This follows the same form of generating rules for the Pac-man game proving that this idea wouldn't just work on video games but board games as well (although the board game rule set would be created via a video game or simulation).

As for games that use this and have been released, they have taken a different view on what PCG can do and have used their selected methods well. Minecraft was the previous example mentioned which gives procedurally generated worlds. Façade uses PCG to make stories for the player to interact in. The game places you in the company of a couple known as Grace and Trip who seem to have tension in their relationship at the moment. The player can interact with the couple to try and calm everything down or make the situation worst starting a domestic confrontation. Every action the player makes will generate a new random outcome causing procedurally generated reactions and endings to the game. Elite is one example that deserves a definite mention as it was most likely the first game to use PCG. Elite is a space trading game, which lets the player upgrade weapons, cargo and other equipment too. Completing missions ranged from asteroid mining to piracy. Elite was played on an eight-bit system and had eight galaxies holding two hundred and fifty six planets. This was the maximum they could do on an eight-bit system, which is very impressive. This is why this game is important to the project in hand. This game is the first shoulder of a giant of which we stand upon for this was a revelation!

These games have shown that PCG is a great edition to any game and show that it is possible to create a game which can be classed as enjoyable to a player.

2.3 Game Enjoyment

There is a question that philosophers and scientists in the games industry have always tried to answer. This question is “what is a game?” and it’s a question that no one has truly answered. Now although this project will not go anywhere near this question, one factor that many people believe is critical to being a game is the “fun” aspect. In other words, is the product enjoyable? Game enjoyment is a field of study which people like Robin Hunicke have tried to form a system of deconstructing games to find fun. However, to find the cause of fun or game enjoyment you have to know what these terms truly mean.

2.3.1 What is Game Enjoyment?

Game enjoyment is what makes a player want to play a game. It is the “pull” of the ability to act in a particular way which appeals to a player. This could be because of the challenge that the game presents or the competition between players to prove that one is better than the other. Peter Vorderer and Tilo Hartmann (2003) have found evidence of this claiming;

“On a broader level, the user’s feeling to play against an opponent likely evokes a social-competitive situation that should be especially capable to engage and to involve the user. Therefore, it appears reasonable to regard competition as a major factor in the explanation of video game enjoyment and of the preference for such games”

So if this is what game enjoyment is, how does one make a game enjoyable?

2.3.2 What Makes a Game Enjoyable?

As mentioned previously, game enjoyment can be the sense of challenge in games. We can link this form of fun in a MDA analysis or framework. MDA stands for Mechanics, Dynamics and Aesthetics and is a formal approach to try and understand games. MDA was designed to help deconstruct games and fill any links that cannot be foreseen:

“MDA is a formal approach to understanding games - one which attempts to bridge the gap between game design and development, game criticism, and technical game research.” (Hunicke et al, 2004).

The mechanics can be described as the basic rules of a game or the components of the game, at the level of representing data such as algorithms. Dynamics are the parts of the system that effect the game or the behaviour of player mechanics during the applications run-time. The Aesthetics

are what can be considered the fun aspects that the player goes through or the emotional responses of players who interact with the game. In this case, it is the Aesthetics that will show what factors of the game are fun. This is where the previously mentioned “8 kinds of fun” rule comes in. This rule basically states that the enjoyment a player can go through when playing a game can be spilt into eight categories. These categories are:

1. Sensation
2. Fantasy
3. Narrative
4. Challenge
5. Fellowship
6. Discovery
7. Expression
8. Submission

Sensation is the pleasure gained from the players senses (sight, touch etc.) The best example is when particular animation that the player enjoys is seen or, the vibration of the controller in the player’s hand fitting with the situation they see before them (an earthquake or explosion in game). Fantasy is the idea of make-believe in games. This is just like standard RPG’s (role playing games) that have been set in the future or set in a magical time period with creatures such as elves. Narrative is the effect of drama in games or an unfolding story. This can be witnessing a murder or creating your own story in game. Challenge is arguably the most famous kind of fun and the most important. It shows the game as an obstacle course for the player to navigate through. Fellowship is the idea of the game being a social framework. This means players have the ability to play with friends either competitively or cooperatively. Discovery shows the game as uncharted territory and a new experience for the player. This is best described by games which allow the player to wander around its environment and, find items that they wouldn’t have found if they hadn’t went off the course of the story. Expression is the ability of the player to have an effect on their surroundings in the game. This can be done easily in sandbox games where the player can affect their terrain to build or damage areas of the environment. The final one is submission where the player lets themselves be taken into the game and “get stuck” in it. This isn’t just about addictiveness but also that the game is a quality way to “pass the time”.

From this knowledge, we can deconstruct games and find out what kinds of fun these games possess. With this, it is now possible to see what kinds of fun GAR has and which ones are most important. This is key to find out so that the changes that will be made to GAR for the project do

not effect too many (or any) of these points. GAR has at least four of these in its makeup. These are Challenge, Sensation, Discovery and Submission. However, it is possible to get different types of fun to become more influential in the game depending on the models created. It all about the player being able to identify themselves with their character.

2.4 Identify with the Player Sprite/Model

When playing a game, the person who is interacting with the application will see who they are in the environment. It is important the player believes that they are the main protagonist in the game rather than a “follower” of the ship. Though this tends to be an issue with other sources of media (i.e. books), it is important that it doesn’t happen in the game. The idea is that the player and their “avatar” associate with each to the point that they almost become one (Jan Van Looy et al, 2012). So how so we make a player become one with their character? Referring back to the “8 kinds of fun”, these can be used to bring the player and their avatar closer together with possibly an emotional bond. One method of doing this could be to add some narrative to the game but more than just a story. Perhaps more drama or more uniqueness to the player. This is where PCG player models may have an advantage over standard models.

2.5 Procedurally Generated Player Models

This has never been done before or if it has, there is very little documentation about it. Procedurally generated player characters would give the player a new character each time an event is reached. This event could be when the player makes a new game or when a character dies (depending on the game of course). In the case of the project, it will when the player makes a new game. As mentioned before, players have always found a way to associate with their characters. With procedurally generated characters, the player may find new ways to associate themselves with their new avatar. It would be hoped that this will give players a new experience each time a new game is played. It’s about the experience that one gets through the body that they play in on screen. This has been mentioned by Jan Van Looy (2012) *saying “...the fact that one experiences one’s environment through one’s body container, and which has previously been used in relation to game experience e.g. by Ducheneaut.”* This claim is also followed with;

“This allows us to maintain the connection with the notion of presence while at the same time emphasizing the fact that the experience is not unmediated but mediated by an embodied role in the game world.”

So with this we know that to get the player to link themselves with their generated character, they must have experiences that cause them to link together. With the project being built upon GAR, it is fair to say that the space environment will give the players plenty of room to create

their own narratives and links with the procedurally generated ships. The only question to answer is, will these generated models truly effect the players game enjoyment?

2.6 Will Different Player Models Effect Game Enjoyment?

As said previously, no one has done anything with PCG with player sprites so there is no answer to this question at the minute. The only course of action that can be taken is to theorise the possibilities of this are and what the end results could be. When looking at the possible results, it would be wise to consider the perspectives of the people playing the game and the designers of it. The reason for this as explained by Hunicke et al (2004) is;

“It helps us observe how even small changes in one layer can cascade into others. In addition, thinking about the player encourages experience-driven (as opposed to feature-driven) design.”

So one possible result could be that it does work perfectly and we would see that the player has developed a link to their in-game avatar. If the worst case scenario became apparent and that no linked developed then it would be clear that this idea would either need further exploration or complete removal from all future endeavours in PCG for player models. It is hoped that there will be some potential for future games to look into the possibilities of doing this for character generation. With all the research gathered, it is now time to make a plan and design this project so the implementation of versions two and three (along with the original) are tested and results can give a satisfactory conclusion.

3 Methodology

3.1 Introduction

This section will go into more detail about the project itself. There will be an overview of what the aims and objectives are of the project to show what is hoped to be achieved. There will be a discussion on how these targets will be met by discussing what methodologies could be used. A description of advantages and disadvantages will be given and ultimately the preferred methodology will be revealed. A risk assessment and its contingency plans will be put forward to show what can halt the projects progression and how to deal with the issue. A list of Milestones will be made to see what points must be reached to get closer to completion. The research methods and tools used will be the final edition to this section explaining how the information needed will be found out. After all these points have been discussed, the design part of the document will commence.

3.2 Aims and Objectives

The overall aim for this project is to see if the generated space-crafts increase the fun gained from the game GAR. This will be done by creating two versions of GAR which will be tested by willing participants. One version will just have the procedurally generated ships as its only change. The other will have this plus the edition of random stats for ship attributes (i.e. speed, armour etc). The versions when tested will hopefully give results which will either show that procedurally generated ships do make a difference to the player on their own. It is possible that this result will not come about but rather the difference will be noticed when the ship stats change.

In order to achieve the aim, there are certain objectives that must be met in order to reach the overall target. The objectives of this project will be to learn how GAR works so it will be easier to work out what to add without damaging the original game. One other objective is to keep GAR versions one and two as close to the original GAR as possible. A testing session will need to be conducted to see how people react to the ships and to see their reaction overall whether it is a positive one or not. Once these objectives have been met then the aims will have been accomplished as well. A plan must be created however to show what steps need to be taken and how long these stages will take to reach the overall goal.

3.3 Project Management Methodologies

Project methodologies are the ways in which an aim can be broken down and managed into small targets, while following a particular method to complete the ultimate task. This is an efficient way to organise a project of this calibre. The project becomes deconstructed and results in a plan which has subtasks and milestones as points of progression. Each subtask is given its own time period in which the job starts and ends. Each subtask is formed by each individual task that would need to be completed in order for there to be progress in the project. It is worth considering to add extra time to the tasks in case any contingency plan becomes active. To create this project plan, one must find out of the methodologies that form these management methods so that the best one is used for the main aim.

3.3.1 Types of Methodologies and their Cycles

These methodologies are special ones for software development. These are frameworks that are used to plan and control the process of developing a system. There is not just one framework, for over time different methods have been created to help with certain projects that the others could not assist with. Frameworks have also improved/evolved by removing major weaknesses with its form. This is so that they become useful in modern day projects. Examples of these frameworks are:

- Waterfall
- Agile
- Spiral
- Prototyping

Waterfall and Agile shall be discussed in their own sections for they are the most important to the project. Spiral is a risk-driven process that bases itself on the risks of the project (Skumar 2014a). The spiral model tends to adopt features of other methodologies such as waterfall and prototyping. This model has four phases called planning, risk analysis, engineering and evaluation. The project will repeatedly go through each phase in iterations also known as spirals. At the beginning, the project will be in the planning phase. Requirements are gathered and the first risk is assessed. The advantages of this are that there is a lot of analysis on risks which means that there is small chance of any appearing, very good for critical or large projects and all software is made early in the software phase. The disadvantages are that success depends on the risk analysis phase, it doesn't work well for small projects and risk analysis needs highly specific expertise.

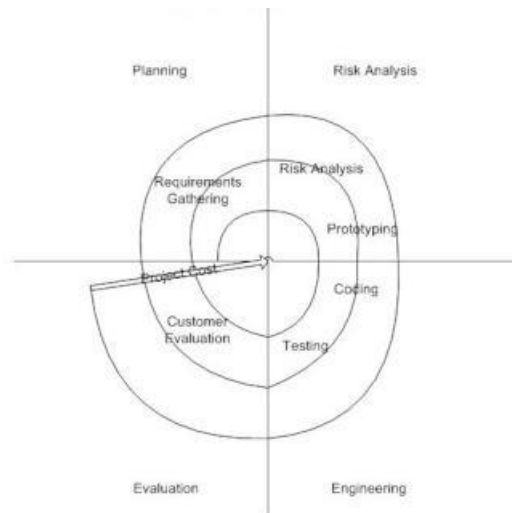


Figure 1: Table showing the spiral Development (Skumar, 2014a)

Prototyping is very different from other methodologies. This method basically doesn't start with a design, requirements or implementation phase but instead starts with a throw away prototype (Skumar 2014b). The prototype is built with the current requirements that the project is aware of. The idea is that the client will get a feel for how the system will work and interact. This helps for understanding what the extra requirements are. The advantages of the prototype model are that errors are found a lot sooner than other methodologies, any functionality that could have been missed can be identified a lot sooner and the user feedback is quicker which helps finding better solutions. The disadvantages are that the prototype could be way off what the client intended and therefore repairing the first design, the system may get too complex and therefore go out of scope and the application may not be used as the whole system was incomplete or the problems were not analysed correctly. These methodologies although good in their own right would be inadequate to this project because the project is only a small one so spiral would be useless. As GAR is already a published game it would be completely stupid to use prototyping method for there is already a system that I can use. Waterfall and Agile however are more useful but there is a major problem with one of them.

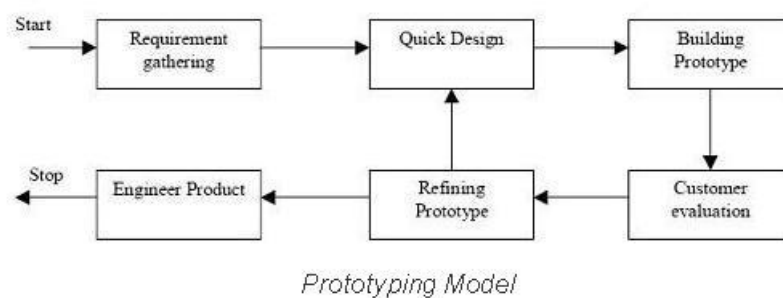


Figure 2: Table showing the prototype development stages (Skumar, 2014b)

3.3.2 Waterfall

Waterfall is the most famous methodology as it were. It was the first process model suggested to businesses (Skumar, 2014c). It is sometimes referred to as the linear-sequential life cycle model. The process of the water model is to complete each phase fully before continuing onto the next phase. At the end of every phase, a review is done to determine if the project is on the right track and if the project will continue or not (i.e. completely scrap the project). None of the phases in the waterfall model must overlap for this breaks the model entirely.

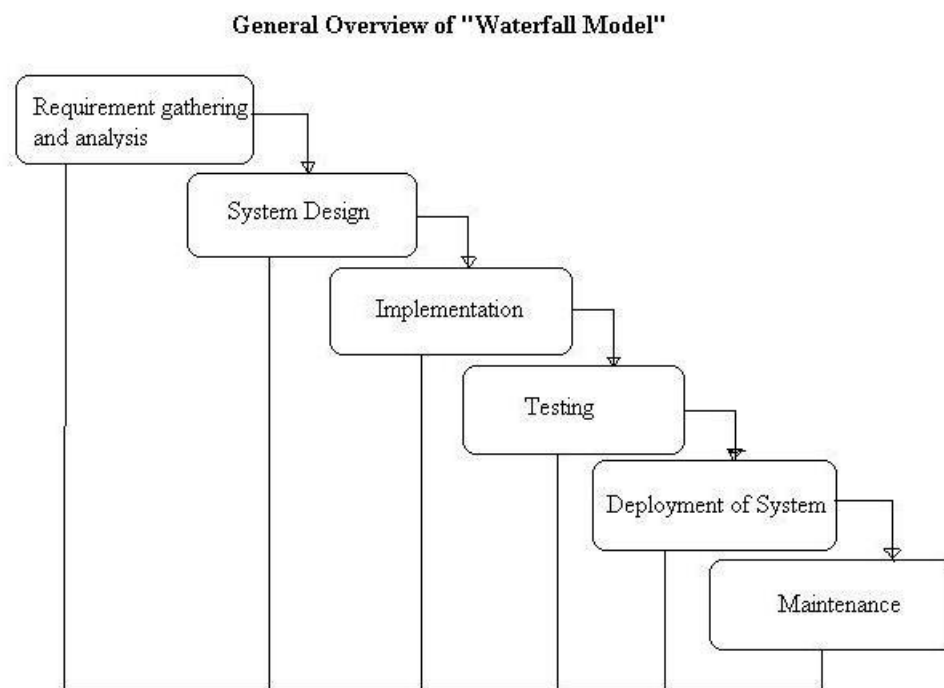


Figure 3: Table Showing the Waterfall Development Stages (Skumar, 2014c)

Now there are some good advantages to using this model. It's a very simple and easy model to understand and use. It's a very easy model to manage as each phase has certain deliverables and a moment to review the stage. Phases are completed one step at a time. The final advantage is that this model works well for small projects when all requirements are fully understood. There are some disadvantages to this model however which makes this style not as good for programming tasks. As soon as the testing stage has been reached, it is very difficult to go back to a previous phase and change something that has proven to be a flaw in the concept stage. No working software is made until the later stages of this process. There is a high amount of risks and uncertainty in the project. This is a very poor model for complex tasks or anything object-

oriented. This is a terrible method for long projects. The final disadvantage is that it's not suitable for projects that are at a high risk of changing. Basically, the waterfall model should only be used if the technology is understood, the project is short and the requirements are well known and clear.

Although the waterfall model sounds good there is a flaw that was mentioned previously that makes this completely useless to this project. As the program is built on an object-orientated approach, waterfall would cause too many problems that will ultimately hinder the project. Waterfall is a terrible model to make any system. Even its designer claimed it was a flawed methodology (Royce, 1970). There is one methodology that will fit very well with the project and GAR itself. This method of project management is the Agile methodology.

