[redacted]

2D TILE-BASED GAME

Author: [redacted]

Supervisor: Alan Crispin

Computer Games Technology BSc Hons

Declaration

No part of this project has been submitted in support of an application for any other degree or qualification at this or any other institute of learning. Apart from those parts of the project containing citations to the work of others, this project is my own unaided work.

[redacted]

Signed _____

Abstract

This document was created with the purpose of aiding in the planning, justification and the recording of the development process involved in creating a 2D Tile-Based game. In addition to this the document shall also display the means and results of evaluation.

Table of Contents

Figure List

Table List

1. Introduction

The aim of this project is to research, design and develop a series of algorithms and structures which compose a 2D Tile-Based game.

This report will focus on the various processes involved in the implementation and application of both old and new skills which have been acquired over the past three years. The majority of this report will be dedicated to the research and implementation of the numerous resources found through research.

2. Literature Review

2.1 Abstract

This literature review will focus on the algorithm techniques required for developing the project, related research material and required software for development.

2.2 Terminology

1. **Mecha**: Also known as **Mech** or **Mechs**. Fictional, robotic vehicles often humanoid in appearance. Popular concept originating from the United States made popular by Japanese culture in cartoons and printed media.

2.3 Required Algorithms

Being a game based around the movement of objects on a 2-Dimensional grid, pathfinding is essential for navigating objects around obstacles using the shortest and most efficient path. As

such, this requires an algorithm which can present and perform each possible legal path in an efficient manner, using the least amount of recursion and/or being the least complex algorithm.

In order to provide a strategic challenge for the user outside of using complex game mechanics to overcomplicate the gameplay, the project will require an Artificial Intelligence to control any Non-Player Characters (NPC). A basic AI system will first need to be implemented and then if possible, depending on the chosen method, develop a more advanced AI system to add more layers of depth to the strategy element.

2.3.1 Comparison of Required Techniques

2.3.1.1 Pathfinding

The pathfinding technique that is commonly applied to situations such as this is the 'A*' Algorithm (Appendix B, Fig. B.1) which is used to find the cheapest path by means of checking each possible efficient path to the destination, not the cheapest in terms of memory usage but arguably the most effective. Another algorithm, going by the name of "Best-First Search", is a very simple pathfinding algorithm in which the cheapest path on hand is chosen first. By applying logic based on algorithm complexity, this may take up less memory rather than checking each potential. While potentially cheaper, algorithms of this type can derive results of lesser quality, "Although this may seem like a reasonable solution, the best-first algorithm can often result in sub-optimal paths." (Madhav, 2013:185)

The quality of the pathfinding algorithm will somewhat depend on the number of obstacles on the grid, if the organisation of the obstacles becomes complex then an accurate algorithm will be required. Due to the potential inaccuracy of the Best-First algorithm, it is safe to say that the A* algorithm will be the more logical choice when approaching the pathfinding of this project.

2.3.1.2 Artificial Intelligence

Gameplay design and complexity can only provide so much strategic depth for a game, for this game to provide more for the users it will require an AI (Artificial Intelligence) to control enemy units. One of the most common methods of creating AI in computer games is “State-Based Behaviours” (Appendix B, Fig. B.2), built around the concept of finite state machines controlled by conditions triggered by environment or the user. This method has the potential to be improved when implemented in a basic form simply by adding additional states or conditions to respond to the player’s actions.

2.4 Relevant Research Material

The basis of how this game will function as a strategy game has already been decided, however the method of implementation and structure of the gameplay require examples of how they are meant to be designed. As a result, several works, not necessarily related in overall gameplay structure (2-D tile based), have been chosen to collect inspiration and help further develop the gameplay elements required for this project. E.g. Unit statistics, movement rules, damage calculation.

2.4.1 Ring of Red

A 2D Tile-Based strategy game developed for the PlayStation 2, Ring of Red (Konami, 2001 EU) will be used as a focal point not only for the gameplay but for the arty style, animations and statistical representation of the units. The gameplay comes in two forms, tactical map and battle phase, the latter will be disregarded until needed to explain statistical representation. Ring of Red is to be considered a very closely related piece of work due to its similarity to the project being undertaken.

In terms of gameplay, interacting with units will bring up their movement/attack range represented by overlays on the respective tiles. After selecting a destination, an interaction menu appears giving the player the option to allow their selected unit to attack within the specified range, recover health, open the statistics panel of the unit or skip their turn. Each unit is permitted a single move action based on their movement range and an action which may be used as previously specified. While this seems simple, this can promote strategic thinking through forward planning of available actions and careful placement of units.

Vehicles in Ring of Red may have additional indicators representing the current status of a part of that vehicle, e.g. legs, weapons, wheels. This is a feature that will hopefully be implemented in a similar manner, using percentage bars rather than two-state indicators and allowing more parts to be represented. Depending on the part destroyed different effects can occur, such as, destroying the legs will slow/immobilize the unit or destroying a weapon will prevent the weapon from being used until repaired.

Art style for Ring of Red is mostly based in the World War II era where units are represented by either a 2D sprite on the tactical map or as full 3D models in the battle stages. The hulls of the Mechs and vehicles are weathered and often show exposed wiring or hydraulics. The Mechs and vehicles are also decorated in their respective faction's camouflage so as to easily distinguish the units. On the tactical map, sprites are created with emphasis on the unit's main features, e.g. long range units have long cannon, and soldier units are a group of one or more soldiers. From this it is important to know how to represent units in a manner that players can easily understand through assumption rather than a reference.

For the animations in Ring of Red, the focus will mostly be on the Mechs so as to gain an idea of how these are seen and should feel to the user. The animations are mostly seen in the battle

stages where the units are represented in 3D and consist of movement (forwards/backwards), firing/attacking, firing position and damage. Each of these animations are slow, bulky and are meant to emphasise the size and mass of walking tanks

2.4.2 Warhammer 40,000

Created by Games Workshop (Games Workshop Limited, 2016) in 1999, Warhammer 40,000 is a table-top turn-based strategy game based around the gameplay and usage of unit statistics; the focal points of this analysis. Units in this game are assigned statistical values (Fig. B.3) which represent combat ability in different areas which depend on the current stage of combat. Vehicles have a similar representation of statistics, but most of the statistics that would seem logical to a soldier are replaced with armour values for the front, sides and back of the vehicle. This means that players must choose wisely in order to be an effective threat towards vehicles and position themselves carefully around the vehicle.

Warhammer 40,000 utilises dice in much of the gameplay from deciding who should play first, the number of shots fired hitting their target or checking if soldiers will flee from battle. The number of dice can vary depending on the statistics of both the weapon being used and the user of said weapon. Success of an action depends on whether or not the value of the dice exceeds or matches the requirement calculated through comparisons, for example, a player rolls six dice to determine if their squad hits the target. Each dice is required to land on a 4 or higher in order to succeed. If 3 dice land on a value greater than 4, the rest do not, this means that the player's units have made 3 successful shots and may move on to the damage phase.

This level of complexity can either greatly enhance the experience of a strategy game or diminish it with the sheer amount of statistics and data that the player must track. With this, one must take into account how they are going to present this data in order to avoid overwhelming

the user with responsibilities and excessive data. Regarding the use of dice in this game, controlled chance is a good element to introduce at it promotes a risk/reward scenario in most cases. This can however, detract from the strategic value of the player's actions; making placements and careful use of actions seem worthless after receiving a failed result.

2.5 Required Software

2.5.1 Required Software Types

This project requires a number of assets of differing formats ranging from 3d models to 2d images and development software on which to code, test and debug. Each required software type will be listed with potential candidates for usage with both advantages and disadvantages including the price (if any) of the software and if it can be justifiable to the means.

2.5.2 Development Tools

Development of this game first requires both a tool and a language in order to write the code and apply it to whichever platform is deemed fit for release. As such some examples of software are designed for creating games to compare them all on a technical standpoint which includes the programming language used and compatibility with the three major operating systems.

2.5.2.1 Unity (Version 5.3.2)

The Unity game engine (Unity Technologies, 2005) allows for the use of either Java or C#, both a derivative of C language with a large library of pre-defined classes and utilities such as Lists (dynamic array) or Stacks which are most useful when creating pathfinding algorithms. Being a game engine allows for easy creation and organization of assets and the generation of levels using an in-built terrain editor. In addition to these tools it also allows for exporting to different forms on multiple platforms such as Playstation and Xbox. All that is required from this is the ability to export to Windows, Linux and Mac which the program does support.

The personal variant of Unity is free to an extent, including only the basic tools for use, once the product has made a significant sum of money then a portion of further income is taken by Unity as royalties. This project cannot make money and will not be developed for the purpose of

making money, as such the royalty payments pose no issue which would make Unity a viable choice.

2.5.2.2 Unreal Engine (Version 4.12.5)

Unreal Engine (Epic Games, 1998) is another game engine bearing some similarities to Unity with the exception of using C++ for coding and algorithms and the easier achievement of a higher graphical fidelity. Using C++ does allow for higher limits on memory for the programming stages and is more versatile when creating classes, however this use of memory can cause the created programs to become unstable at times due to potential excessive usage of memory. In addition to this possible scenario, based on personal experiences using Unreal, and despite checking the program many times over, unlogged errors became a common occurrence and resulted in the program skipping essential pieces of code.

Much like Unity, Unreal is free to download and use to an extent, one would only pay for additional features or assets on Epic Game's marketplace or royalties depending on how much the product is earning. No money is to be spent or gained on this project which makes this another viable option. However, the instability brought on by a lack of built-in "clean-up algorithms" and general lack of personal knowledge of the Unreal based C++ code is something to take into account when deciding on the development method.

2.5.2.3 Coding from Scratch

Without a doubt the most time consuming method of development, writing code for this project from scratch, would require a large amount of time to plan, write and test each written method and class. In addition to this, advanced techniques for creating the required complex algorithms would require precious time to learn and practice in order to create the necessary code for 3D representation using the chosen language. Despite these setbacks should this method be chosen,

building the code from scratch would allow the option of choosing any language deemed fit, preferably a language that can be compatible amongst the three major operating systems. Since a specialised program (e.g. Unity and Unreal) is not being used to build the product, the code can be tailored to only include what is deemed necessary rather than having unused optimisation features littering the coding.

The cost of this method depends solely on what program is chosen to write the code. Two examples of these are Notepad++ and Microsoft Visual Studio 2016. Both Notepad++ and Visual Studio are Integrated Development Environments (IDEs) which allow structuring and referencing amongst the code and will also highlight any components of a program based on the programming language being used. Visual studio however, provides better organization with a solution file for listing and grouping core files appropriately, referencing of other files with a suggestive text feature and a built-in compiler and debugger for testing programs. Since both of these examples are free it would be safe to assume that Visual Studio would be the better choice should it be decided to develop this game from scratch. However, with the limited time and knowledge on 3D coding available it would be unwise to utilise this development method in addition to the accessibility issues created by a lack of “cross platform” access which other IDEs possess.

NOTE: Visual Studio is packaged with Unity as a means of editing scripts and classes.

2.5.3 2D Assets

In order to make the user interface of this project more appealing, it would be wise to use images in place of interface elements so as to aid the immersion of the player and to create an environment that would better suit the overall theme of the project. To accomplish this, a program will be chosen that will provide the best tools for creating both textures and images

while being able to export the finished images to the required format dependant on the development tool. A list of programs has been compiled which have a strong relation as to what is needed.

2.5.3.1 Adobe Photoshop CS6

Part of the Adobe Creative Suite, Photoshop CS6 (Adobe Systems, 2003) is an extremely versatile program with a wide plethora of advanced tools and is commonly used professionally in most visual media-based industries, definitely a high quality piece of software for creating many different kinds of images and exporting them to almost any format, providing the correct plugin is installed. The most glaring downside to this high quality tool is the price, £17.15 per month for a licence. This option is not overly expensive if used for a single month but if possible, it would be wiser to find an alternative option of equal or slightly lesser quality and ideally free.

2.5.3.2 GIMP 2

A program very similar to Adobe Photoshop with the exception of fewer features but has the ability to export images to a wide variety of formats; GIMP 2 (Kimball & Mattis, 1995) is a free alternative to Photoshop which immediately makes it the ideal choice. Aside from the disadvantage of having fewer features than Photoshop, GIMP 2 has a large community dedicated to support and free plugins due to the program being ‘open-source’ which allows the user to expand the format types and the tools available, a good choice if one only wishes to have select features. Overall this seems to be a vast improvement over paying to use Adobe Photoshop and would be considered the most viable option for creating the 2D assets.

2.5.4 3D Assets

Despite the project being classified as a 2D game, 3D assets may be used to represent units and the environment. Creation of these assets would require an appropriate tool which is able to

export the 3D model to the appropriate format; once again dependent on the development tool, and it would be preferred if the selected program has a User Interface that is easy to understand and learn due to lack of personal skills in advanced 3D modelling. As with the development tool and the 2D asset tool, the prices of the found programs will be taken into consideration.

2.5.4.1 Autodesk Maya

This is a professional program commonly used in a variety of different industries requiring 3D assets known as Autodesk Maya (Alias Systems Corporation, 1998). Due to the professional nature of the program, Maya has a wide variety of high-end tools used for creating advanced 3D models including, but not limited to, hair/fur physics, cloth physics, and fluid physics. This scenario is very similar to Photoshop, while the program is indeed of a high standard, the price of £170 monthly is simply beyond reason. If possible, find a cheaper alternative with a vivid support community and additional plugins for exportation if needed.

2.5.4.2 Blender

Once again similar to the 2D asset tool, Blender (CITE) is an open-source 3D modelling tool with a vivid support community. The program is free and offers a wide variety of plugins for both exportation and additional tools similar to GIMP, however the interface of Blender is something unfamiliar and would require time to learn and apply to the project. Despite this, the program is free to download and use which immediately makes this a more optimal choice over Maya.

2.5.5 Chosen Candidates

For writing the code during the development stage, it has been decided to base the decision on overall experience with the chosen software and the programming language and as such, a decision has been made to choose Unity using C#. Choosing a game engine was always going to

be the more probable choice as little success has been had when it came to creating a functional game from scratch in personal history. Generating 3D objects using C# or C++ is a concept that has also had very little personal practice applied. As for the language being used in Unity, C# was considered the optimal choice in comparison to Java for its versatility and optimisation as stated by Marc Eaddy (2001:9),

"For Java programmers, C# picks up where Java leaves off by providing a high-performance, component-oriented language that integrates tightly with Windows."