3.3.3 Agile

The Agile methodology is a special kind of process known as an incremental model. An incremental model is when the whole requirement is divided into different builds (Skumar 2014d). This creates multiple development cycles which the project goes through. Agile is the same where the software is developed in incremental, quick cycles. The results of this are small releases which have been built on the previous functionality. Every release is tested thoroughly to make sure that the quality of the software is maintained and kept to a high standard. Agile can be broken down into smaller methodologies like extreme programming.

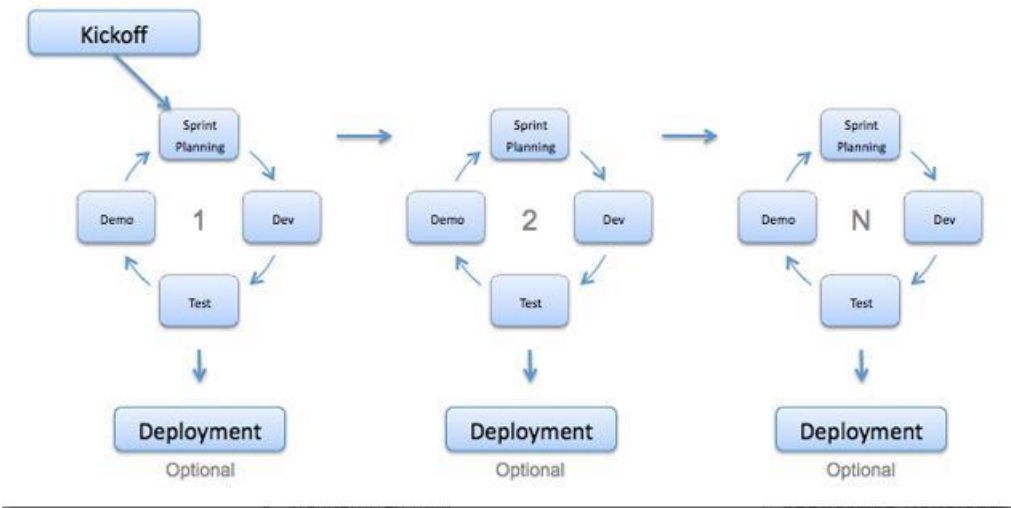


Figure 4: Table Showing the Agile Development Stages (Skumar, 2014d)

As usual, these methodologies have their advantages and disadvantages. The first advantage of Agile is the satisfaction the client will get when new software updates are delivered rapidly.

Instead of focusing on tools and the process on it, the focus is on people and the interactions of the developers, clients and testers are constant. Working software is delivered in weeks rather than months. It encourages face to face communication which is the best way to communicate in business. There is daily cooperation between the developers and the people who work on the business side of the industry. Attention is always on the quality of the technical aspects and the good design. There is regular adaptation to the changing circumstances of the system. The final advantage is the ability to add late changes to the project and not damage the quality of the result. The disadvantages of Agile are that in some cases (where the project is large) it is difficult to assess how much effort is needed at the beginning of the life cycle. There is a lack of emphasis given to any designing or documenting in Agile. The project can go of the right course if the client is not sure on what the final outcome should be. The final disadvantage is that only expert programmers are capable of making the decisions necessary during the development phase of the project.

It is best to use this methodology in a number of situations. When new changes need to be implemented in the system as agile gives the freedom to do so with little cost. Adding a new feature to the system is easier because developers will only lose work that they have done in a few hours or days. Unlike the waterfall model, Agile demands a very limited planning to start the project. This is because it is always assumed the end users' needs always change therefore, Agile gives the ability to add or remove features based on the customer's feedback. The final reason you would use agile would be because all members of the team will find they have more freedom of time and options than if the project was developed in a rigid fashion. The ability to have options gives the opportunity to leave important decisions until more data has been collected meaning that the whole project can continue to move in the right direction without the fear of reaching a point where it must stop.

This will be how the project will be ran from a management point of view. Agile gives the freedom needed to go into GAR and play with the functions within the code. There would be no worry of changing the whole project if there is an issue with the models or the GAR code itself. Overall Agile is the outstanding candidate for the projects methodology and it is certain that this will help stay on the correct track. Now that this has been sorted, a risk assessment must be made with contingency plans to go along with it.

3.4 Risk Assessment and Contingency Plans

To properly analysis the risks that may affect the project, a risk assessment has been created (see table 1). The risk assessment allows the potential risks to be identified and grade accordingly by

the likelihood of them appearing. It will also allow the chance the grade by how great of a risk they are to the project. With this method, contingency plans are made to combat any risks if they become apparent during the project cycle. These plans give a number of actions that will need to be acted upon if a certain risk is met. This should help reduce the chances of these risks effecting the project.

Risk	Likelihood	Impact	Contingency plan
Loss of Work	Low	High	Back up work all the time and store on external devices (such as a USB stick) or online services like dropbox
Loss of data	Low	High	Make lots of copies of the data gathered at a regular basis
Lack of skills or knowledge on PCG	Medium	High	Research about the area of study and understand how to use it knowledge and skills improve
Time management is bad or time is running out	Low	High	The project must be planned carefully. Tasks must have realistic time constraints with the overall project. Extra time should be given in case a contingency plan comes into play. This way, problems can be fixed and this doesn't affect the timeline of the project.
Late changes to project	Medium	Low	Changes are allowed but there must be constrictions. Not too many changes should be allowed for this may take the project of course. The change must be able to slot into the projects timeline without rearranging too much otherwise the previous risk may appear.
Scope creep may get into the project	Medium	Medium	Give an overview of what the project is and what needs to be done. Anything that can be considered unimportant and don't have to be done can be deleted from the project plan
Not a lot of people test project	Medium	Medium	If contacting people via e-mail or social networks (like Facebook) and not many sign up to do testing then a face to face invite may be necessary so people are less likely to ignore the test organiser.

Table 1: Risk assessment table at its contingency plans

3.5 Gantt Chart

Once all of this has been done, it will be time to make the Gantt chart. The Gantt chart is a tool that is used to visualize the process of the project with all its subtasks and milestones listed. For this project, the Gantt chart is an excellent reference for what has been done, what needs to be done and how long is left to do it. This helps when trying to see if there is enough time to add new tasks to the project. The Gantt chart is an effective way of keeping a project under control and is very useful to any project. A picture of the first draft Gantt chart is below (Figure 1). The full Gantt chart can be found as appendix 1.























		Task Mode ▾	Task Name ▾	Duration ▾	Start ▾	Finish ▾	Predecessors ▾	R
1			▶ Learn GAR Code	11 days	Mon 11/11/13	Sun 24/11/13		
7			▶ Research	11 days	Mon 25/11/13	Sun 08/12/13	1	
12			▶ Design	11 days	Mon 09/12/13	Sun 22/12/13	7	
17			Research and design done	0 days	Mon 23/12/13	Mon 23/12/13	12	
18			Robotics Assignment	4 days	Mon 09/12/13	Thu 12/12/13		
19			Christmas Holiday	6 days	Mon 23/12/13	Sun 29/12/13		
20			▶ Make space-craft	11 days	Mon 30/12/13	Sun 12/01/14	12	
24			▶ GAR version 2	36 days	Mon 13/01/14	Mon 03/03/14	20	
32			GAR version 2 created	0 days	Tue 04/03/14	Tue 04/03/14	24	
33			▶ GAR version 3	6 days	Mon 03/03/14	Mon 10/03/14	20	
38			game engines assignment	4 days	Mon 20/01/14	Thu 23/01/14		
39			GAR version 3 created	0 days	Tue 11/03/14	Tue 11/03/14	33	
40			▶ Test game	6 days	Mon 10/03/14	Sun 16/03/14	33	
46			▶ Questionnaire Preperation	6 days	Mon 10/03/14	Sun 16/03/14	33	
51			▶ Big test	6 days	Mon 17/03/14	Sun 23/03/14	40,46	
57			Testing Done	0 days	Mon 24/03/14	Mon 24/03/14	51	
58			Robotics Assignment 2	4 days	Mon 24/03/14	Thu 27/03/14		
59			make notes for evaluation	7 days	Fri 28/03/14	Sun 06/04/14		
60			▶ Evaluation	19 days	Mon 07/04/14	Thu 01/05/14	51	
67			Dissertation done	0 days	Fri 02/05/14	Fri 02/05/14	60	
68			project completed	0 days	Fri 02/05/14	Fri 02/05/14	60	

Figure 5: Gantt chart

The Gantt chart as seen above follows ten main tasks to complete the project, these tasks are:

- Learn GAR code
- Research
- Design
- Make space-craft
- Gar version two

- Gar version three
- Test game
- Questionnaire preparation
- Big test
- Evaluation

The first task is simply to understand the code that Hastings and O' Stanley have made. This code must be fully understood if it is to be used for the project. Secondly, research must be done to understand how PCG works and how it is used in the game. Once that is done, it will be time to start working out how the ship parts will be generated and what ship parts will be needed for the project. The next phase will be to make version two with the procedurally generated ship-crafts in it. After that is added a test is needed to make sure it works, then the work on version three can begin. Version three will be just the random stats for the ship parts so this won't take as long to accomplish. After both versions have been made and tested, the questionnaire must be made to answers the questions that this project asks. Once this has been made, it is time to test the game on the public and find the answers. The final stage will be the write down the results in this very document. All of these can be split into milestones which show that the project has made progress.

3.6 Milestones

Milestones for project plans are significant stages in the cycle of the product. These are the points that are most important and are considered a step closer to the end of the project when one is reached. In this project, there are five milestones which insinuate that progress has been made in the project. These stages are:

- Research and design done
- GAR version 2 created
- GAR version 3 created
- Testing complete
- Project handed in

The first milestone shows when all planning work is completed. Therefore, when this milestone has been reached all designs and research have been accomplished so the programming can begin with all the unknown knowledge that was needed gathered. When version two is made, it

shows that the procedural code has been created and that it works. When GAR version three is completed, that will show that all programming has been completed and the testing can begin. The testing complete stage is when all the data has been collected and all that remains is this write up. The final milestone is when this write is completed and the project had been handed in thus ending the cycle. Now that all the milestones have been discussed it now comes onto the research methods of the project.

3.7 Research Methods

For research methods there are two basic ones that can be used for the project. They are qualitative and quantitative research. Qualitative research focuses on the difference of quality in the product while Quantitative focuses on numbers (e.g. statistics) of the product. As a lot of data will be collected and stored, it would be best to focus on a qualitative method of research rather than quantitative. However, although the method of research has been selected, a way to collect the data needed must be formed. This method of data gathering will be in the form of a questionnaire.

The questionnaire will be used to find out how the procedurally generated ships effect the players when playing GAR. It would be wise to collect members of the public who would consider themselves as “gamers” as the main target audience for the testing. This is because people who don’t play games often may have not experienced the deep contention one player feels with their avatar thus, their answers to the questionnaire may not aid the project in reaching its aim but rather give useless data. Due to the time constraints from the project plan, there will not be a lot of test days to do this on. It is likely that the testing will be done one day, maybe two if time will allow it.

There are some benefits to using a questionnaire that will help this project collect the data needed. One is that this way allows the data to be collected, to be gather and analysed in a very simple way. Another plus of using a questionnaire is that questionnaires are an effective way to collect data from any number of people, no matter how big or small the group is. As this is a new test on PCG, it is unlikely that there will be a questionnaire out there which can be used for this project. Therefore, it is deemed correct that a new questionnaire is to be made. One which will focus directly on the projects aim.

A test day will have to take place in a room which will have the ability to play the game with no issues occurring with the hardware. A suitable place to do this would be lab B at the University of Lincoln’s MHT building. This is because the PC’s in that lab are of a high quality and can run the game with ease. There is also the plus that the university has a games course thus gathering

students to partake in the test day should be a little easier than before. On this day, the data shall be collected for the project and there will be close analysis upon it so an answer can be given as this projects conclusion. All this research when completed leaves the tools as the only matter to be discussed

3.8 Tools used

Certain tools will be needed to complete this project successfully. Tools are an essential part of the project for they must be used to complete the projects aims in the correct manner. If the best tools are not used then the chances of the project failing would increase worryingly. The tools that were chosen to complete this project are:

- Visual Studio 2010
- XNA framework
- Microsoft office suite
- Blender

Visual studio is the most important tool to the project. Visual studio is software that allows a user to program in high-level languages. These languages (such as C#, C++ HTML, Java and f#) can all be written using this software. The downgraded version which could work is notepad++. Notepad++ is designed to work with beginners, which means it is very hard to work on complex projects with it. Visual studio however, gives you all the advanced methods this project needs plus, GAR was written in C# using Visual studio so it would be unwise to re-write the whole GAR code in a whole new language. It has proven to be a wonderful tool to use and has been fantastic as a way to show the program layout and debugging with it has been fairly easy.

XNA was a framework designed by Microsoft to work over C# (in visual studio) to help indie companies make games. It was a good quality framework, which allowed the user to make any game they wanted to make with no negatives for making 3D instead of 2D. Microsoft have since spoken about XNA claiming that it's not in their future plans which means a new alternative may need to be found. XNA however, can be downloaded from other sites and it is feasible as the main choice because it can be downloaded at Dreamspark for free. GAR was made using XNA and there would be no point trying to remake GAR in another framework to the same reasons as re-writing GAR in a new language. This tools has made all programming much easier and the GAR code fits nicely with the code that has been implemented with it.

The office Suite Microsoft offer is a high quality tool, which most people will have used or have heard of at some point. The project may deem it necessary to design a spreadsheet for testing or

write up a document to store notes in. This tool will have as the software needed to help in these tasks. Another similar tool that could be used is open office. Open office is free and offers the same type of software as Microsoft do. There is one problem with using open office. This problem is that most documents will be written in word. Open office is known to have issues with converting from itself to word. This issue is one that would cause many problems for the project so Microsoft office is still the first choice. This tool has been used to create the questionnaire and this very document. The office tool Microsoft project was also used to create the Gantt chart for this project.

Blender is a free kit, which is used to design and create models. Blender is an easy tool to use and can link with most programming software (including Visual Studio). Blender has a unique feel to it as it seems to be designed as if it was meant to be used by programmers. The alternative to this is 3DS max. 3DS max is a top quality tool which allows its user to develop almost any model they want. There is deciding factor that means Blender is the main tool and not 3DS max. This factor is that 3DS max is very expensive while Blender is free which means hardly any money is spent on the tools for the project. This was used to create all the models in versions two and three of the project.

All of these tools will be important to the project and with them, there should be a great chance of the aim being reached in time.

4 Design

4.1 Introduction

In this part, all the design details of the ships and the questionnaire shall be explained. There will be a closer look into how the ships were designed and why they were designed in that particular way. This will go over each part and explain why they were picked and the ideas behind them. More depth about the questionnaire will be done as well to show what questions will be asked and why those questions were picked.

4.2 The Ships

GAR starts off with one basic ship which you can either upgrade to its full potential or buy another ship which is vastly superior to the starter craft. With this project, the player will start off with a randomly generated ship instead. The problem is generating the ships, what would make the ships truly procedural? It would be impossible to design 100 ships and make them all different. Even then, there would only be the 100 that the player can get a ship from which wouldn't make it feel random enough or truly procedurally generated. It was decided that instead of designing different ships, different parts of a ship would be generated instead. At first the parts that were planned are the body, head, engine, wings and weapons. Due to the fact there was extra time, it was decided that an extra part would be made namely the ship tail (the reason for this will become apparent).

Now the plan was set on what was going to be made, the question was how many of each part was to be made? After careful consideration, the solution was that ten models for each part would be made. At the time there was going to be only five parts to a ship so a suitable number had to be picked. If you were to say that the number of models was "N" and the number of parts was "X", then the statement N to the power of X can be made. Now in this case, X will be five so this means whatever number is used to represent "N" will always be multiplied by the power of five. In the case of ten models, this would mean that the result number of ships that could be generated is ten to the power of five. The answer to this sum is 100,000! This means that every time a player starts a new game, one out of one hundred thousand ships will be given to the player. This makes the generation of the ships properly procedural. Now although there is already a lot of ships being generated, what if more were added? If one more part was added, the mathematical statement would be ten to the power of six. This gives an answer of 1,000,000! This

is why a ship tail part was added in the later stages. The extra tail part just gives that extra proof of procedural content generation can do by giving the player 1/1,000,000 ship. It is very unlikely the player will get the exact same ship again.

Once this was done, it was time to scale the ship parts and work out what sizes should be used. Cubes were created to test the sizes of the ships and work out how much each part should be scaled by. One issue with XNA is that the Y and Z axis are flipped when the model is transferred so that had to come in mind when scaling the parts.

	Body	Engine	Head/Nose	Wings	Weapons	Tail
X Max	100	30	100	50	30	100
X Min	80	10	80	40	30	80
Y Max	100	37.5/38	50	37.5/38	30	50
Y Min	80	30	20	30	30	20
Z Max	50	50	25	25	30	25
Z Min	40	40	20	15	30	20

Table 2: Scaling the models

With these numbers found and recorded, it became apparent that the ships shouldn't be all the same size. This is because each ship must feel unique to the others. This could be because the engines are different or any other part but, it's more likely to be noticeable if there is a size difference. As table 2 shows, maximum and minimum values were recorded from the cube testing. These results were put together and mixed in a list so that each part had a different scale to each other. Thus making all ship parts having model of a unique size.

Model number	X Scale	Y Scale	Z Scale
1	Max	Max	Max
2	Min	Min	Min
3	Mid	Mid	Mid
4	Max	Max	Min
5	Min	Max	Max
6	Max	Min	Min
7	Min	Max	Max
8	Max	Mid	Min
9	Min	Mid	Max
10	Max	Mid	Mid

Table 3: List of scale ratings for each models axis

Looking at table 3, it can be seen that each model either has the maximum, middle or minimum value for each axis. For example model one for the body is scaled (100,100, 50), while the model for engine is scaled (30, 37.5/38, 40). Now that all the panning was completed, it was time to start creating each part.