Choosing the program for creating the 2D assets was an easy choice; paying for a professional program over a free alternative with community support was considered an unwise decision. As such, GIMP 2 has been chosen to create the required assets and will require the download and installation of the plugins needed to export each asset to the required format.

Creation of 3D assets may require a compromise in which both tools are used depending on the complexity of the required object and time available. Maya is an incredible piece of software with pre-existing personal knowledge but paying for a subscription is not applicable in this situation, however a version of this does exist on the University systems. Blender is not the optimal choice for this situation, due to the lack of personal experience with the program, it is however both free and flexible which are both very necessary attributes for this project. The compromise being proposed for this issue is to use both programs for modelling when available; Blender is to be used for basic modelling due to its price and personal skill and Maya is to be used for more advanced models. Since the use of 3D models is not a definitive possibility, the use of these tools should not take priority over the foremost required assets such as the development of code and general 2D assets.

3. Design

At the end of this Design section there will be;

- a plan of how the interface will be laid out,
- how the gameplay will be structured,
- type and amount of classes required for the game to function.

3.1 Interface design

The design of this interface shall be centred round the concept of being able to provide the user with an adequate view of a grid system and means of displaying data relevant to the gameplay. For this to be achieved full utilization of the built-in Unity assets created for designing an interface system will be required.

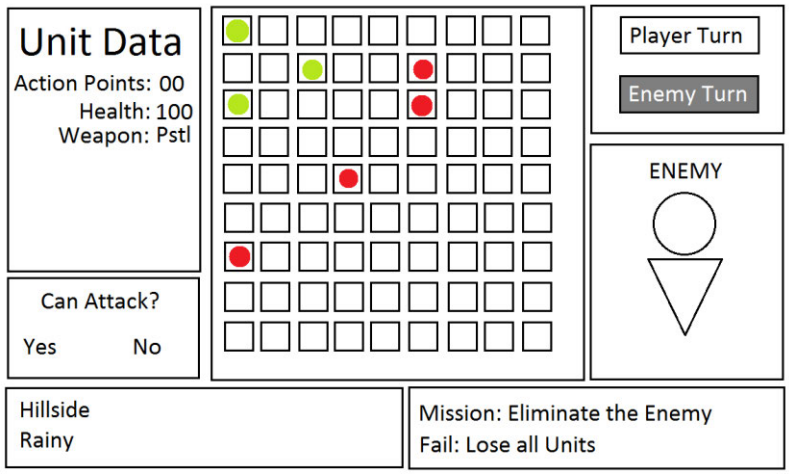


Figure 3.1. Diagram of proposed Graphical User Interface design.

3.2 Gameplay structure

Being a strategy game in a 2D tile-based system limits the number of options for progression and how the game would "flow" under normal circumstances. Therefore, it is logical that one would utilize a turn based system so as to retain a level of control when dealing with assets which have been imposed with a limited range of movement.

3.3 Classes and scripts

In order to achieve the desired functionality of the areas stated above, the product will need a series of classes and scripts coded in C#, all of which will have to work off or with each other so as to represent the relevant data structures as a compilation of variables and methods. Attainment of these required assets is possible through the creation, organization and delegation capabilities of the Unity engine.

3.4 Product explanation

This product will be a 2D strategy game based round the idea of having units combat each other in order to achieve a pre-determined goal. The player must use their knowledge and skills for the purpose of eliminating all the opposing units which are controlled by an Artificial Intelligence. Once the player has achieved this task, they are rewarded with a congratulatory message. Should the player fail this given mission, then they are presented with a failure message. As the units battle, the attacking units use their weapons to deplete the health of the defending units. Should they be available the units are allowed to use "Med" kits in order to restore lost "health". The units are comprised of weak, but strong in numbers, soldiers or slow, but powerful, tanks. Players make a selection of the units by means of "buying" their armies with a pre-determined amount of points. Players must wait for their delegated turn in order to take action against the enemy.

4. Implementation

4.1 Introduction

This section of the report is dedicated to the explanation of how the product of this project is implemented.

In segments regarding the C# classes, due to the vast amount of methods and variables present in each of the entries, these class sections will only go into detail where absolutely necessary and instead give an outline as to what methods or variables have been created to accomplish this.

4.2 Terminology

- **GameObject**: An asset used within the Unity game engine which often refers to a non-static mesh and/or transform. These are often used in this project to represent the tiles on a grid or as a means of organising the games hierarchy through parenting.

4.3 Gameplay

This section will focus on how the flow of the game will progress during a standard session of the game. The game consists of two players, the first being a User-controlled player who will utilise the interface as a means of controlling their respective units. The second player will be controlled by an Artificial Intelligence (AI), the likes of whom will control the units based on built-in parameters, averages and percentages of its controlled units.

As the gameplay of this game is turn-based, only one player may manipulate their units at a time while the other must wait for their turn to begin. Turns may be changed over manually by means

of the interface through a button, the use of this feature rather than an automatic turnover system allows the player to retain the feeling of control over their gameplay session where permitted.

The endgame point in these sessions is determined simply by who has lost all of their units, the likes of whom will be designated as the loser while the opposing player will be designated as the winner. It may be possible to implement an end-game state based on a turn limit in which the game ends when a pre-determined number of turns has passed. In this case, the victor would be decided by determining which faction/player has the highest number of units remaining. Implementing this end-game state would result in the risk of draws becoming a possible outcome.

4.4 Level Design

Each level within this game is to be displayed as a tile grid of varying size, scattered around this grid in a haphazard manner are tiles which have been blacked out so as to represent areas in which units may not move through or onto. It should be noted that any levels of significant import may not utilize the random nature of the obstacles and instead be pre-planned in order to highlight the strengths and weaknesses of any distinctive features of units unique to these levels. The outcome of this choice of presentation is a lack of visual variety in each of these levels but also provides a level of unpredictability for the user.

4.5 C# Classes

This segment of the design statement will focus on the clarification of the classes within the product, which were created, with the use of the C# programming language. While the intricacies of the classes are to be stated, the actual code used in the project will remain separate only to be present and referenced in Appendix B.

Please note that this section does not cover all of the classes found within the product, rather it shall only describe those deemed necessary for the product to function on a technical level. Those that control assets such as the Graphical User Interface will remain separate as they are similar in function and focus on the referencing of data.

4.5.1 Tile Map

The classes within this segment are associated with the generation, manipulation and storage of the grid map used within the levels. The grid itself is created as a 2D array of 3D objects, which represent the tile. Despite this method of possibly being contradictory to the idea of the product being a 2D tile game the grid still operates within a 2D plane and is made to appear as such utilising an orthographic camera.

4.5.1.1 Tile Class

This class acts as a script which is to be attached to a 3D tile shaped object within the Unity engine, this object will then act as what is known as a ‘Prefabricated Object’ or ‘Prefab’ for short. This Prefab may be instantiated at any time as an object within the Unity scene which makes this an ideal method of generating many tiles at once in order to form a grid-like pattern. This method of creating a tile grid also allows the changes of any methods or variables and distributing them efficiently since they are all copies (or Instances) of the same tile.

Variables

As per usual, variables within this class are assumed to be public read-only unless stated otherwise. The following variables are to be considered of a higher import than others found within the class.

- ***RefIdX*/*RefIdY***: Both of these variables are integers which store the X and Y co-ordinate values of the tile respectively. This is the Tile's main method of retaining its own position in the grid without having to refer elsewhere.
- ***CurrentUnit***: This variable is responsible for storing a reference to the *Unit* class which is currently occupying the tile.
- ***State***: This is a variable of type *TileState* which stores the current state of the tile in the form of an integer-based enumerator value, this allows a switch statement to be used to assign.

Methods

The methods of this class are focused around the referencing and manipulation of the variables within the class. Typically, these methods centre around the state of the tile, changing the material of the tile object to be relevant to its current state in order to provide feedback to the user where needed.

4.5.1.2 Tile Anchor

Organization and management of the Tiles on the grid is done through the Tile Anchor class, this is a script which is attached to an empty GameObject so as to allow repositioning of the tile generation if required. This class acts as a central hub to some classes and as such contains a large amount of

4.5.2 Unit representation

4.5.2.1 Unit Class

Representation of units within the world in Unity C# is achieved in this project by means of attaching a script to a moveable ‘GameObject’. Using a method such as this can allow for multiple classes of differing structure and representation to be used given that the appropriate methods are present to compensate for data interpretation by the main ‘Unit’ class. This method of depiction can present problems of its own however, using two separate classes which require attachments to the respective object will require some form of ‘flow control’. This will allow the appropriate methods to be called given the type of addition class present. For the time being, this will be a simple Boolean variable as there are only two unit types available, additional types will require a more complex means of representation and manipulation.

Variables

Since the data used for gameplay is stored and referenced within the attached classes, variables within this class are used for either internal purposes or the referencing of the appropriate attached class.

- ***IsVehicle***: A private Boolean value used for flow control within some methods. When set to true, said methods will correspond to using algorithms relating to the class ‘Vehicle’ and vice versa.
- ***CurrentTile*****:** A public, read-only variable of type ‘Tile’. This represents the tile the Unit is currently occupying. This value can only change during initialization, movement or deconstruction.

Methods

The majority of the methods within the *Unit* class are often used for ‘flow control’ as previously mentioned, referencing the attached *Squad* or *Vehicle* class based upon the value of a Boolean variable, *IsVehicle*. The rest of the methods in the *Unit* class are based around the universal functions, which by logic would not require unique variants to be present within the attached classes such as methods, which control the movement of the unit and checking if the unit can be removed from its owning faction and deleted.

4.5.2.1 Squad Class

In order to utilize the soldier class numerous times in a cohesive manner, a collection-based class known as ‘Squad’ needed to be implemented. This class is responsible for handling action points derived from the total of the action points assigned to each ‘Soldier’ class contained within the list. This class is also responsible for removing soldiers, which have died, and amending the list and other related variables accordingly.

Variables

As previously mentioned, this class is collection based so as to properly represent the *Soldier* classes within the squad, it is then expected from this class to contain some form of list or array. In addition to the collection variable, this class will employ more basic variables as means of retaining other valuable data to be used by the class itself or for gameplay purposes. For example, Movement action points are an integer stored and denoted by the smallest value of ‘action points’ found in the *Soldier* classes.

- ***Soldiers***: A public, read-only list or array, which retains the data of each of the soldier classes inside a squad, the number of entries within this list, may not be greater than nine.

When the health of any soldier class in the list becomes less-than or equal-to 0, that soldier is removed from the list and the appropriate changes are made to other variables.

- ***SoldierCount***: A private integer used for keeping record of how many soldiers are inside a squad, decreases when a soldier's 'Health' is reduced to zero and that soldier is removed. When the value of this integer becomes zero then the owning *Unit* class triggers a removal method of the owning *Faction* class, removing this the owning Unit from both the game and the Faction.
- ***MoveActionPoints***: A public integer, which represents the maximum number of tiles the owning unit, may move, this is determined by obtaining the lowest action point value contained in the *Soldier* classes from the list, *Soldiers*. This is updated in accordance with usage during gameplay and reset at the end of each turn to the value of *MaxMoveActionPoints*.
- ***MaxMoveActionPoints***: A private integer which represents the value that *MoveActionPoints* is reset to after the owning players turn has ended. This value is only changed when a soldier is removed from *Soldiers*, this change is necessary to keep up with any potential changes with the minimum value of action points present in the *Soldiers* list.

Methods

Management, manipulation and evaluation of the *Soldiers* list are the focal points of the methods within the *Squad* class. The majority of these methods will involve a full, linear search of each entry of *Soldiers* in order to fulfil the task of removing or editing specific values of the soldier classes present.

4.5.2.3 Soldier

The ‘Soldier’ class is responsible for representing the data and statistics of each soldier that is found within a list in the ‘Squad’ class. The data for this class is stored within a database document which is accessed through a plugin for Unity known as ‘SQLite’ (Hipp, 2017). The purpose of this is to allow ease of access when changes need to be made to the data for ‘balancing’ purposes or new additions are required. The data for the Soldier classes are referenced, initialized and stored in a class known as ‘UnitGen’.

Variables

All variables within the *Soldier* class are assumed to be public, read-only values unless stated otherwise. The purpose of this is to reduce the risk of any unauthorized changes to the variable occurring through external methods. These are some of the variables which are of a higher importance.

- ***Health*:** An integer which stores the current health of the soldier, decreased by an internal public method. Starts as the same value as *MaxHealth* by default.
- ***MaxHealth*:** A private integer which stores the maximum health of the soldier, would typically be used if *Health* is ever increased.
- ***ActionPoints*:** An integer which stores the current action points of the soldier, decreased by movement of the owning Unit or when the weapon is accessed and used to attack
- ***MaxActionPoints*:** A private integer which stores the maximum value of *ActionPoints*,
- ***Weapon*:** This variable stores a *Weapon* class which corresponds to the data set when the data for the *Soldier* class is referenced and created in a *UnitGen* class.

4.5.2.4 Vehicle

Vehicle is similar to the *Squad* class in that it is attached to an object through the *Unit* class and is based around the idea of using a collection system as a means of self-representation.

Variables

In this class, the health of the unit will be represented by means of a list or array of integers which store the health of each individual part of the unit and a collection of equal length containing strings which represent the name of the respective part. As an alternative to having each part represent the full health of the unit, the representative part or 'main part' will serve this purpose. Should the value of this part become less-than or equal-to zero, the unit shall be removed from the game as a casualty.

As usual, all variables are assumed to be public-read only unless specified.

The vehicle class also contains two read-only variables of the *Weapon* class,

4.5.2.5 Weapon

4.5.3 Factions

The 'Faction' class will act as a means of collecting and managing the various units owned by either player. In terms of management, this class is responsible for triggering the reset methods of each unit after every turn and checking a unit after it comes under attack; if the unit is to be deleted then it will be removed accordingly. In addition to management of its own units, the *Faction* class will also trigger an endgame event when all of its Units are removed as a result of casualties, depending on whether this faction is controlled by the Player or the AI, this will present a win or lose state respectively.

Variables

The variables within the *Faction* class are assumed to be public, read-only values unless stated otherwise. The purpose of this is to reduce the risk of any unauthorized changes to the variable occurring through external methods. The only variable worthy of note is the *Units* variable which is basically a list of the *Unit* class.

Methods

Similar to the *Squad* class, methods involved here are centred around the manipulation and referencing of many entries within a list. Rather than deleting itself after the list becomes empty however, this class triggers an event within a global, static class which ends the game.