4.2.1 Body Parts

These parts were probably the easiest to design but the hardest to make in blender. All ship bodies have a rectangular shape to them in the sense that they are vertical on the Z axis. Mostly designed with cubes or rectangles with the odd ones built with circular objects such as cones or cylinders. This however leads to why these designs became very difficult to make. It became clear that the values used to scale the ship on its Z axis were wrong at some occasions. This would be because of the different figures for the scaling. It would appear that some parts had been dismantled from the ship which made it look very unconvincing for a player to interact with. Some ships had to be cloned and merged together so that they were longer and looked realistic. These body parts now fit well with the ship from these changes and still look different from one another size wise.

4.2.2 Wing Parts

The wing parts were very difficult to design as it was hard to make ships wings with shapes other than a rectangle. It was obvious to see that most of the wings would have to be rectangular but, some wings were successfully built using cylinders. When placed into the game, it was a pleasing sight to see the wings fit well with the ship bodies. It was also good to see that the wings rotated well with the ship. There was an issue with the rotation at first but that will be explained in the implementation part.

4.2.3 Head Parts

Of all the parts made these were probably the easiest to design and make overall. The head parts are made from a verity of quadrilaterals and circular shapes to fit at the front of the ship. As the front of the head would never be seen, it was easy to design the parts which the player could see and suggest what was on the front (i.e. using a flat headed cone to suggest a point at the end of the head). When added into version two of GAR, this proved to be true and pleasingly fitted nicely on the front without looking out of place at all.

4.2.4 Engine Parts

The engine parts were very awkward if honest. They were based around circular shapes as is customary with most engines in real life. Of course quadrilateral shapes were used to make different designs as with all the ship parts. It became clear that not all ships have the same

number of engines. Some ships in games have one big engine while others have lots of smaller engines. As this was the case, it only seemed fitting that the number engines that appear on the ship change as well as the shape of parts. When added into GAR, the engines fitted very well at the back of the ship and looked as if they were part of the actual game. This was a very pleasant sight indeed. There was an issue with the rotation (same as the wings prior to the engines) but that will be explained later.

4.2.5 Weapon Parts

These were the odd ones in the design stage of the project. According to table 2, these were the only models not to have a maximum, middle or minimum value. The weapons were all design to be thirty across the X, Y and Z axis for scaling. This was because when testing the weapons, it became apparent that the weapons had a small difference between its maximum and minimum values. The reason for the small difference was that the weapons couldn't be too big so that they were unrealistic but they couldn't be too small so the player can't see it. So when the value were collected, it proved that there was no point to changing the scaled of the weapon. Instead, the weapons were designed to be as different as possible so that they looked different in the game from their shape and figure rather than the size. Implementing these into the game was very simple. The weapon model fitted nicely with no issues what so ever.

4.2.6 Tail Parts

The last parts to be made for the procedurally generated ships. These were designed very similarly to the ship heads. This basically means that the tails would follow the same scale chart as the head. The tails were designed to be mostly rectangular but like the others have the odd ones that were circular so there were differences between the models. Placing the model in game proved to be tricky as at first it was difficult to find the place where it looked right. This was mostly because of the engines taking a bit of space at the back of the ship body model. The tail eventually had a slot on the ship that fitted it perfectly. This ended all the design work for all the parts of the ship. All that was left to design was the questionnaire.

4.3 Questionnaire

The questionnaire itself must be made according to the objectives of this project. Most of objectives are about the implementation of the project but one is about the reaction of the testers to the ships generated. This means that the design of the questionnaire needs to focus on how testers feel about their ships and if they approved of them instead of the original GAR's ship.

To make this questionnaire as easy as possible, the questions will be split depending on what version was played. Each section of the questionnaire will look at how much the player enjoyed

that particular version of GAR and ask their opinion on the ship to see if it effected the way they approached the game. GAR versions two and three will have an extra two questions at the end of their sections. The testers will be asked if they thought one of the ships they had was better than another. If they agree to this answer then they will be asked to list the ships in an order with their favoured on the top and the worst at the bottom. Once all the versions have been tested, they will be asked if they thought there was a noticeable difference between all the versions of GAR. If they agree that there was a difference, they will be asked to list which version they favoured the most in a descending order.

Now that the order of the questions has been sorted, the next part would be to work out how to ask the questions. When talking about types of questions, Dr Thomas F Burgess (2001) claims that there are multiple ways to make questions "Different types of questions can be used, e.g. open vs. closed, single vs. multiple responses, ranking, and rating." Open and closed questions are ways in which we make the tester answer our questions. Open is when we have no control over what the answers that the tester gives. Closed is where the tester is given questions with predetermined answers that cover most of the possible responses that the participant could give. Single and multiple responses are just how many times a tester can answer a question. Obviously, single is answering the question once and multiple is responding with more than one answer. Ranked responses are best to use when giving a tester a question which demands an answer which lists the possible responses by a level of importance. Finally, there are rated responses in which the tester answers the question using a statement to show how important that aspect is. An example of this is the Likert scale which Dr Thomas F Burgess (2001) gives the impression that it is a popular approach to social sciences.

To gather data which must be used to reach the main aim, open questions would not be recommended as the data would be too inconsistent. As it would be easier for the testers to work on a questionnaire which is as simple as possible, multiple responses may not be necessary because almost all the questions can be answered in a single manner. Any other questions can use the ranked response as a method of answering. The problem with single though is the answers have to be a guess as to what the testers will probably answer when working on the project. Therefore, it would be wise to use rated responses for these questions. The rated responses will range from:

- Extremely disagree/bad
- Quite disagree/bad
- Slightly disagree/bad

CMP3060M Project Item 1

- Neither
- Slightly agree/good
- Quite agree/good
- Extremely agree/good

With this in place, the project now has a questionnaire in which testers will be asked to rate their experiences with their ships and rate them accordingly. Now that this is done, the implementation GAR versions one and two and begin.

5 Implementation

5.1 Introduction

This section will go over the development process of the project and will go over the steps needed to create GAR version one and two. In addition to this, any issues that were found along the way will be explained along with the solutions to that particular problem. GAR version two will be explained first, follow by version three.

5.2 Implementing Galactic Arms Race Version 2

At the beginning of this project, it became clear that GAR was a very complicated game indeed. The number of classes held by this demo of GAR go over one hundred and fifty. Obviously not all of these classes would affect the player model in game so the first task was to find out which classes would not be involved with the project. Once this was done, it would be time to find the class which held the original ship. This class was helpfully called “Ship”.

Ship held data such as the speed, rotation, hull, armour, shield and collision radius. All this data was important in the Ship class for this data is what gave the ship its in game stats. The class also held the model which was stored in another class called ModelMgr (Mgr standing for manager). The problem with this class is that everything was stored here, which makes perfect sense for GAR but not for the project. It was clear that a new class for each part should be made. Using the naming convention Ship followed by the part name. The first new class made would be the “ShipBody” class.

The ShipBody class was very similar to Ship but it didn’t have certain data such as speed and rotation. This was because that data could be stored with other models such as the engines or the wings. When this class was made, the next task would be to find the data that called ship and change it to body. This proved to be a harder task than previously thought.

In hindsight, it would have been easier to get rename the Ship class to ShipBody. The original reason why this wasn’t done was that other A.I. ships use methods in the ship class so a new class just for that model was as first thought to be the best idea. It had not occurred not many null pointer exceptions there would be by referencing ShipBody instead of Ship. The reason for this was simple, some variables were calling Ship instead of ShipBody.

```
// update ship
//ship.Update();
body.Update();
```

An example of this can be shown from the “Player” class (figure 6). This class had a variable referencing the Ship class called ship and the same for ShipBody but that was called body. Player was still calling in some places ship instead of body

Figure 6: An example of changing ship to body

and as Ship was not being referenced, the result would be null and thus the exception was thrown. As the reason for this was known, a couple of hours was all it took to fix this issue.

Once the ShipBody class was fully implemented, it was time to add the other classes. There were some variables like speed which referenced ship still as body did not have data to represent it. They were commented out for moment while the models that would represent them were having their own classes being built. Once these were made and all commented code had been uncommented and linked with their new class, it was time to start adding models.

```
static void LoadModel(string path, string name)
{
    if (ModelIsLoaded(name))
    {
        Log.WriteToDebugLog("Dupe model load: " + path + name);
    }
    else
    {
        Model3d t = new Model3d(path, name);
        if (t.model != null)
        {
            modellist.Add(t);
        }
    }
}
```

Figure 7: The loadModel method in GAR

GAR has its own procedure to adding models in XNA. Normally you would load the models in the initialize of part of the program but GAR as its own class. As mentioned before ModelMgr stores all models needed for GAR. They are stored in one big list called "Model3D" and are read into the game all at once. They use their own loadModel method to add models to the list or duplicate them as can be seen by figure 7. With this method in place the models can be added together in a big list just by called loadModel and giving the path and name the model wanted. The result can be seen on figure 8 in this document. As mentioned before, ModelMgr is referenced in the ship class and was therefore added in the classes for each part of the ship. Now that the models could be called, way of calling them had to be done. GAR already had their own method for that as well.

The models for the project would be loaded into the game and given suitable names. The names would be the first initial of the part of the ship the model represented and its number (e.g. b1 for the first body part made). The only issue with this was weapons and wings as both had the same

initial. It was decided that weapons would have the next character in its name with it (e.g. we).

Figure 8 shows the wing models being added to the load list.

```
LoadModel("player\\", "w1");
LoadModel("player\\", "w2");
LoadModel("player\\", "w3");
LoadModel("player\\", "w4");
LoadModel("player\\", "w5");
LoadModel("player\\", "w6");
LoadModel("player\\", "w7");
LoadModel("player\\", "w8");
LoadModel("player\\", "w9");
LoadModel("player\\", "w10");
```

Figure 8: LoadModel examples

Once the models had been added into GAR it was time to get them appearing in the game. GAR loads everything into the main class (otherwise known as “GalacticArmsRaceMain”). There is one class that is called in the main that occurs quite a lot called “FileManager”. The FileManager stores a lot of effects and lists which are loaded into the main game. One of the lists is for the player ship which loads all the ships which the player can have. This list is actually created by another list stored in another class called “Globals”. Globals stores all data that can be accessed by any other class in GAR. Within Globals there are lists that hold the data for the player and the A.I.’s. To make the new models for the parts, a new list must be made in Globals which is referenced in FileManager. In FileManager, all the models for each part will be added and then the new list in FileManager will be referenced in the main class. Figures 9 and 10 show what globals and FileManager stores now the new parts have been added.

```
//-----
// ship archetypes
// these are the stat definitions for each ship type
//-----
public static List<Ship> playerShipArchetypesList;
public static List<ShipBody> PlayerShipBodyArchetypes;
public static List<ShipHead> PlayerShipHeadArchetypes;
public static List<ShipWings> PlayerShipWingArchetypes;
public static List<ShipEngines> PlayerShipEngineArchetypes;
public static List<ShipTail> PlayerShipTailArchetypes;
public static List<CPPNWeapon> PlayerShipWeaponArchetypes;
public static List<Ship> npcShipArchetypesList;
public static List<Ship> alienShipArchetypesList;
public static List<Ship> pirateShipArchetypesList;
```

Figure 9: The globals which store the ship parts along with other A.I. ships

Once the models appeared on screen, it was time to add the rest in the list. There was just the one problem. All this was fine but it only ever read the first model therefore, never making a new ship each time. It was discovered that the model is read from the player class and then the final steps to make GAR two took place

```
public static void LoadPlayerShipWingArchetypes()
{
    Globals.PlayerShipWingArchetypes = new List<ShipWings>();

    ShipWings w;

    w = new ShipWings("w1", 2000f, 15f, "w1", 2500);
    Globals.PlayerShipWingArchetypes.Add(w);
}
```

Figure 10: The first wing added into the list in FileManager

Now previously not mentioned on in this section was how the model was loaded before. An Int variable was made on the original GAR called "shipIndex". When initializing this would always be set to zero. What GAR does next is what made the ship load models from the list. GAR sets a new "Ship" class and then uses a "ChangeShip" method to call the items from the list. As ChangeShip used the shipIndex to show what model is what in the list, model zero would be the first one loaded. This was going to be used for every part of the ship so the models would be called into player. Figure 11 shows the new "ChangeBody" method that as created to change the body model. New indexes were made for each part of the ship thus leaving one issue left.

The procedural generation was not in the program still. This still needed to be added so the project could move in the right direction closer to achieving its main aim. Where the new indexes were made, it was decided that they wouldn't hold a certain value (such as zero) but a random variable will be added instead. This random variable will go through the list of models for the part its look through and choose a random one form that list. Figure 12 shows the random code that picks the ship parts for the game. With this added, GAR now generated randomly the player ships and with that, version two of GAR was complete. Figures 13 and 14 shows the game with a random ship.

```

public void ChangeBody(int shipBodyIndex)
{
    // change ship stats
    try
    {
        // change index of ship and copy stats
        // scale ship stats to player's level
        this.shipBodyIndex = shipBodyIndex;
        body.CopyShipStats(Globals.PlayerShipBodyArchetypes[shipBodyIndex], this);
        ScaleShipToLevel();

        // change collision radius based on ship
        collisionRadius = body.collisionRadius;

        // reset teleport and warhead cooldown
        warheadLauncher.StartCooldown();
        TeleporterReset();
        engineOn = false;
    }
    catch (Exception ex)
    {
        // tried to change to a ship that does not exist
        if (ex != null)
        {
            this.shipBodyIndex = 0;
            ChangeBody(0);
        }
    }
}

```

Figure 11: The ChangeBody method

```
// MAIN SHIP USE JUST IN CASE
//-----
// initialize ship
//shipIndex = (int)new Random().Next(10);
//shipIndex = 0;
//ship = new Ship();
//ChangeShip(shipIndex);
//ship.RepairFully();
//-----
//initialize body

//shipBodyIndex = 0;
shipBodyIndex = r.Next(Globals.PlayerShipBodyArchetypes.Count);
body = new ShipBody();
ChangeBody(shipBodyIndex);
body.RepairFully();
```

Figure 12: Comparison of the ship code and the body part code that randomly picks model



Figure 13: Randomly generated ship (1)



Figure 14: Randomly generated ship (2)

5.3 Implementing Galactic Arms Race version 3

With version two completed, it was time to add the random stats to version two to create a new GAR game. This was a much simpler to do than version two especially that the GAR code had fully understood at this point. Figure 10 showed the wing model being added to the game but in a constructor made from the “ShipWings” class. The main constructor held the data that gave the ship stats. As mentioned before some of these stats were separated from the body because they fitted better with that part (e.g. engines had speed and wings had rotation). To create the random stats a similar method for the models was used. A random variable was created and given a minimum and maximum value so that the results couldn’t be ridiculous (i.e. speed value so great you can’t control where you’re going in the game). Figure 15 shows the random variable added to the wing constructor. Comparing figures 15 and 10 shows the difference between the two versions of GAR.

```

public static void LoadPlayerShipWingArchetypes()
{
    Globals.PlayerShipWingArchetypes = new List<ShipWings>();

    Random r = new Random();
    ShipWings w;
    int ro = r.Next(10,20);

    w = new ShipWings("w1", 2000f, ro, "w1", 2500);
    Globals.PlayerShipWingArchetypes.Add(w);
}

```

Figure 15: Wing list in FileManager with random value for rotation

This was done to all stats in GAR however, one didn't work. Figure 15 shows the random value for rotation but this never changed. It was instead being overwritten by the value in the constructor. This also showed another problem as this value was a float and random.Next functions only work with ints. To account for this, a random had to be converted to a float. The best way to do this was to use the random.NextDouble function and convert it using "(float)" before the random. This problem was fixed but then another problem appeared. The NextDouble function picks a random number between zero and one and the normal rotation speed was 0.0025. The same problem as before could happen where without a maximum value the player would not have control of the ship. This function however, does not allow minimum or maximum settings to be placed on it. There is a solution to this problem however. By multiplying the function by the maximum value we would have wanted to put on it, the function will only be able to pick a number between that value and zero. The value chosen as the maximum in this case was 0.006. There is now only one problem left. The rotation speed cannot be zero otherwise the ship will not be able to rotate. To fix this a series of if statements will be made to make sure the random value doesn't reach zero or go over the maximum (just to be safe). Figure 16 shows the random generator for the rotation value and its, if statements.

```

public ShipWings()
{

    Random r = new Random();

    // base attributes
    name = "Default Ship";
    collisionRadius = 2000;
    //rotationSpeed = 0.0025f;
    rotationSpeed = (float)r.NextDouble() * 0.006f;
    if (rotationSpeed == 0)
    {
        rotationSpeed = 0.001f;
    }
    else if (rotationSpeed >= 0.006)
    {
        rotationSpeed = 0.006f;
    }
    else if (rotationSpeed <= 0.001)
    {
        rotationSpeed = 0.001f;
    }
}

```

Figure 16: Random rotation in GAR version 3

With this now sorted the ships now have randomly generated stats every time the player starts a new game. Figures 17 and 18 shows the new generated ships but new stats for each one. The stats made obvious to the player are next to the ship picture in game (bottom left)



Figure 17: Random stats 1



Figure 18: Random stats 2

With this now done, the versions of GAR that were needed for the project have been created. It is now time to start testing the new versions of GAR to gather the data necessary for reach the projects aim.