4.5.4 UnitGen

This class simply references the database which retains the data for each soldier or vehicle while utilising the plugin, SQLite. After the appropriate table for each class type has been referenced, instances of said data are created and then stored within public lists so as to allow other classes to reference this data for the purposes of initialization.

4.5.5 Artificial Intelligence

Control over the enemy units including their movement and attacking methods are all handled by the Artificial Intelligence (AI) class. Essentially this class is designed to mimic the process that a player may take when manipulating their own movements. This is achieved by first picking a target unit which is determined by choosing an enemy unit which is closest to the unit currently being manipulated. Second, a path is found to the target unit which is then trimmed down to accommodate the controlled unit's remaining action points. Finally, the path is compared with the average range of the controlled unit's weapons; if the target is in range then it will attack the

target rather than moving closer to it. This process is repeated for each unit within the faction that the AI is controlling, after that the AI triggers a global event to switch control back over to the player.

4.5.6 Turn Manager

Gameplay progression must be managed by a form of class which is capable of enabling or disabling certain restrains and controls for use by either the player or the AI. For this, a class known simply as ‘Turn Manager’ was created. In addition to managing the inputs for the two players, the Turn Manager is also capable of pausing, restarting and closing the game in the form of a ‘Pause Menu’.

5. Evaluation

5.1 Introduction

The purpose of this document is to present, describe and justify the data and the means of collection in order to further improve or amend areas of the project. In addition to identifying potential improvements, this document shall also mark the corrections made to errors within the 'Black Box Testing' section.

5.2 User Evaluation

Whilst the majority of the product is evaluated based upon its functionality and optimization, the following areas of the project will require testing by users.

- Gameplay: This area focuses on the balance and implementation of certain game mechanics. Having Users test this will allow changes made to the internal statistics to make the experience more enjoyable. Compared to the following two areas; this does not have a higher priority.
- User Interface & Accessibility: The means of interaction between the user and the product and ease of access between the two. Testing these areas is essential for providing the user with the most efficient means of interpreting valuable data. Accessibility will focus on the difficulty that Users with visual impairments may have when attempting to interpret the User Interface. Testing this will broaden the appeal range by allowing them to focus on the product without the need to struggle through interpreting the Interface.

Reasons for the involvement of users during this stage is simply that standard software testing methods cannot provide the results needed to ensure that these areas are viable for the variety of potential players. Testing these areas without the use of an outsourced view can result in the product being unable to meet the desired standards set by a large majority of the strategy gaming community.

5.2.1 Participants

Participants of the testing of the product will take part in a survey, which provides them with a means of leaving feedback related to the areas focused on. Survey Monkey (Finley and Finley, 1999) will host the survey to collect and graphically display data in an efficient manner.

5.2.1.1 Gameplay

Those who will be participating in the survey would already have an interest in computer games. This selection of users will allow the collection of data based on comparisons between this experience and their experiences with games of this genre. Gameplay analysis shall also cover the functionality of the product and whether

5.2.1.2 User Interface

The reason for the analysis of this area is to gain an understanding of how easily the users can understand the interface without the need for interference and time dedicated to teaching the user how the Interface functions. In addition to this, the participants will play a role in deciding if the placements of the interface elements are adequate for the gameplay and if they provide the necessary feedback.

5.2.2 Testing Method

The testing method for this project is simply to send out copies of the latest build leaving a '.txt' file within the folder which will contain instructions on how to install/play the game and also a link to the survey created on Survey Monkey for the user to complete. In doing this, the need for tutoring on the game's mechanics is no longer required unless it is warranted through errors being encountered by the participants.

5.3 Survey Response Analysis

Following the completion of the user testing, a total of 7 users had left feedback using the online questionnaire provided. Results originating from the first section of the questionnaire did provide valuable insight into the habits and interests of the target audience in that they take part in gaming based activities on a regular basis. The habit of these users would suggest that they partake of their games for long-periods of time rather than in short bursts. This implies that the choice to have some randomized factor within the product as a means of elongation and implementing a form of unpredictability may not go unappreciated should a project such as this be attempted once more.

Despite the encouraging answers from the first section of the questionnaire, results collated from the gameplay section of this survey were not as favourable as one could have hoped for, which became apparent in the number of users who answered 'No' when questioned on the functionality of the product (Appendix D, Question 5). This answer was to be expected due to the program becoming progressively unstable as the code for the product increased in both volume and complexity.

5.4 Black Box Testing

This section of the evaluation is dedicated to the recording of both the testing of the main classes within the product and the changes that follow. Any evidence for these tables are stored within Appendix C and references to any figures should be directed to the location specified within the Appendix.

Table 1. Tile Map Functions

5.4.1 Tile Map Functions

Index	Tested Area	Expected Outcome	Actual Outcome	Notes
1.	Left clicking an unoccupied tile.	Tile state changes to selected, colour of tile changes to red.	As expected.	
2.	Left clicking an occupied tile, not the unit occupying said tile.	Tile state changes to selected, colour of tile changes to red.	As expected.	
3.	Left clicking a unit occupying a tile	State of occupied tile changes to selected, colour of tile changes to red.	Error; null reference exception.	Fixed, now functions as expected.

4.	Left clicking a blocked tile.	Tile state does not change; colour of tile remains the same.	As expected.	
5.	Generating single dimension path.	Simple path generated.	As expected.	Fig C.1.1
6.	Generate two-dimensional path.	Correct path generated	As expected	Fig C.1.2
7.	Generate short two dimensional path involving obstacles.	Optimal path generated.	As expected	Fig C.1.3
8.	Generate long two-dimensional path; involving obstacles.	Optimal path generated.	Path generated, not optimal; deviations generated.	Fig C.1.4
9.	Generate impossible path.	Error shows; path not possible.	Game crashes.	Add path length limits to pathfinding algorithm.

Table 2. Unit Interaction

5.4.2 Unit Interaction				
Index	Tested Area	Expected Outcome	Actual Outcome	Notes
1.	Left click on player unit or tile occupied by player unit.	Unit information is displayed on bottom-right and right panels. Movement range is shown.	As expected.	Fig. C.2.1
2.	Left click on enemy unit or tile occupied by enemy unit.	Unit information is displayed on bottom-right and right panels, shows that unit is an enemy. Movement range is not shown.	Unit information is displayed; movement range is show.	Fixed, now functions as expected.
3.	Right click on empty tile while player unit is selected.	Unit is moved along generated path to selected tile.	As expected.	Fig. C.2.2

4.	Right click on empty tile while enemy unit is selected.	Unit remains stationary, no path is generated.	Path is generated, unit moves along the path.	Fixed, now functions as expected.
5.	Right click on tile occupied by obstacle with player unit selected.	Unit remains stationary, no path is generated.	As expected	

Table 3. User Interface

5.4.3 User Interface

Index	Tested Area	Expected Outcome	Actual Outcome	Notes
1.	Select a 'Squad' unit.	Squad relevant data is displayed.	Some information is missing.	Fixed, now functions as expected.
2.	Select a 'Vehicle' unit.	Vehicle relevant data is displayed.	As expected.	
3.	Turn indicator. Player to Enemy.	Enemy turn indicator becomes active	Indicator changes to enemy very quickly.	Add 'wait' functionality for AI

		when turn changes.		
4.	Turn indicator. Enemy to Player.	Player turn indicator becomes active when turn changes	Turn indicator remains the same.	Issue unrelated to interface script, error originated in Artificial Intelligence script.