6 Evaluation

6.1 Introduction

Now that all the design and implementation was complete, it was time to evaluate the whole project with the testing that needed to be done on all versions of GAR. This included testing to make sure that each version of GAR was as bug-free as possible and the questionnaire testing in which the answers to the projects aim would hopefully be revealed. This section will also go over all the data that was found during the questionnaire testing.

6.2 The Test Procedure and its Participants

Testing in any software development process is fundamental to a projects possible success. The ultimate aim is to check the functionality and the performance of the system. With GAR versions two and three, it was absolutely necessary to test each version against the original GAR. This is because it is a projects objective that the new versions of GAR are kept as close to the original GAR as possible. If this objective is not met then the gameplay experience will be completely different, taking the attention of the players away from the procedurally generated ships. There will be black-box testing to find this out. To test and make sure that the code does truly procedurally generate its ships, white-box testing will be used for this. Black-box testing is looking at the inputs and the outputs of the system while white-box is having a closer look at what the code is doing in the system.

The black box testing was made after the implementation stage of the project. To see the list of what was tested please refer to appendix 2. From this list there were a few errors that had occurred. All of them were fixed apart from five problems which there was no time to fix before the test day.

One issue was the collision between the player and the asteroids. In the original GAR, the asteroids were sent in the direction in which the player interacted with it. With the newer versions of GAR, the asteroid would not move, causing damage to mount on the player for as long as they were colliding with it. To fix this, the original GAR and the newer versions had their "CollisionManager" classes compared. When looking at the area which showed the check for player and asteroid interaction, there was a vector in the newer versions which set the position of the asteroid after the check to (0,0,0). This meaning that the asteroid would never move as its position has never changed. This line was sharply removed and the issue was fixed from that action.

Another issue was the weapon damage was a lot weaker than the original GAR. The weapon was appearing to damage the enemies within the game at a rate of 10 rather than 55. The reason for this was that a new constructor had been made in the weapons class to load the random models. As that class was reading that new constructor, it had lost the weapon damage value therefore going to default. Instead of adding a new variable to the constructor and changing all the references to it with the new data, it was felt that changing the default to 55 was the best way to tackle this problem. This ultimately fixed the issue and the game now was back at the original value

One more issue (this time only with version 3) was the rotation speed. This has been mentioned before in the implementation part of the project but the minimum and maximum values were not just random ones picked out of thin air. When the rotation speed was added testing was done to try and find the best speeds to make sure the speed changed but it was never too slow or too quick. It didn't take too long to find a suitable range in which version 3 now uses.

As for white box testing, White box testing was used to just make sure that the ships were being procedurally generated and that the stats of the ship were random as well. Please refer to appendix 3 for the tables of metrics taken from versions one and two. There was a problem with the procedurally generated ships that was noticed from old metrics. Every ship part generated was the same number in the list (i.e. body part 7, wing part 7, head part 7 etc.) This was obviously not procedural as this meant that instead of 1,000,000 ships, the player only got 10. Luckily the fix for this was easy to find for the reason for this problem was that the random values were being initialized too quickly. The speed of which they were accessed was so fast that the random seed never got to change in time. From this, the random value was initialized earlier and this made the ships truly generate procedurally.

Once this was done, a message was sent to people at the University of Lincoln's games students. This message asked them to come down and test the versions of this game of GAR and answer a few questions about their experiences with the ship in each one. Four students had turned up for the event and the testing could begin.

6.3 The Experiment and its Results

The experiment to reach the projects aim was designed to be as simple as possible for the testers. The participants would be asked to play three different versions of GAR, one being the original GAR and the other two with generated ships. The participant will only play the first GAR version once but will play the other versions two times. Each play through should last approximately five minutes (amounting to roughly 25 in total). During each play through, game metrics (such as the

ship parts) will be recorded for later analysis. Following each play through, participants will be asked to fill out the two questions per play-through of a 16 question questionnaire, with an extra two questions at the end of each last play through and a final extra two questions when all the play-throughs have been done. The reason for this is that some of the questions ask for feedback regarding each ship generated, while the rest two ask about the participant's experience with the ships in general. In addition to the questions, the participant will be allowed to express their thoughts and opinions of the game and ships at the end of this questionnaire. The participants are reminded that questions asked in this questionnaire will require NO personal information and, as such, NO personal information will be gathered from anyone. The questions will only concern the game and the ships each participant experienced.

The results gathered from the questionnaire have been summarised to give a good explanation of what was discovered from the event. The questionnaire can be found as appendix 4 in the appendices sections. For showing how the testers rated their experiences with each version of GAR, a number is used to represent each rating so they can be showed on charts. These numbers are:

1. Extremely disagree/bad
2. Quite disagree/bad
3. Slightly disagree/bad
4. Neither
5. Slightly agree/good
6. Quite agree/good
7. Extremely agree/good

6.3.1 Question 1 & 2

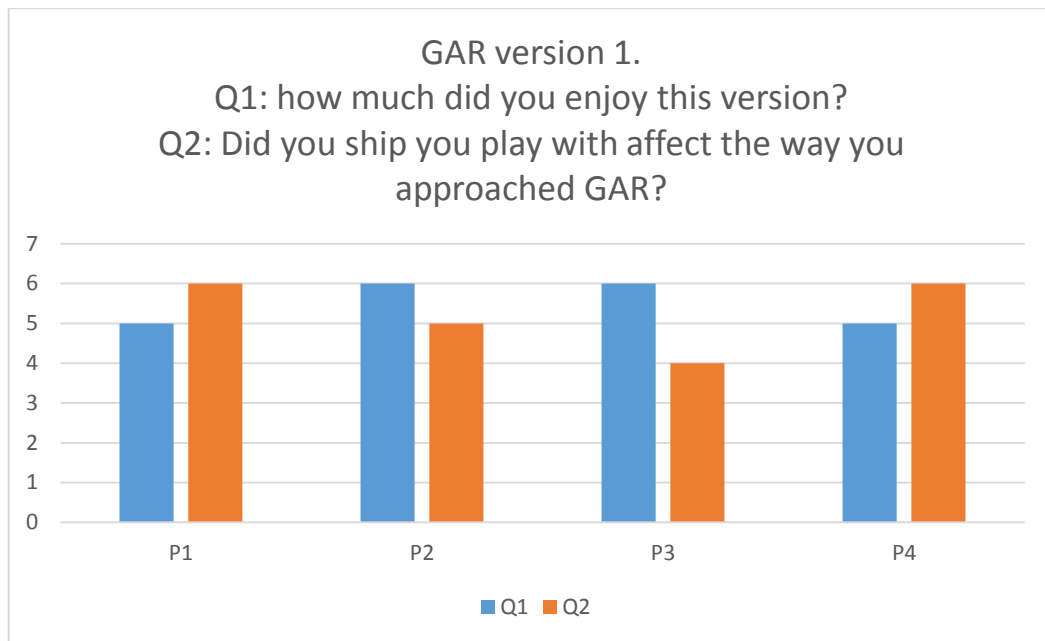


Figure 19: Question One & Two Results - Bar Chart

The results of this showed that the people who played the original version of GAR did like the game and agreed that the ship affected the way they approached GAR. All participants thought the game was enjoyable (two thought it was quite good while the other two only thought it was slightly good). In regards to if the ship affected the way they played the game, the results seem to suggest they mostly thought so (two quite agreed, one slightly agreed and the other was neutral). These results are good to have so they can be compared to what these testers experience with the projects versions of GAR.

6.3.2 Questions 3 – 6

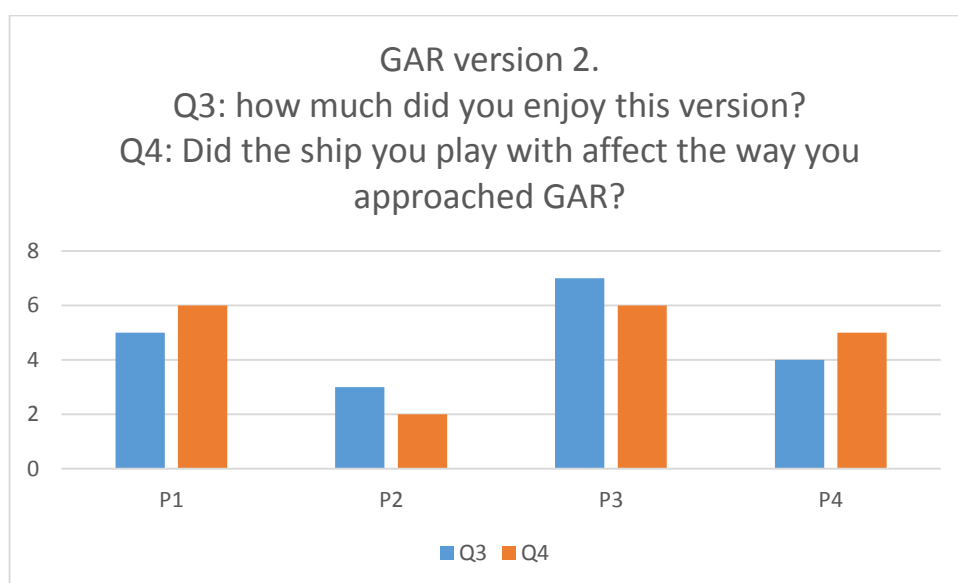


Figure 20: Question Three & Four Results - Bar Chart

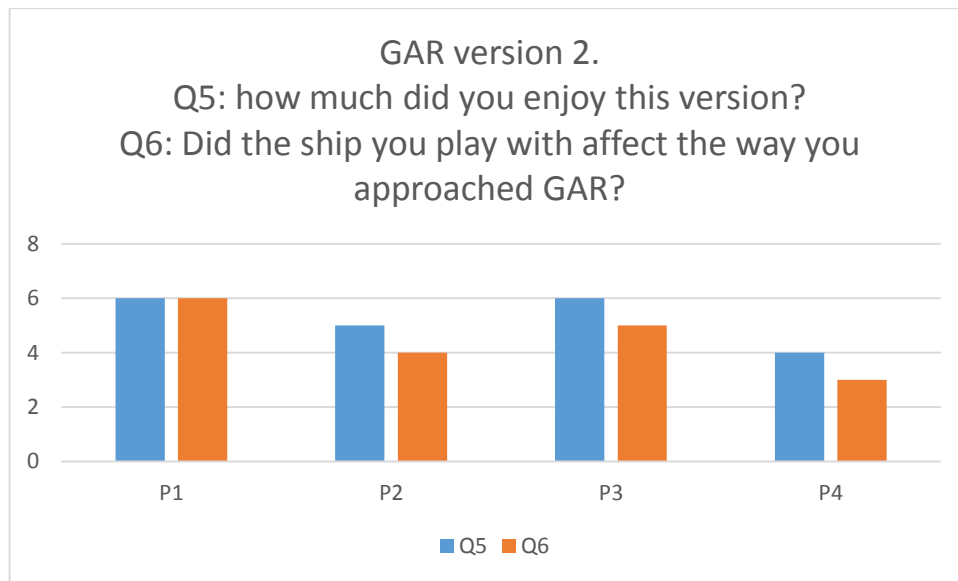


Figure 21: Question Five & Six Results - Bar Chart

These questions were about version two of GAR. The results here for figure 20 suggest a mixed reception to the ships that were generated for each person. P1 seemed to enjoy the game only slightly in the first play-through, also quite agreeing with the claim that the ship changed the way he approached GAR. P2 did not like the ship given claiming that play-through was slightly bad and the ship did not change P2 method of player the game. P3 on the other hand seemed to have extremely enjoyed version two and felt the ship did impact how P3 played GAR. The final participant P4 seem to not enjoy this game as much claiming it was neither good nor bad (according to results). Although, P4 did say it changed the approach that the participant had with GAR.

In the second play-through (figure 21), P1 preferred the ship that was generated and still felt it changed the way he approached the game. P2 had a much better experience claiming that GAR was enjoyable this time though stating the ship didn't change his approach in this play-through. P3's experience overall wasn't as good this time although found it still quite enjoyable, the ship only slightly changed the way P3 played. P4 found this play-through the worst claiming this version wasn't bad but not good either plus, stating that the ship did not affect the P4 played the game.

6.3.3 Questions 7 & 8

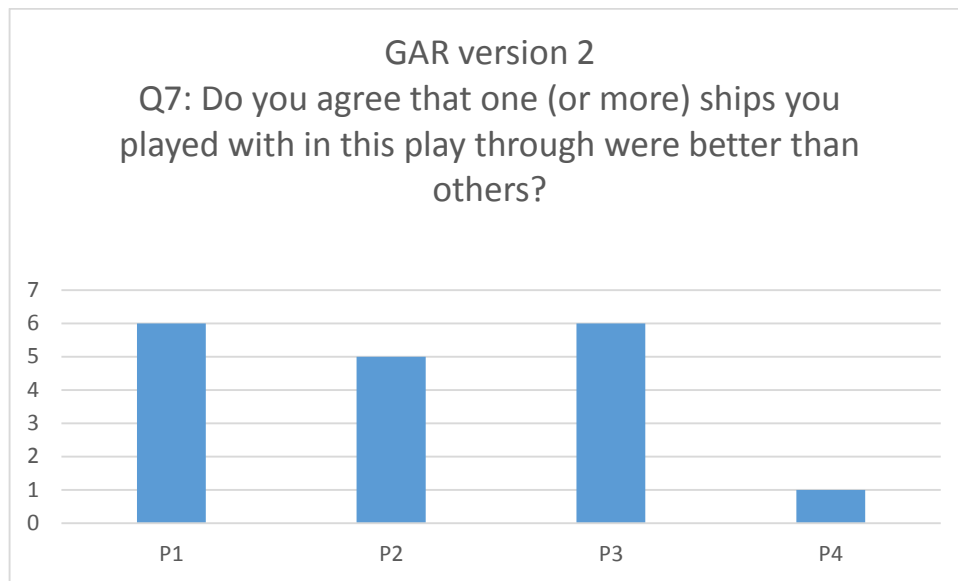


Figure 22: Question Seven Results - Bar Chart

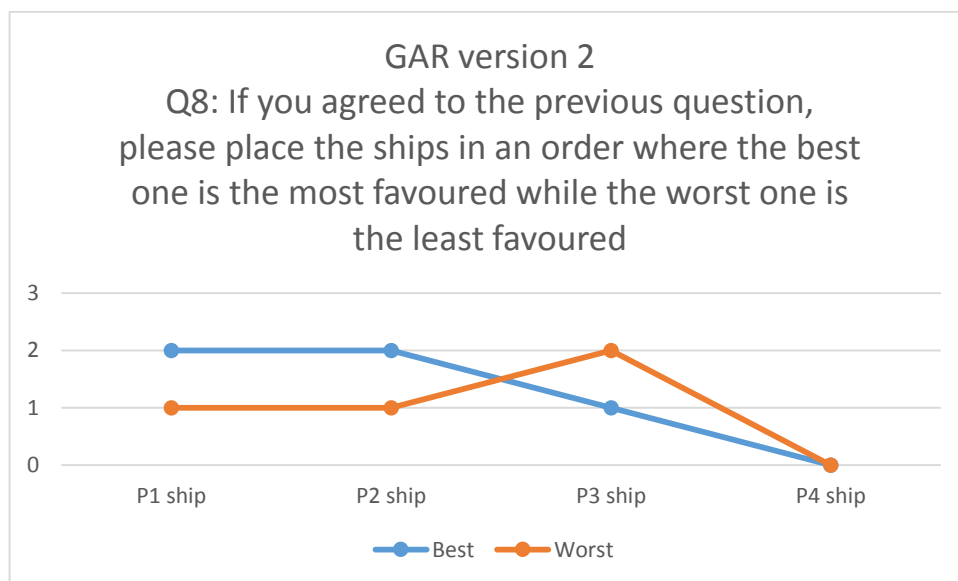


Figure 23: Question Eight Results - Line Chart

These questions were to see if the testers preferred a particular ship and if they did, show which one they preferred. According to the Figure 22, P1 and P3 quite agreed with the questions statement. P2 only slightly agreed but P4 extremely disagreed thus stating that person did not believe any ship was better. As P4 did not agree to any of them, the next test did not need an answer from that person. Figure 23 shows that P1 and P2 both said that their second ship was the best and P3 picked ship one as the personal favourite.