Table 4. Unit Generation

5.4.4 Unit Generation

Index	Tested Area	Expected Outcome	Actual Outcome	Notes
1.	Connecting to database within asset folders.	Database connection successful	Database not found.	Fixed, now functions as expected.
2.	Connect to 'Soldiers' table.	Table connection successful	Table not found.	Fixed, now functions as expected.
3.	Reference data from 'Soldiers' table.	All data correctly referenced.	Some data correctly referenced, others	Fixed, now functions as expected.

			improperly casted to target storage variable.	
4.	Connect to ‘Vehicles’ table.	Table connection successful	As expected	
5.	Reference data from ‘Vehicles’ table.	All data correctly referenced.	As expected	
6.	Connect to ‘Weapons’ table.	Table connection successful	As expected	
7.	Reference data from ‘Weapons’ table.	All data correctly referenced.	As expected	

6. Conclusion

Bringing this project to a close, overall it can be said that opinions directed towards the end-result are less-than favourable. As was made apparent in the survey feedback, the developed project encountered numerous errors which made the functionality both unpredictable and disappointing for certain users. This can be due to their hardware, as in the case of GUI errors, or their misfortune in the event of the product malfunctioning.

With hindsight, it may have been more beneficial to plan ahead utilising pseudo-code algorithms rather than improvising most of the algorithms as the project was developing. What became apparent was the resulting instability of the program as previously mentioned with regards to the survey.

Overall the developed product has been a personal failure, but it is hoped that this experience will lead to an improvement in future projects, encourage a more solid “plan of action” and, in turn, an enhanced performance outcome.

References

Goldhawk Interactive. (2014) *Xenonauts.* PC. [Game] [Accessed on 19th October 2016] http://store.steampowered.com/app/223830/

Square Product Development Division 6 (Square Enix). (Japan: 1999, North America: 2000, Europe: 2000) *Front Mission 3*. Playstation. [Game] Japan: Square, North America: Square Electronic Arts, PAL: Square Europe

Goldman, T. (2011) *Indie Studio Takes X-Com Into Its Own Hands With Xenonauts*. 12th May. The Escapist. [Online] [Accessed on 19th October 2016] http://www.escapistmagazine.com/news/view/109979-Indie-Studio-Takes-X-Com-Into-Its-Own-Hands-With-Xenonauts#&gid=gallery_239&pid=1

Brian. (2000) *Front mission 3 review*. 1st April. Game Revolution. [Online] [Accessed on 19th October 2016] http://www.gamerevolution.com/review/front-mission-3

Konami. (2001, EU) *Ring of Red*. Playstation 2. [Game] Konami.

Unity Technologies. (2005) *Unity* (Version 5.3.2) [Computer Program]. Available at https://www.unity3d.com (Accessed 02 November 2016)

Epic Games. (1998) *Unreal Engine* (Version 4.12.5) [Computer Program]. Available at https://www.unrealengine.com (Accessed 04 November 2016)

Microsoft. (1997) *Visual Studio* (Version 2015) [Computer Program]. Available at https://www.visualstudio.com/ (Accessed 08 November 2016)

Adobe Systems. (2003) *Adobe Photoshop* (Creative Suite 6) [Computer Program]. Available at http://www.adobe.com/uk/ (Accessed 11 November 2016)

Kimball S. & Mattis P. (1995) *GIMP* (Version 2.8.18) [Computer Program]. Available at https://www.gimp.org/ (Accessed 12 November 2016)

Autodesk Inc. (1998) *Autodesk Maya* (Version 2016) [Computer Program]. Available at http://www.autodesk.co.uk/ (Accessed 15 November 2016)

Games Workshop Limited. (2016) *Games workshop Webstore*. Games Workshop. [Online] [Accessed on 22nd November 2016] https://www.games-workshop.com/

Hipp, D. (2017) *SQLite Home Page*. Sqlite.org. [Online] [Accessed on 16 March 2017] https://www.sqlite.org/.

Madhav, S. (2013) *Game programming algorithms and techniques: A platform-agnostic approach*. Boston, MA, United States: Addison-Wesley Educational Publishers.

Bibliography

Eaddy, M. (2001) 'Do we really need another language?' *C# Versus Java*, February.

Warwick, K. (2011) *Artificial intelligence: The basics*. New York: Routledge.

Lecky-Thompson, G. W. (2008) *AI and artificial life in video games*. Boston, MA: Charles River Media/Cengage Technology.

Eaddy, M. (2001) 'Do we really need another language?' *C# Versus Java*, February.

Appendix – A

Terms of Reference

Course specific Learning outcomes:

1. To study the history of computer games, game genres, game structures and game design principles and to use the skills acquired to specify and evaluate new game applications.
2. To become knowledgeable in the use and development of computer graphics software/game middleware tools and to be able to apply this knowledge to the implementation of real-time interactive systems.
3. To learn structured approaches to computer programming.
4. To learn how the theories and techniques of behavioural systems can be used to enhance the playability and sophistication of computer games.
5. To study a range of mathematical tools and techniques and to be able to use these, in particular, to solve problems in the area of game modelling and animation.
6. To gain an appreciation of the multidisciplinary environment in which commercial games are designed and produced and to acquire skills in project management and team working.

Project Background:

This project is dedicated to creating a 2D tile based game of any possible genre, the outcome of this project should result in a playable full game or a demo, which would demonstrate the full range of game mechanics, made available to the player.

Examples of 2D tile games can be as simple as draughts or chess, each using rules that confines the movement of the playing pieces to the tiles on the board. While these are good starting examples, what will hopefully be achieved in this project is something more complex in terms of gameplay. As a result, what was found was that two examples of strategic games shared many similarities to the aims of this project.

Xenonauts

Xenonauts (Goldhawk Interactive, 2014) is an isometric tile-based tactical strategy game developed as a spiritual successor of the same genre 'UFO: Enemy Unknown'.

Figure A.1 Screen capture of the start of a standard mission in Xenonauts (Tom Goldman, 2011 Online)

Although this cannot technically be counted as a 2D game due to its use of elevation, making it a 3D tile based game, it still utilises many of the mechanics and structures that are to be used as inspiration in this project. Some examples of these are:

- Time units; a resource used by all types of units which can determine how far they can move and can also decide if and how well they can shoot.

- Soldier stats; descriptions of each soldier's combat ability and how well they would perform some tasks such as moving, shooting and carrying heavy equipment.
- Hit chance; a statistic displayed to show the player the percentage chances that the shot of the unit will hit the desired target based on the soldier's stats and the type of shot chosen (based on time units).

Front Mission 3

Front Mission 3 (Square Product Development Division 6, 1999) is a tactical, turn-based strategy game with an emphasis on using giant, in-directly pilotable robots known as 'Wanzers', to fight and defeat enemies who may use Wanzers or other armoured vehicles.

Figure A.2 The cinematic view of a player unit (left) being attacked by an enemy unit (right) in Front Mission 3. (Brian, 2000 Online)

Like Xenonauts, this game cannot be counted as a 2D strategy game as it incorporates the elevation of models into its combat; increasing or decreasing the hit-chance depending on the weapon. Regardless of this, Front Mission 3 offers gameplay mechanics that the project will use as inspiration for part of this project. As previous, the following are some examples of the mechanics the project will be focusing on:

- Selective damage; as an alternative to the unit's health being represented by a single bar, in its place the user is given five bars, which represent each of the unit's body parts (Figure A.2). Each of the health bars have different effects on the unit should the respective bar become depleted, e.g. if the bar representing the health of the legs is depleted the unit's movement range is decreased or immobilised but is still able to shoot given that they have a functioning ranged weapon.

- Battle cinematics; as seen in Figure A.3, when a unit engages another in combat a short cinematic animation showing the units attacking with weapons previously selected in the tactical view also showing in real-time, the damage that is being applied to the parts of the units.

Aim:

The aim of this project is to create a single level to be used as a demo for a 2d tile-based strategy game founded on the idea of using walking tanks set in a war themed world.

Objectives:

Research

1. Play and document games which have a relevant theme, genre or gameplay mechanics.
   - Is the game fun and/or successful?
   - If so, how does the game make use of its assets and mechanics?
2. Read and document material on relevant game mechanics and design techniques, specifically pathfinding.
   - Which pathfinding technique is most efficient?
3. Read and document material on Artificial Intelligence.
4. Try and compare different game engines, what kind of code do they use?
   - C#, C++, Java or other.
5. Try and compare different modelling software for 3D assets.
   - Which can export in the required format? (dependent on game engine)
6. Try and compare different design software for 2D assets.
   - Exported formats must be either or include: Png, Bmp, Jpg or Gif.

Development

1. Create basic traversal demo.
    - Add an optimal pathfinding system.
    - Test system with variety of factors, e.g. obstacles, complex movement.
2. Design and program unit classes and objects.
    - How many units and types of units will there be?
    - Plan and designate statistics for each unit and unit type.
    - Are the units balanced for fair gameplay?
3. Apply the units and objects to the traversal demo as a new demo.
4. Design and develop game mechanics which allow units to target and damage each other.
5. Design, create and apply a User Interface (UI) based on suitable heuristics of usability.
6. Split units into 2 teams, player controlled and AI controlled.
7. Create a basic Artificial Intelligence which moves the units belonging to the respective team.
8. Expand the Artificial intelligence to move units into range and attack where possible.

Evaluation

1. Find users who have an interest in this genre of games and ask them to participate in the survey
2. Setup an appropriate evaluation system for data collection.
3. Create questionnaires which enquire about key elements of the game.
    - Enquire about the usability of the interface.
4. After developing a functional prototype & UI, allow participants to playtest the prototype and afterwards provide feedback using the questionnaire.

5. Improve prototype based on feedback.
6. Repeat Evaluation objectives 4 and 5 after each game mechanic implementation.
7. Document feedback and any changes made.
8. Conclude Results.

Additional Objectives

1. Expand Development objective number 4 to allow the units to target specific parts of other units.
    - Expand further, create effects to be applied depending on the destroyed/damaged part.
2. Create a narrative which can be applied to the demo as a segment of and/or the full story.
3. Create additional levels for the demo.
4. Expand the Artificial Intelligence by allowing it to attack and move based on the opposing unit types.
    - Could also expand around the use of the specific part targeting if possible

Potential Issues

When using game engines, one must always take into account the unpredictability of a development plan. As a result, this chart has been created so as to specify what actions must be taken in the event of these risks coming to fruition.

Table 5. Potential Issues

Risk	Probability	Effect	Mitigation
Hardware Failure	Minor	Major	Backup Code and Documents on regular

			basis to separate storage device.
Unable to implement advanced targeting system	Moderate	Moderate	Fulfil original objective prior to attempting.
Unable to create story	Moderate	Minor	Create simple, one paragraph narrative as a backup or starting point.
Unable to enhance the Artificial Intelligence	Major	Minor - Moderate	Implement original Artificial Intelligence, regardless of complexity.
Unable to design additional levels	Major	Minor	Have single level demo ready (**mandatory**)

Required Resources:

The majority of essential resources for this project will be software necessary for designing the code and assets. For this a selection from the following will be required:

- Game Engine (Self-Made or otherwise), must be able to export the final product to desired platform(s) through an in-built system or other means.
- 3D Modelling software (if utilising 3D models in final build), must be able to export the 3D asset in a format recognizable by the game engine.
- Art & Design Software (for 2D assets).

- Code Editing Software.

Timetable and Deliverables:

The plan is to meet with the supervisor of this project on a weekly basis so as to remain up-to-date on which section of the final report should be a priority and keep to a personal timetable as described below.

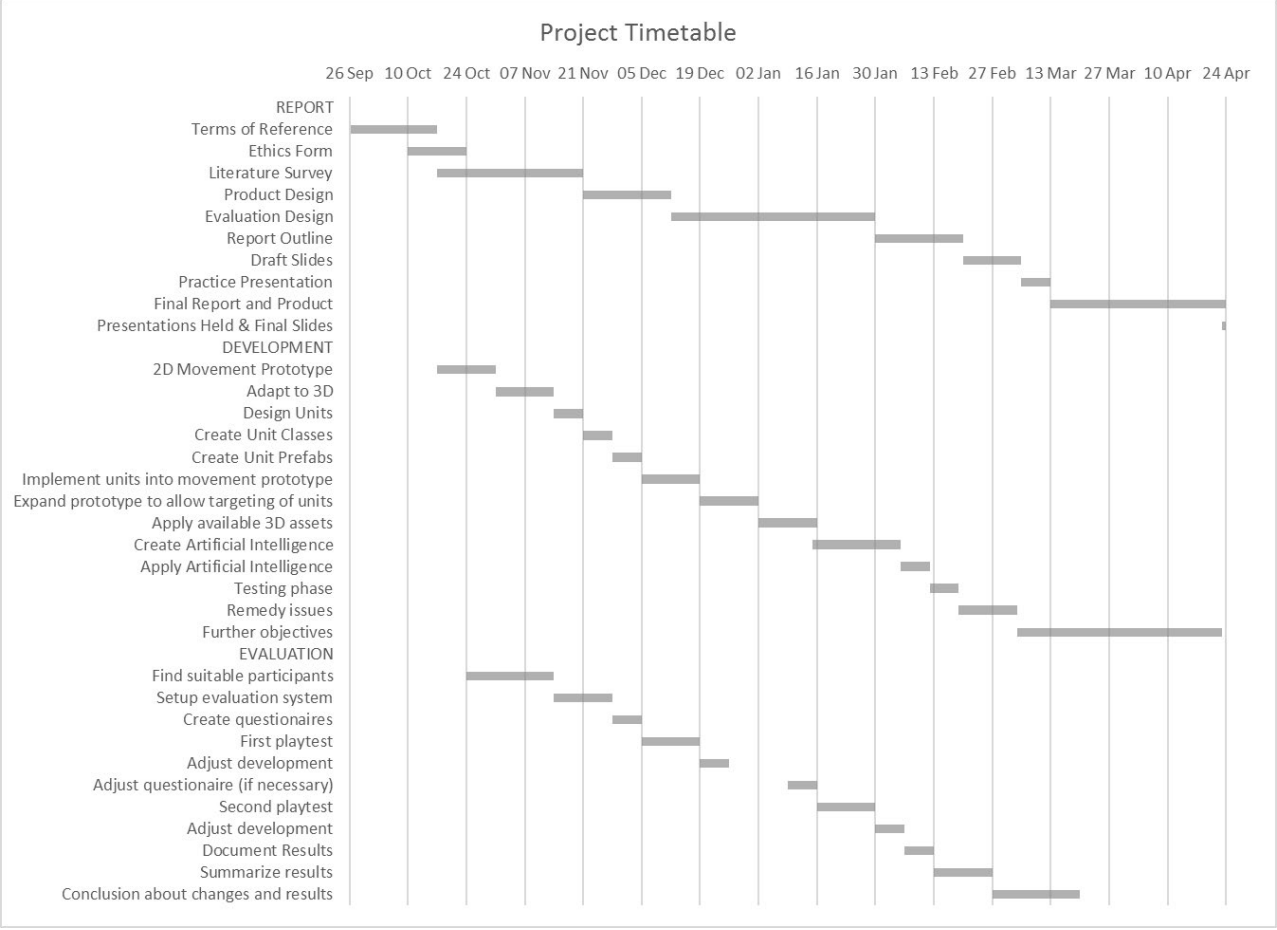


Figure A.3: Project Timetable

Ethics Checklist

[redacted]