6.3.4 Questions 9 – 12

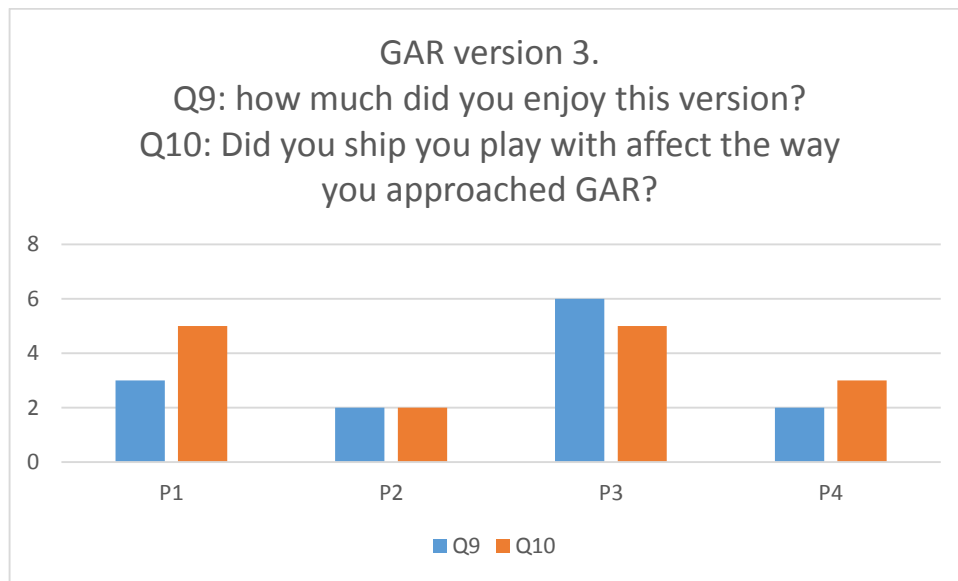


Figure 24: Question Nine & Ten Results - Bar Chart

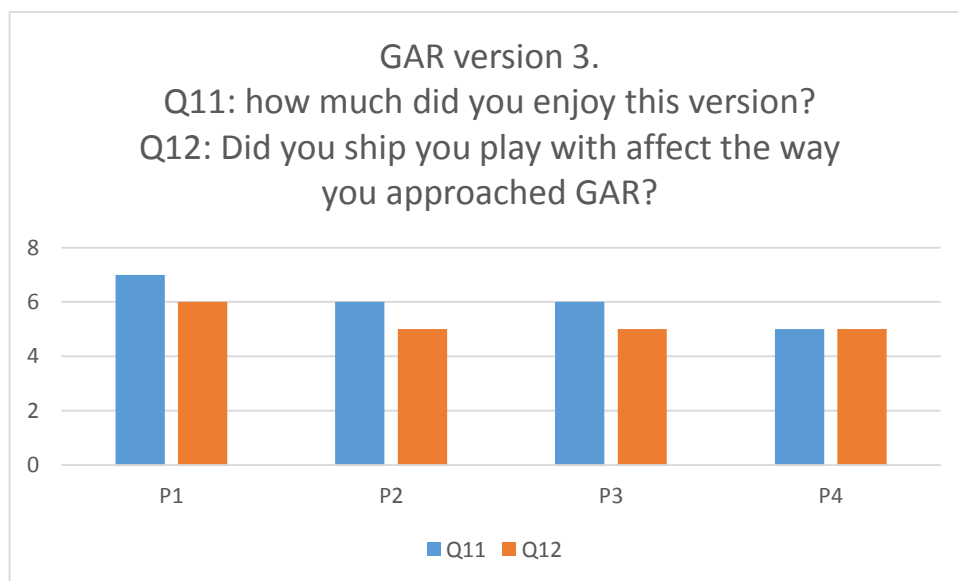


Figure 25: Question Eleven & Twelve Results - Bar Chart

These questions were about version three of GAR. The results here for figure 24 suggest a bad reception to the ships that were generated for each person. P1 seemed to not enjoy GAR by quite a bit, but slightly agreeing with the claim that the ship changed the way he approached the game. P2 (similar to P1) did not like the ship given as well also saying that this play-through was quite bad. The ship did not change P2's method of playing the game. P3 on the other hand seemed to have quite enjoyed version two and felt the ship had an impact on how P3 played GAR but only slightly. The final participant P4 seem to not enjoy this game as much as P1 and P2 and claiming it didn't affect the way that person played.

In the second play-through (figure 25), P1 had a much better experience, thinking the ship that was generated was extremely good and felt it changed the way he approached the game. P2 had a much better experience claiming that GAR was quite enjoyable this time though stating the ship didn't change his approach in this play-through. P3's experience overall was very much the same as P2s. The only difference was that P3 felt nothing had changed between the two play-through. P4 found this play-through better than the previous one although saying it was only slightly good. P4 also stated that this new ship had slightly changed the way the person played the game.

6.3.5 Questions 13 & 14

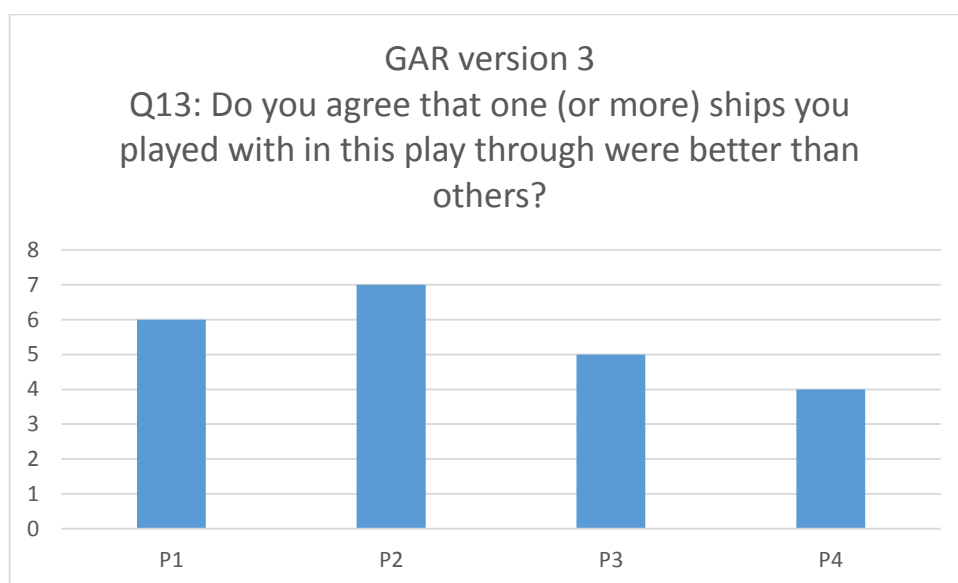


Figure 26: Question Thirteen Results - Bar Chart

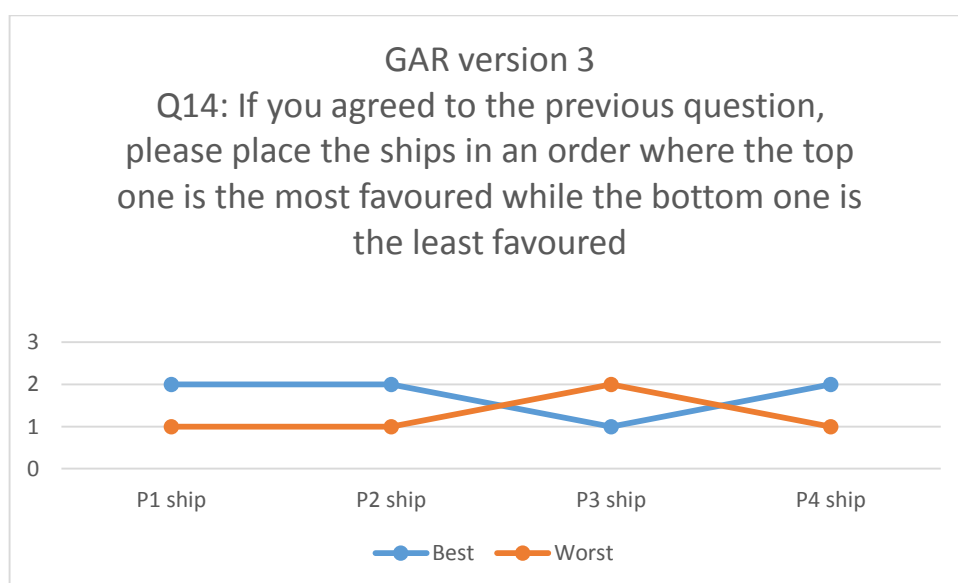


Figure 27: Question Fourteen Results - Line Chart

These questions were to see if the testers preferred a particular ship and if they did, show which one they preferred. According to the Figure 26, P1 quite agreed with the questions statement. P2 extremely agreed. P3 only slightly agreed and P4 neither agreed nor disagreed. Figure 27 shows that P1, P2 and P4 said that their second ship was the best and P3 picked ship one as their personal favourite.

6.3.6 Questions 15 & 16

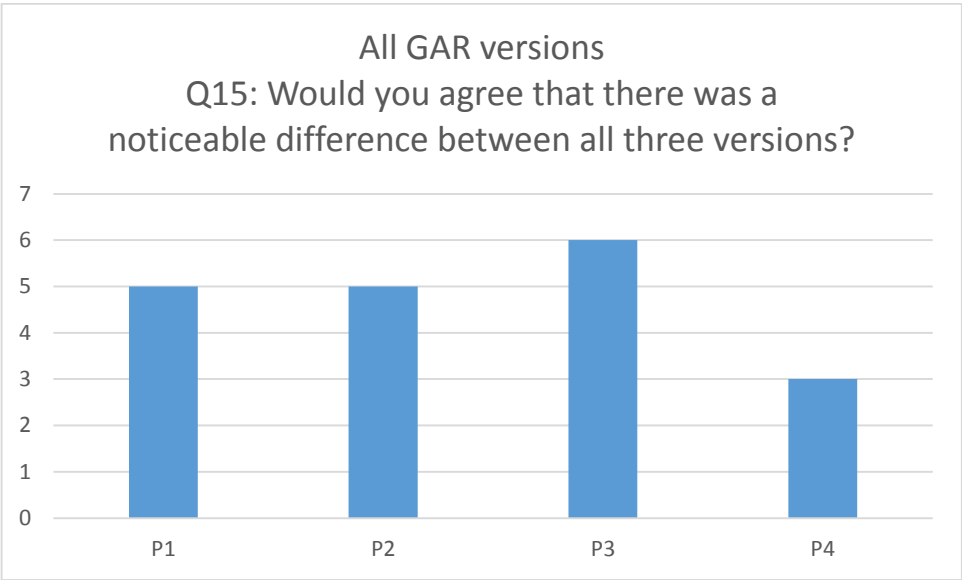


Figure 28: Question Fifteen Results - Bar Chart

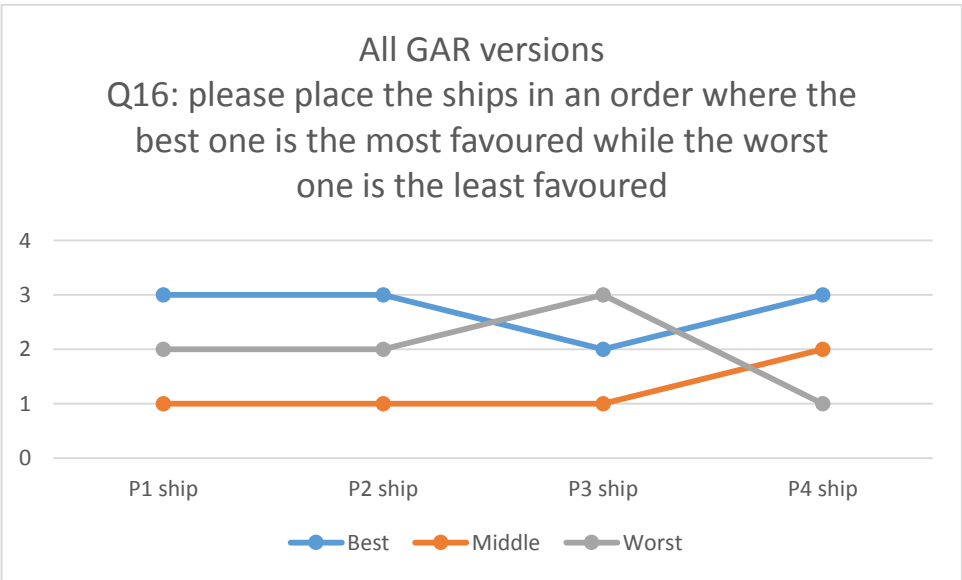


Figure 29: Question Sixteen Results - Line Chart

These questions were to see if the testers noticed a difference between the versions and show which one they preferred. According to the Figure 28, P1 and P2 slightly agreed with the questions statement. P3 Quite agreed with the question and P4 slightly disagreed. Figure 29 shows that P1 and P2 said that version three was the best with version one second and version two as the worst. P3 picked version two as the best, version one as the middle choice and version three as the worst. Finally P4's order was version three as the best, version two as second and version one as the worst. Now that the question data has been revealed, the code metrics data gathered from these players will be looked at.

6.3.7 Code Metrics Version Two

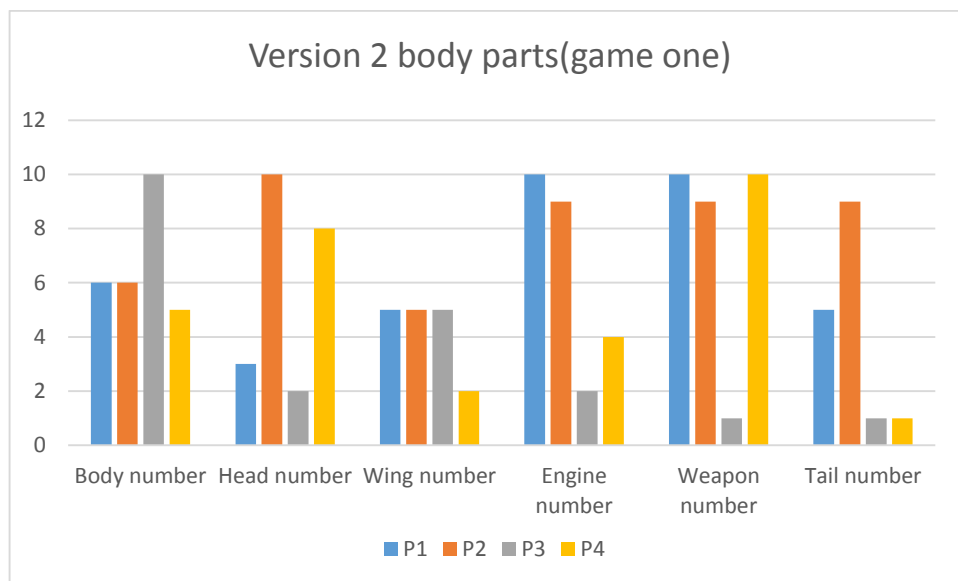


Figure 30: Models Generated 1

The Code Metrics of GAR version two show the models that were generated. Figure 30 shows which players got what models. P1 and P2 got the same body and the same wings along with P3. P1 also got the same weapon as P4 and P3 got the same tail as P4. Comparing this with figure 20 or 36, it can be seen that P1 and P2 had similar parts yet had a different experience of the game. Looking at figure 27, it can be seen that there was no difference in the stats of the player's ships therefore the results of figure 20/36 are purely on the ships appearance.

When looking at figure 31 for the models used in the second play-through of GAR version two, players have similar models again. P1, P2 and P3 all have the same ship body. P3 and P4 have the same head and tail. P1 and P2 have the same engine. The final observation is P2 and P3 have the same weapon. When looking at figure 21/37 with figure 31, there may have been a similar feel that body five was good as all three players enjoyed the game. Again, the only difference between these ships are the models as all the stats were the same (figure 32).

Comparing both figures 30 and 31 with figures 22 and 23 gives some suggestive information. Firstly, P4 must have really disliked the experience with the ship. As the ships stats were the same, it is safe to assume that this player did not like the models that created his/her craft therefore ruining the experience of the game. Apart from that the other players seemed to enjoy at least one of the ships given to them. Judging this with figures 20/36 and 21/37, gives us the impression but P3 and P1 both had good experiences with both their ships. P3 on the other hand had a terrible first ship but a good second ship. These are observations of GAR version two. Version three however show some more interesting results

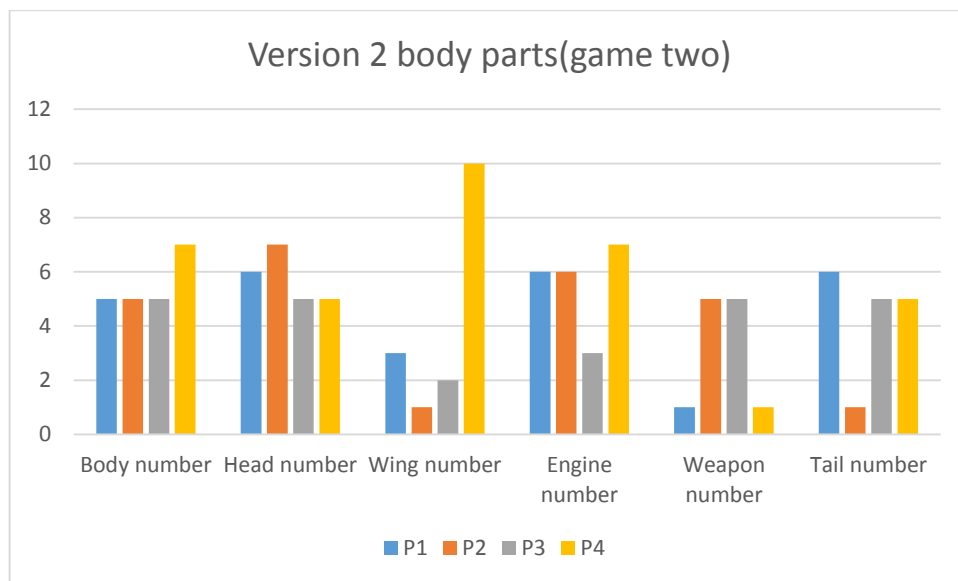


Figure 31: Models Generated 2

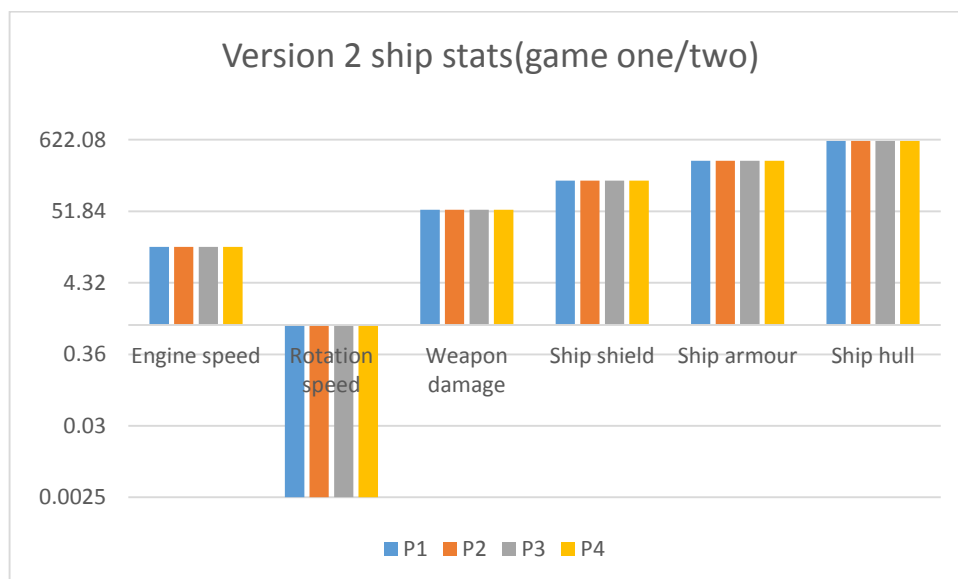


Figure 32: Ship Stats 1

6.3.8 Code Metrics Version Three

In this version, the models were quite ranged when looking at figures 31 and 32. The only similarities were P3 and 4 with the same body part, P1 and 2 with the same head number, P1 and 3 had the same wings and finally P2 and P3 had the same tail. Looking at figure 29, all the players' bar P3 seem to dislike their ship. Now that the stats change in this version, we can look at figures 34 and 35 to see if it's because the stats were bad. Comparing these with figure 24/41 shows that there wasn't a lot of difference between the ships, in fact there isn't too much difference between P2's ship and P3's. This would suggest that the players did not like the models rather than how their ship acted in game.

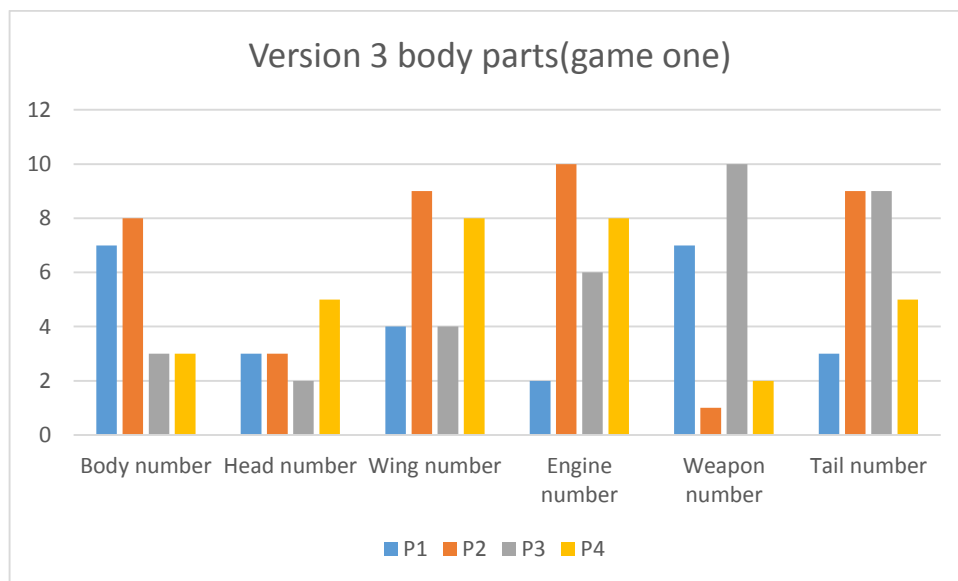


Figure 33: Models Generated 3

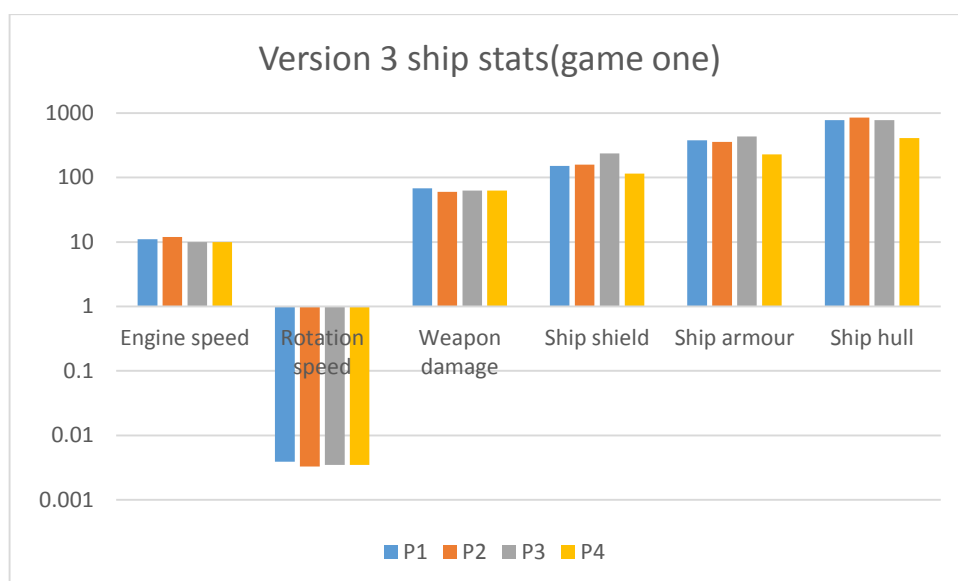


Figure 34: Ship Stats 2

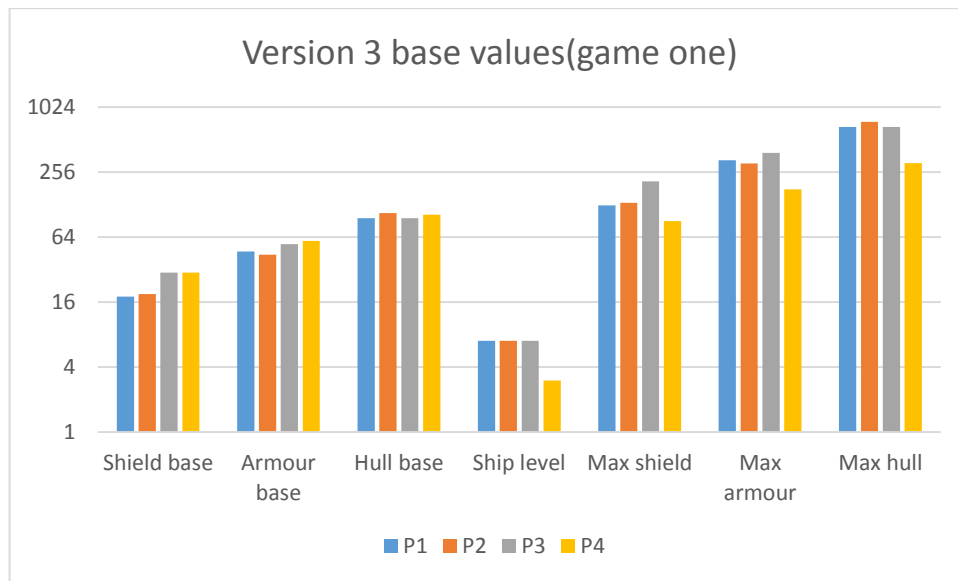


Figure 35: Base Values 1

For the final test, deciding metrics were pulled out this play-through. Firstly the least amount of duplicate models were generated according to figure 31. P3 and P4 had the same engine, P1 and P4 had the same body part and they also had the same wings. Just like in the first play-through, the stats changed and just by looking at figure 33 and 34 can a great difference between the ships be seen. However, when comparing with figure 25/42, it becomes obvious that every player enjoyed their ship. Noticeable stats can be taken from figure 39 are P3's. This is because apart from the speed which is quite high, the rest of the stats are quite bad. The one which sticks out the most is the rotation which is very slow as it's very close to 0.001. Yet looking at figure 25/42, the player still enjoyed playing with the ship even admitting that it changed his style ever so slightly. This shows that the stats did have an effect on the players and the way they played. The fact they still enjoyed the game version showed that they still like the ships.

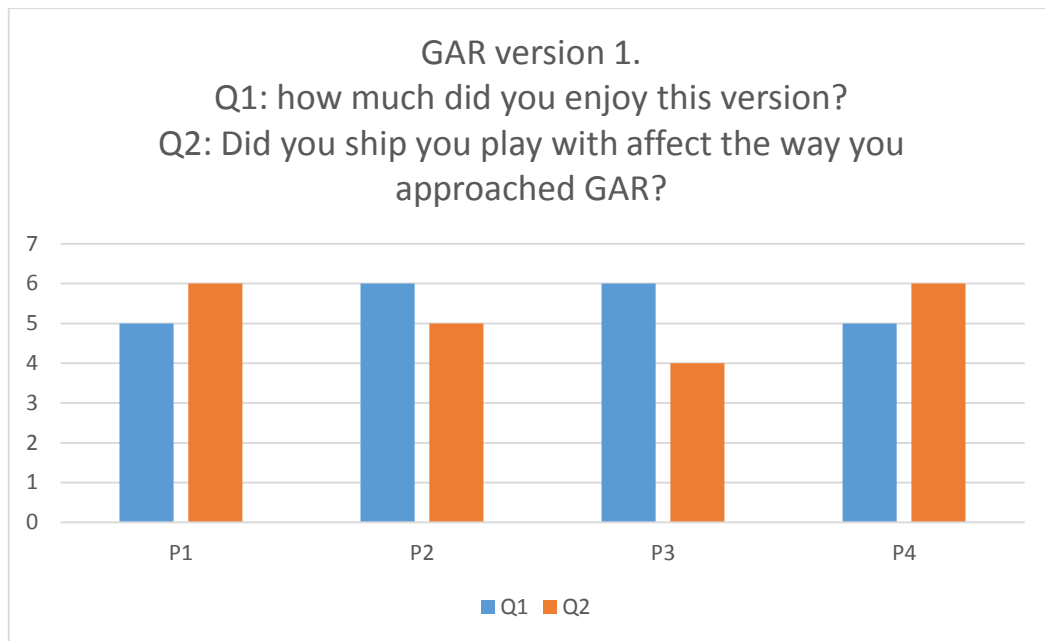


Figure 36: Question One & Two Results - Bar Chart

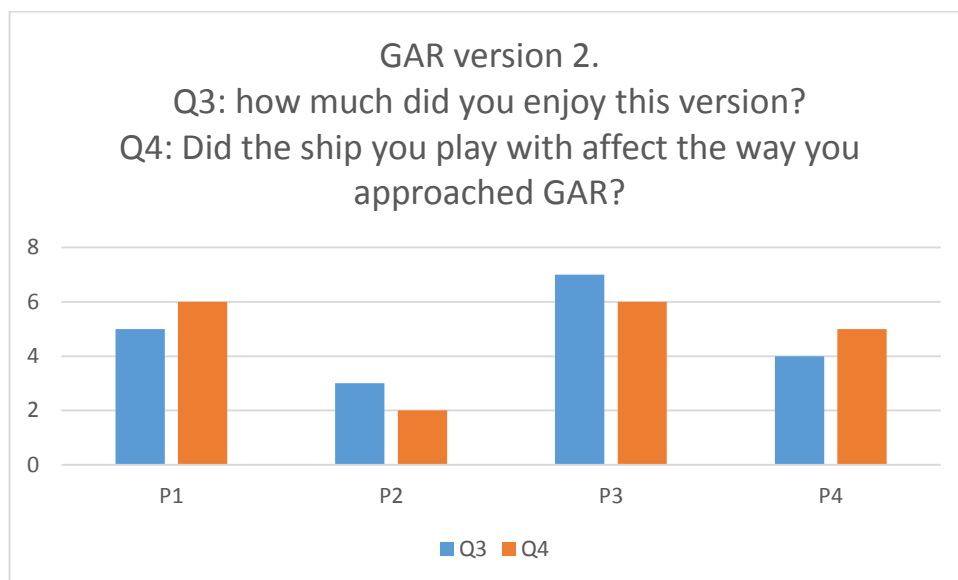


Figure 37: Question Three & Four Results - Bar Chart

Comparing these results with figures 26 and 27 seem to show that most of the players did think the ships were better. P4 did agree with this but neither disagreed which could have meant he/she could have like them both. Comparing figures 24/41 and 25/42 however show that this is not the case. Figure 27 shows that everyone preferred their second ship apart from P3 who chose the first ship. Figures 24/41 and 25/42 deem this defiantly possible as both ships has the same rating. Therefore the conclusion on this is that P3 defiantly preferred the visual look of ship 1.

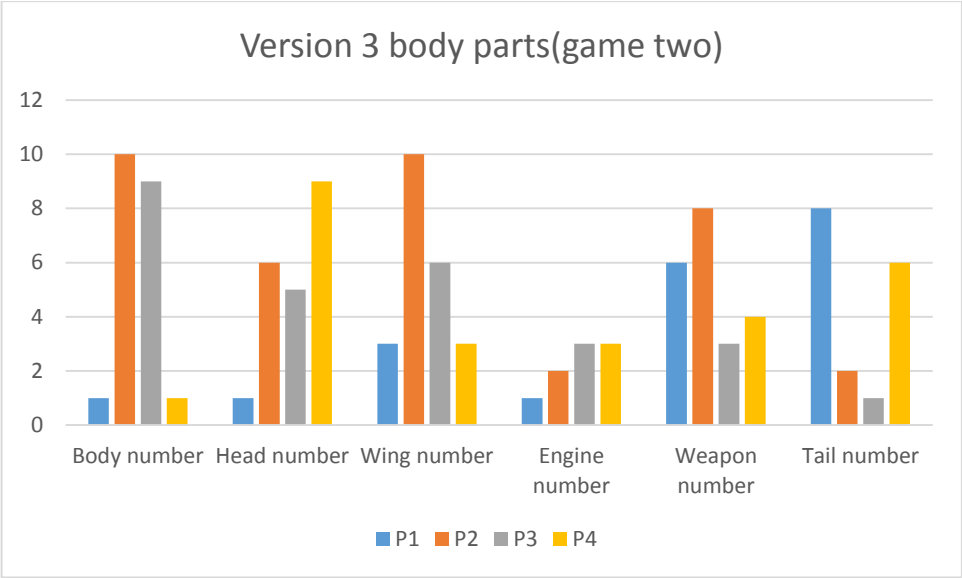


Figure 38: Models Generated 4

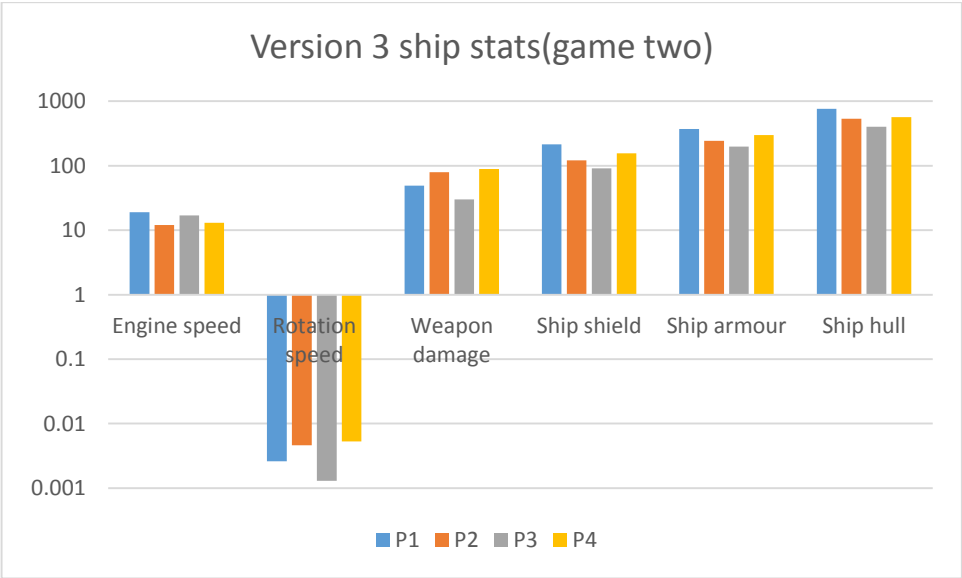


Figure 39: Ship Stats 3

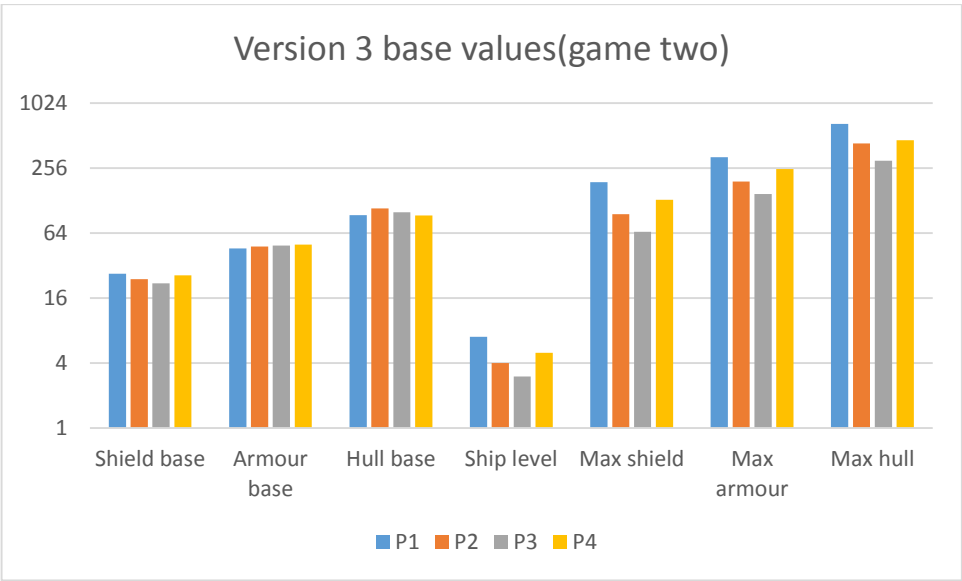


Figure 40: Base Values 2

The differences were revealed to the players at the end of the event and were met with some interesting responses. The players seemed to not really notice the difference but did claim that they did think their ships were slower at some points. P2 had spoken about his/her ship and claimed “I preferred the look of the ship in version 3, but I preferred the speed of the ship in version 2.” This information when compared with figures 28 and 29, seems to contradict one another. From this, it is possible that the stat changes had an affect with the players and they didn’t know about it. The actual numbers for all the graphs are in appendices (see appendix 5). With this knowledge collected, it is time to conclude the project.

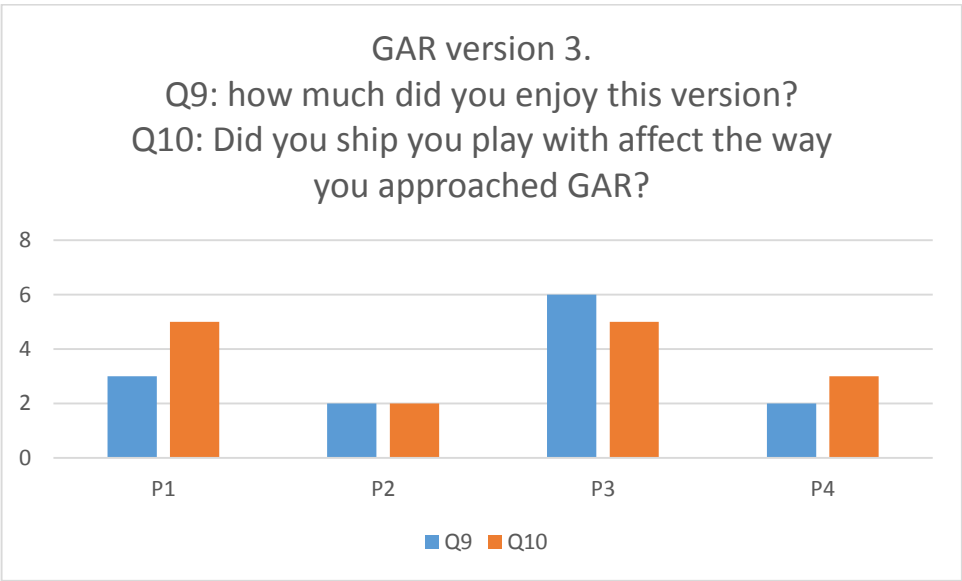


Figure 41: Question Nine & Ten Results - Bar Chart

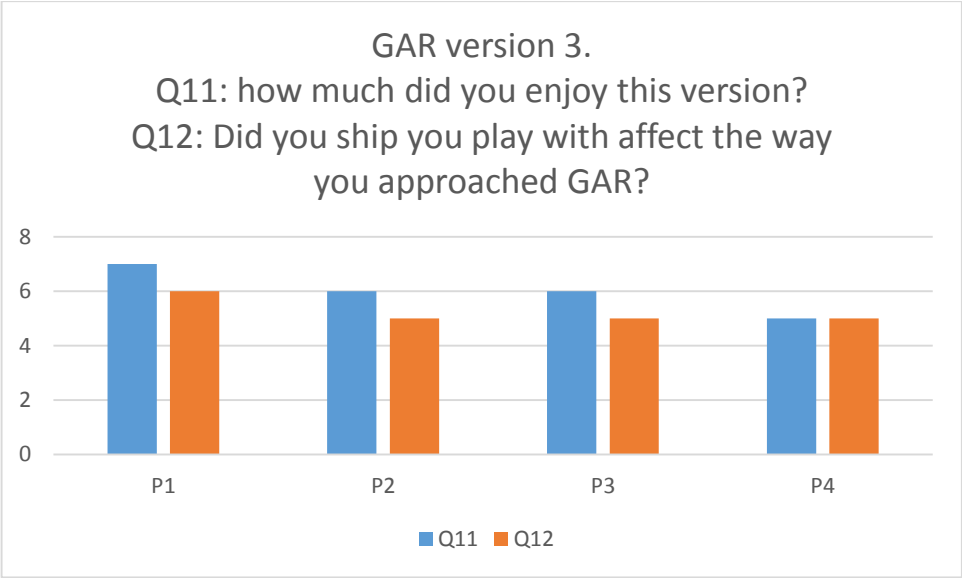


Figure 42: Question Eleven & Twelve Results - Bar Chart

7 Conclusion

Looking at all the research gathered from the testing session, it is believe that the aims and objects mentioned earlier (see methodologies) have all been answered successfully. The project has reached its end and an answer has been found.

Firstly, all objectives were reached to reach the main aim. The GAR code was learnt early so the project versions could be made. This proved to be very successful as not too many issues were found when making the new version. The testing session was a complete success with valuable data collected plus the participants enjoyed their involvement which is always nice to see. The final objective was keeping the new versions as close to the old version as possible. It was exceedingly successful as it appeared that all gameplay dynamics had been kept or any noticeable ones at least. With only four bugs still in the versions, it was very good to limit it to that. Another pleasing aspect was that none of these bugs appeared during the testing, which is always helpful to get no data which could be false.

Secondly, two new versions of GAR were created successfully for the project. GAR version two had the ability to change its models every time the game application was activated. This enabled the player to get a different ship every time they played that version. This was a very successful implementation with the program being almost bug free. GAR version three had the ability to change the stats of the ships that the player had from a good range so some stats could be worse than the original or better. This was a very successful implementation of the idea has this version had no more extra bugs in it than it already had before from version two. So, do procedurally generated space crafts for the game demo “Galactic Arms Race” make gameplay more enjoyable?

It was found that procedurally generated space-crafts (or space ships) certainly had an effect on the players. Figure 29 showed that GAR version three was the most favoured while GAR version two was the least popular. This means one of two possible answers. Either that the ships players had for version two were disliked the most or that the change of stats from version 3 had an effect on the players. Looking at figures 20/36 and 21/37, it seems that players did like ships from version two. In fact if those figures are compared figures 24/41 and 25/42, P2’s worst ship came from the first play-through of version three. Looking at Figure 25/42 overall, it was the only time everyone’s experience was positive and yet the stats were very different for each ship. Now although players such as P2 did confirm that they preferred the look of their ship in that version. The experience of the game still different and this particular player only really had a positive experience in the way he/she played when the original was played and that version. With all this in mind, the answer to the project is simple. Procedurally generated ships can make gameplay

more enjoyable to the player but, not just on its own. If a player does not like their ship then the experience of playing the game will suffer, but with the changing stats a whole new element comes into play where players change their style of play. The stat change is enough to make sure that every ship created by the system feels different as well as looks different to the user. This is what was found and this is the best way to make procedurally generated player avatars according to this project.

8 Critical Reflection

Overall, I feel that the project was successful but the path to completion was not straight forward. There were some challenges that were met along the way which tested my skills as well as my patience. I will discuss further into what problems were met and how these were dealt with. I will also go into depth on how the project was managed.

While I have been at university, I have learned valuable skills that have been used during this project. From my days as project leader in the module group project, I had gained important skills such as planning and organising a whole project. The chance to do this independently gave me the ability to put what I know and believe to the test. This experience has certainly expanded my understanding on project management and how it should work.

I tried to manage the project by separating it into different stages so a linear approach could be taken towards the project. I tried to do this while keeping away from the waterfall methodology structure at the same time. The project was split into the following stages:

- Planning
- Learn how GAR code works
- Research
- Design
- Test space-craft
- Create GAR version 2
- Create GAR version 3
- Testing
- Documentation

In the planning phase, I designed the Gantt chart so it would be clear now the timeline of the project should look. It became clear that in some cases I had sometimes given myself too much time (which isn't bad) but I had mostly underestimated some tasks. My answer to this was to keep monitoring the Gantt chart and make changes to it when a task went over or under the time limit. When looking at GAR, I had given myself two weeks to try and understand how the game worked internally. When I had made this chart, I had looked at GAR properly yet. This was a decision that would have consequences. I was shocked to have discovered over 150 classes and over 60,000 lines of code in a demo. To fix this problem, the Gantt chart was redesigned so that it ran alongside other tasks such as research. Research was fairly straight forward. The tasks that

had been set for this area were completed before the set deadlines. This did mean that the Gantt chart had to change but it was pleasing to see.

Designing the ships and questionnaire proved to be harder than I thought. However, the only issue I had with this stage was the designing taking longer than I thought which meant that another Gantt chart had to be made. Now making the test space-craft proved to be the most frustrating part of the project. In my opinion it would have been implementation but if I had 20/20 foresight, I would have saved myself a lot of time and trouble. I made a basic ship to judge the size of models in GAR. The hope of this was to see how big the overall ship should be so each part could have its own scale limitations. When the ship was added, it would not appear. The previous model for the original GAR space ship was gone but nothing had replaced it. I re-scaled, changed the colour and even downloaded models to use instead of mine, nothing changed. It would take nearly a month to fix the issue! The problem in the end was blender and XNA linking. I had very little experience with blender prior to the project and my inexperience had cost me. I had forgotten to apply the scale and the location to my model. This meant that the model was on the screen but was so tiny that the user would not be able to see it. Once this was fixed, a new Gantt chart was made.

To see any issues that happened with GAR versions two and three please see implementation. The final problem came from testing. As certain issues with GAR versions two and three had happened, it held back the testing until the week of this papers deadline. Apart from being frustrated by the problems, the Gantt chart also had to change because of this. The testing had to be a little rushed unfortunately. This was to do with the room that the testing was being organised in being used for an exam. I had to make sure all the tester played all the attempts necessary to gain data. Luckily, I was able to get the data and thank the tester for their contribution but it was very close as the people who organised the exam has just asked for the room to be cleared.

On the whole, this project has tested my skills to the limit and as such I have learned a lot from this experience. I have had to learn how to read and understand other people's code which in any area of computer science is a great attribute. I have learnt how to make a questionnaire and what types of questions are available to put in it. I have learnt how deep a contention can be to a player's avatar thanks to my research. Finally, I have learnt that no matter how hard you try and plan ahead in case issues occur in the project, you can never truly expect anything and as such always be prepared to make a new Gantt chart.

Although the project was successful, I feel that I could do this project again and the end result would be of a vast improvement. This project had made me put my skill set to good practice so I can see where I need to improve as an individual. The whole experience will help me with any future career that I wish to go into. My confidence has greatly increased along with my research skills which will always help in any industry of work. So I am happy with I have accomplished in this project but, I look forward to the next project in eager anticipation, hoping to improve skills even further just like in this one.

9 References

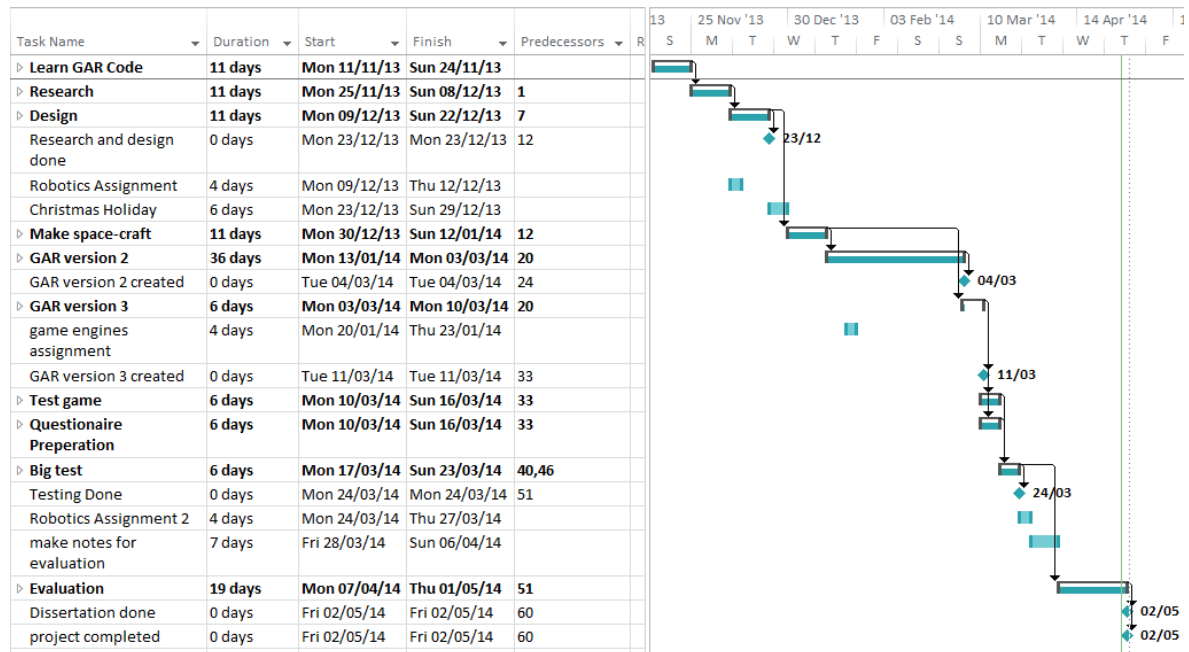
1. Adams(2006)Slaves to Armok: God of blood chapter II: Dwarf Fortress[Online game]Bay 12 Games. Available from <http://www.bay12games.com/> [Accessed 30/04/2014]
2. Braben and bell(1984)Elite[Video game]Cambridge, England: Acornsoft.
3. Browne, C. 2008, Automatic generation and evaluation of recombination games,
4. Burgess, T.F. 2001, "Guide to the Design of Questionnaires", *A general introduction to the design of questionnaires for survey research*, .
5. Hastings, E.J., Guha, R.K. & Stanley, K.O. 2009a, "Automatic content generation in the galactic arms race video game", IEEE Transactions on Computational Intelligence and AI in Games, vol. 1, no. 4, pp. 245.
6. Hastings, E.J., Guha, R.K. & Stanley, K.O. 2009b, "Evolving content in the galactic arms race video game", Computational Intelligence and Games, 2009. CIG 2009. IEEE Symposium onIEEE, , pp. 241.
7. Hastings, E.J. & Stanley, K.O. 2010, "Interactive genetic engineering of evolved video game content", Proceedings of the 2010 Workshop on Procedural Content Generation in GamesACM, , pp. 8.
8. Hunicke, R., LeBlanc, M. & Zubek, R. 2004, "MDA: A formal approach to game design and game research", Proceedings of the AAAI Workshop on Challenges in Game AI, pp. 04.
9. Legendary games(2009)Galactic Arms Race[online game] Florida, USA: computer science department at the university of central Florida. Available from <http://galacticarmsrace.blogspot.co.uk/> [Accessed 21 April 2014].
10. Mateas and Stern(2005)Façade[online game] Available from <http://www.interactivestory.net/> [Accessed 30/04/2014].

11. Mojang(2011)Minecraft[online game] Stockholm, Sweden: Mojang. Available from <https://minecraft.net/> [Accessed 30/04/2014]
12. Pedersen, C., Togelius, J. & Yannakakis, G.N. 2009, "Modeling player experience in super mario bros", Computational Intelligence and Games, 2009. CIG 2009. IEEE Symposium on IEEE, , pp. 132.
13. Pedersen, C., Togelius, J. & Yannakakis, G.N. 2010, "Modeling player experience for content creation", Computational Intelligence and AI in Games, IEEE Transactions on, vol. 2, no. 1, pp. 54-67.
14. Roden, T. & Parberry, I. 2004, "From artistry to automation: A structured methodology for procedural content creation" in Entertainment Computing–ICEC 2004 Springer, , pp. 151-156.
15. Royce, W.W. 1970, "Managing the development of large software systems", *proceedings of IEEE WESCON* Los Angeles, .
16. Skumar (2014a) What is Spiral model- advantages, disadvantages and when to use it[online]. Available from: <http://istqbexamcertification.com/what-is-spiral-model-advantages-disadvantages-and-when-to-use-it/> [Accessed 30 April 2014]
17. Skumar (2014b) What is prototype model- advantages, disadvantages and when to use it[online]. Available from: <http://istqbexamcertification.com/what-is-prototype-model-advantages-disadvantages-and-when-to-use-it/> [Accessed 30 April 2014]
18. Skumar (2014c) What is waterfall model- advantages, disadvantages and when to use it[online]. Available from: <http://istqbexamcertification.com/what-is-waterfall-model-advantages-disadvantages-and-when-to-use-it/> [Accessed 30 April 2014]
19. Skumar (2014d) What is Agile model- advantages, disadvantages and when to use it[online]. Available from: <http://istqbexamcertification.com/what-is-agile-model-advantages-disadvantages-and-when-to-use-it/> [Accessed 30 April 2014]

20. Smelik, R., Tutenel, T., de Kraker, K.J. & Bidarra, R. 2010, "Integrating procedural generation and manual editing of virtual worlds", *Proceedings of the 2010 Workshop on Procedural Content Generation in Games*ACM, , pp. 2.
21. Togelius, J., De Nardi, R. & Lucas, S.M. 2007, "Towards automatic personalised content creation for racing games", *Computational Intelligence and Games*, 2007. CIG 2007. IEEE Symposium onIEEE, , pp. 252.
22. Togelius, J. & Schmidhuber, J. 2008, "An experiment in automatic game design", *Computational Intelligence and Games*, 2008. CIG'08. IEEE Symposium OnIEEE, , pp. 111.
23. Togelius, J., Yannakakis, G.N., Stanley, K.O. & Browne, C. 2010, "Search-based procedural content generation" in *Applications of Evolutionary Computation* Springer, , pp. 141-150.
24. Togelius, J., Yannakakis, G.N., Stanley, K.O. & Browne, C. 2011, "Search-based procedural content generation: A taxonomy and survey", *Computational Intelligence and AI in Games*, IEEE Transactions on, vol. 3, no. 3, pp. 172-186.
25. Van Looy, J., Courtois, C., De Vocht, M. & De Marez, L. 2012, "Player identification in online games: Validation of a scale for measuring identification in mmogs", *Media Psychology*, vol. 15, no. 2, pp. 197-221.
26. Vorderer, P., Hartmann, T. & Klimmt, C. 2003, "Explaining the enjoyment of playing video games: the role of competition", *Proceedings of the second international conference on Entertainment computing*Carnegie Mellon University, , pp. 1.

10 Appendices

10.1 Appendix 1: Gantt Chart



10.2 Appendix 2: Black-Box Testing Data

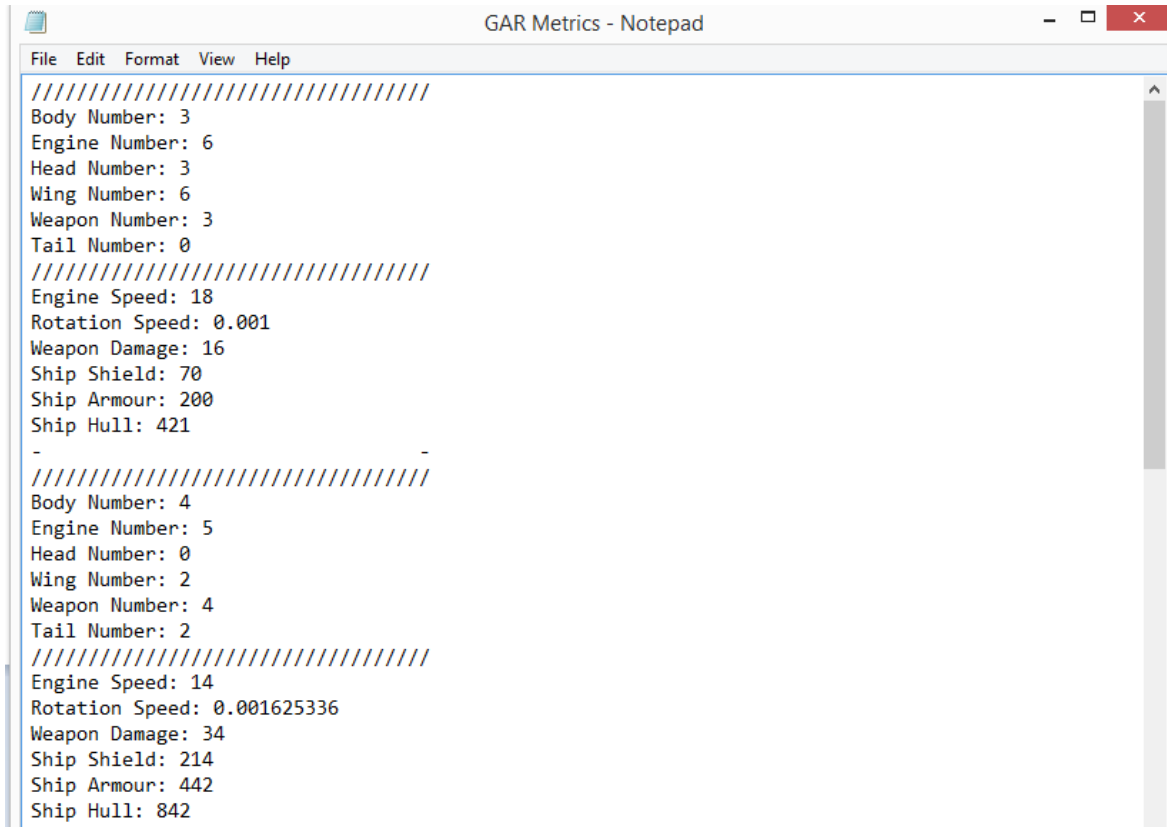
Test	Input	Output	Pass/Fail
New game	Click on single player	Game loads with new ship	Pass
New game	Click on defence mode	Game loads with new ship	Pass
New Game	Player models appear on little screen	Only the body papers	Fail
New Game	All models on screen are visible	Left wing is dark	Fail
Click menu buttons	Click on ship stats	Crash screen appears	Fail
Click menu buttons	Click on weapons screen	Weapons screen loads	Pass
Click menu buttons	Click on Galaxy map screen	Galaxy map loads	Pass
Click menu buttons	Click basic options	Basic options loads	Pass
Click menu buttons	Click Input	configuration screen	Pass

	configuration screen	loads	
Click menu buttons	Toggle music on/off	Music goes on or off	Pass
Click menu buttons	Click achievements screen	Achievements load	Pass
Click menu buttons	Click missions button	Mission appears	Pass
Player move	Move left	Ship goes left	Pass
Player move	Move right	Ship goes right	Pass
Player move	Move forward	Ship goes forward	Pass
Player move	Move backwards	Ship goes backwards	Pass
Player move	Rotate left	Ship rotates left	Pass
Player move	Rotate right	Ship rotates right	Pass
Player shoots	Left click	Weapon fires	Pass
Player selects	Select asteroid	Asteroid selected	Pass
Player selects	Select NPC ships	NPC ships selected	Pass
Collision	Move into asteroid	Asteroid bounces away, damage taken	Pass
Collision	Move into ships	Bounce off	Pass
missions	Complete mission 1	Mission 1 complete	Pass
missions	Complete mission 2	Mission 2 complete	Pass
missions	Complete mission 3	Mission 3 complete	Pass
missions	Complete mission 4	Mission 4 complete	Pass
missions	Complete mission 5	Mission 5 complete	Pass
missions	Complete mission 6	Mission 6 complete	Pass
missions	Complete mission 7	Mission 7 complete	Pass
missions	Complete mission 8	Mission 8 complete	Pass
missions	Complete mission 9	Mission 9 complete	Pass
missions	Complete mission 10	Crash screen	Fail
Weapon upgrade	Isotopes upgrade weapons	Weapons upgrade	Pass
Mod upgrades	Mods upgrade ship	Mods upgrade	Pass

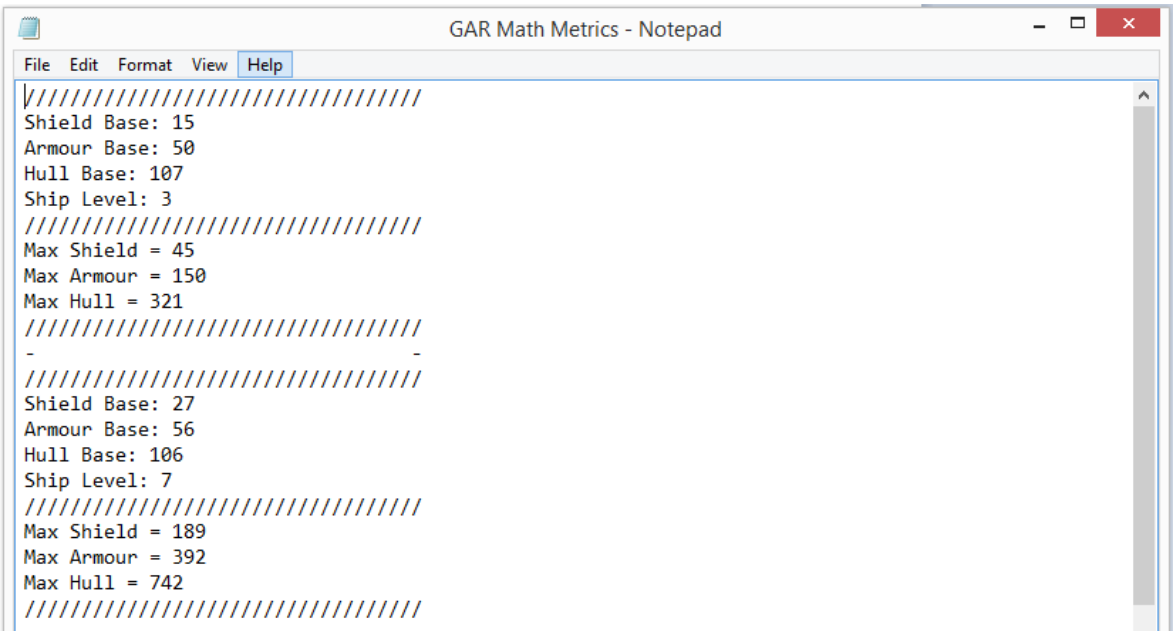
10.3 Appendix 3: White-Box Testing Data



```
File Edit Format View Help
////////////////////////////////////
body number: 0
engine number: 1
head number: 0
wing number: 8
weapon number: 1
Tail number: 7
////////////////////////////////////
Engine Speed: 15
Rotation Speed: 0.0025
Weapon Damage: 55
Ship Shield: 150
Ship Armour: 300
Ship Hull: 600
////////////////////////////////////
body number: 0
engine number: 2
head number: 3
wing number: 7
weapon number: 4
Tail number: 1
////////////////////////////////////
Engine Speed: 15
Rotation Speed: 0.0025
Weapon Damage: 55
Ship Shield: 150
Ship Armour: 300
Ship Hull: 600
```



```
File Edit Format View Help
////////////////////////////////////
Body Number: 3
Engine Number: 6
Head Number: 3
Wing Number: 6
Weapon Number: 3
Tail Number: 0
////////////////////////////////////
Engine Speed: 18
Rotation Speed: 0.001
Weapon Damage: 16
Ship Shield: 70
Ship Armour: 200
Ship Hull: 421
-
////////////////////////////////////
Body Number: 4
Engine Number: 5
Head Number: 0
Wing Number: 2
Weapon Number: 4
Tail Number: 2
////////////////////////////////////
Engine Speed: 14
Rotation Speed: 0.001625336
Weapon Damage: 34
Ship Shield: 214
Ship Armour: 442
Ship Hull: 842
```



10.4 Appendix 4: Questionnaire

Welcome to the GAR questionnaire, please answer the questions that you are told to answer. Use the space underneath your chosen answer and place an “x” symbol in that slot.

The test we be in 3 stages where you will play as many versions of the game Galactic Arms Race. The first stage will be to play the original GAR for 5 minutes. Stage 2 will be to play GAR version 2, 3 times for 5 minutes. Stage 3 will be to play GAR version 3, 3 times for 5 minutes. Each time you finish playing the game please answer the questions the organiser tells you to.

The test should be about an hour so sit back and enjoy the game!

1. You have played GAR version one. Using the following scale, how much did you enjoy this game version?

1	2	3	4	5	6	7
Extremely bad	Quite bad	Slightly bad	Neither	Slightly good	Quite good	Extremely good

2. Do you agree that the ship you played with effected the way you approached GAR as a game

1	2	3	4	5	6	7
Extremely disagree	Quite disagree	Slightly disagree	Neither	Slightly agree	Quite agree	Extremely agree

--	--	--	--	--	--	--

3. You have played GAR version two for the first time. Using the following scale, how much did you like the ship in this version?

1	2	3	4	5	6	7
Extremely bad	Quite bad	Slightly bad	Neither	Slightly good	Quite good	Extremely good

4. Do you agree that the ship you played with effected the way you approached GAR as a game

1	2	3	4	5	6	7
Extremely disagree	Quite disagree	Slightly disagree	Neither	Slightly agree	Quite agree	Extremely agree

5. You have played GAR version two for the Second time. Using the following scale how much did you like the ship in this version?

1	2	3	4	5	6	7
Extremely bad	Quite bad	Slightly bad	Neither	Slightly good	Quite good	Extremely good

6. Do you agree that the ship you played with effected the way you approached GAR as a game

1	2	3	4	5	6	7
Extremely disagree	Quite disagree	Slightly disagree	Neither	Slightly agree	Quite agree	Extremely agree

7. Do you agree that one (or more) ships you played with in this play through were better than others?

1	2	3	4	5	6	7
---	---	---	---	---	---	---

Extremely disagree	Quite disagree	Slightly disagree	Neither	Slightly agree	Quite agree	Extremely agree

8. If you agreed to the previous question, please place the ships in an order using the table given. The top one is the most favoured while the bottom one is the least favoured. The first ship you had is ship 1 and the second is ship 2

1	
2	

9. You have played GAR version three for the first time. Using the following scale, how much did you like the ship in this version?

1	2	3	4	5	6	7
Extremely bad	Quite bad	Slightly bad	Neither	Slightly good	Quite good	Extremely good

10. Do you agree that the ship you played with effected the way you approached GAR as a game

1	2	3	4	5	6	7
Extremely disagree	Quite disagree	Slightly disagree	Neither	Slightly agree	Quite agree	Extremely agree

11. You have played GAR version three for the second time. Using the following scale, how much did you like the ship in this version?

1	2	3	4	5	6	7
Extremely bad	Quite bad	Slightly bad	Neither	Slightly good	Quite good	Extremely good

12. Do you agree that the ship you played with effected the way you approached GAR as a game

1	2	3	4	5	6	7
Extremely disagree	Quite disagree	Slightly disagree	Neither	Slightly agree	Quite agree	Extremely agree

13. Do you agree that one (or more) ships you played with in this play through were better than others?

1	2	3	4	5	6	7
Extremely disagree	Quite disagree	Slightly disagree	Neither	Slightly agree	Quite agree	Extremely agree

14. If you agreed to the previous question, please place the ships in an order using the table given. The top one is the most favoured while the bottom one is the least favoured. The first ship you had is ship 1 and the last ship is 2.

1	
2	

15. Would you agree that there was a noticeable difference between all three versions?

1	2	3	4	5	6	7
Extremely disagree	Quite disagree	Slightly disagree	Neither	Slightly agree	Quite agree	Extremely agree

16. If you agreed to the previous question, please place the versions of GAR in an order using the table given. The top one is the most favoured while the bottom one is the least favoured.

1	
2	
3	

Thank you for participating in this questionnaire. If you would like to express your thoughts on what you thought of the whole experience, you are more than welcome to do so. I only ask that the wording of what you put isn't too offensive to be placed in publishable document!

•

10.5 Appendix 5: Spreadsheet Data

Version 2 body parts(game one)						
Players	Body number	Head number	Wing number	Engine number	Weapon number	Tail number
P1	6	3	5	10	10	5
P2	6	10	5	9	9	9
P3	10	2	5	2	1	1
P4	5	8	2	4	10	1
Version 2 ship stats(game one/two)						
Players	Engine speed	Rotation speed	Weapon damage	Ship shield	Ship armour	Ship hull
P1	15	0.0025	55	150	300	600
P2	15	0.0025	55	150	300	600
P3	15	0.0025	55	150	300	600
P4	15	0.0025	55	150	300	600
Version 2 body parts(game two)						
Players	Body number	Head number	Wing number	Engine number	Weapon number	Tail number
P1	5	6	3	6	1	6
P2	5	7	1	6	5	1
P3	5	5	2	3	5	5
P4	7	5	10	7	1	5

CMP3060M Project Item 1

Version 3 body parts(game one)							
Players	Body number	Head number	Wing number	Engine number	Weapon number	Tail number	
P1	7	3	4	2	7	3	
P2	8	3	9	10	1	9	
P3	3	2	4	6	10	9	
P4	3	5	8	8	2	5	
Version 3 ship stats(game one)							
Players	Engine speed	Rotation speed	Weapon dama	Ship shield	Ship armour	Ship hull	
P1	11	0.0039	68	151	379	772	
P2	12	0.0033	60	158	358	849	
P3	10	0.0035	63	235	435	772	
P4	10	0.0035	63	115	227	409	
Version 3 body parts(game two)							
Players	Body number	Head number	Wing number	Engine number	Weapon number	Tail number	
P1	1	1	3	1	6	8	
P2	10	6	10	2	8	2	
P3	9	5	6	3	3	1	
P4	1	9	3	3	4	6	
Version 3 ship stats(game two)							
Players	Engine speed	Rotation speed	Weapon dama	Ship shield	Ship armour	Ship hull	
P1	19	0.0026	49	214	372	758	
P2	12	0.0046	79	121	242	532	
P3	17	0.0013	30	91	197	400	
P4	13	0.0053	89	155	300	565	
Version 3 base values(game one)							
players	Shield base	Armour base	Hull base	Ship level	Max shield	Max armour	Max hull
P1	18	47	96	7	126	329	672
P2	19	44	107	7	133	308	749
P3	30	55	96	7	210	385	672
P4	30	59	103	3	90	177	309
Version 3 base values(game two)							
players	Shield base	Armour base	Hull base	Ship level	Max shield	Max armour	Max hull
P1	27	46	94	7	189	322	658
P2	24	48	108	4	96	192	432
P3	22	49	100	3	66	147	300
P4	26	50	93	5	130	250	465

CMP3060M Project Item 1

[illegible]