[redacted]

Appendix – B

Figure B.1

```
(Listing 9.2)
currentNode = startNode
add currentNode to closedSet
do
        foreach Node n adjacent to currentNode
                if closedSet contains n
                        continue
                else if openSet contains n // Check for adoption
                        compute new_g // g(x) value for n with currentNode as parent
                        if new_g < n.g
                                n.parent = currentNode
                                n.g = new_g
                                n.f = n.g + n.h // n.h for this node will not change
                        end
                else
                        n.parent = currentNode
                        compute n.h
                        compute n.g
                        n.f = n.g + n.h
                        add n to openSet
                end
        loop
        if openSet is empty
                break
        end
        currentNode = Node with lowest f in openSet
        remove currentNode from openSet
        add currentNode to closedSet
until currentNode == endNode
 (Listing 9.1)
If currentNode == endNode
        Stack path
        Node n = endNode
        while n is not null
                push n onto path
                n = n.parent
        loop
else
        // path unsolvable
end
```

Figure B.1 Extract of A algorithm pseudocode from Listing 9.2 (Madhav, 2013:191-192) and Reconstruct algorithm from Listing 9.1 (Madhav, 2013:189)*

Figure B.2

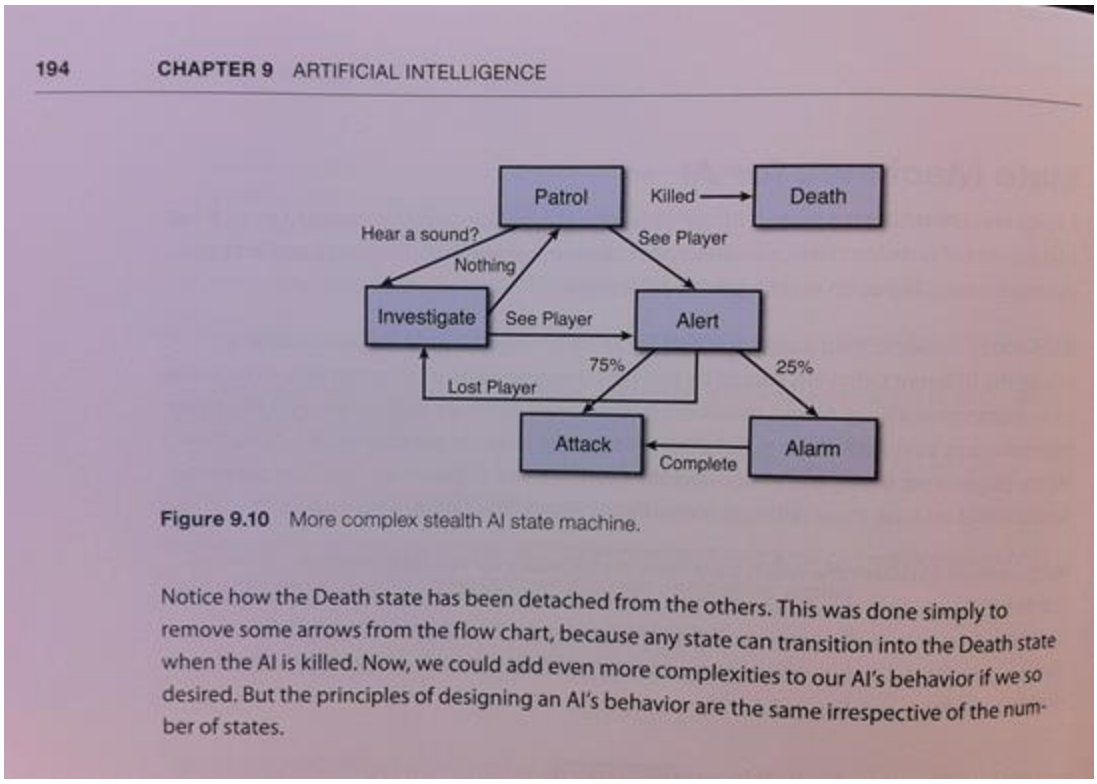


Figure B.2 Example of advanced AI state machine representation (figure 9.10) (Madhav, 2013:194)

Figure B.3

	WS	BS	S	T	W	I	A	Ld	Sv
Space Marine	4	4	4	4	1	4	1	8	3+
Ork Boy	4	2	3	4	1	2	2	7	6+

Figure B.3 Re-creation of extract from Warhammer 40,000 Rulebook (page 9) by Gamesworkshop (2016).
Example of statistical representation for two soldiers, the Space Marine and the Ork Boy.

Appendix – C: Testing Evidence

Figure C.1.1

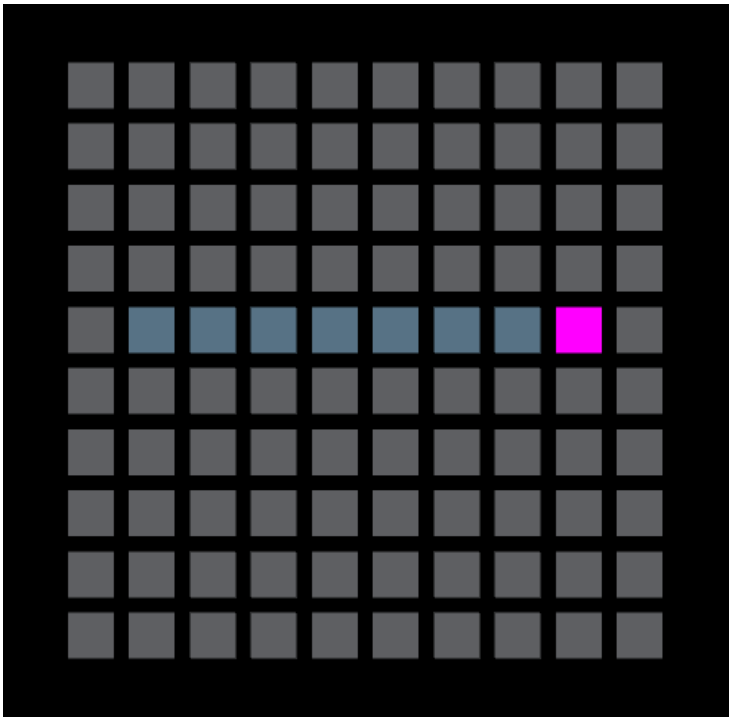


Figure C.1.2

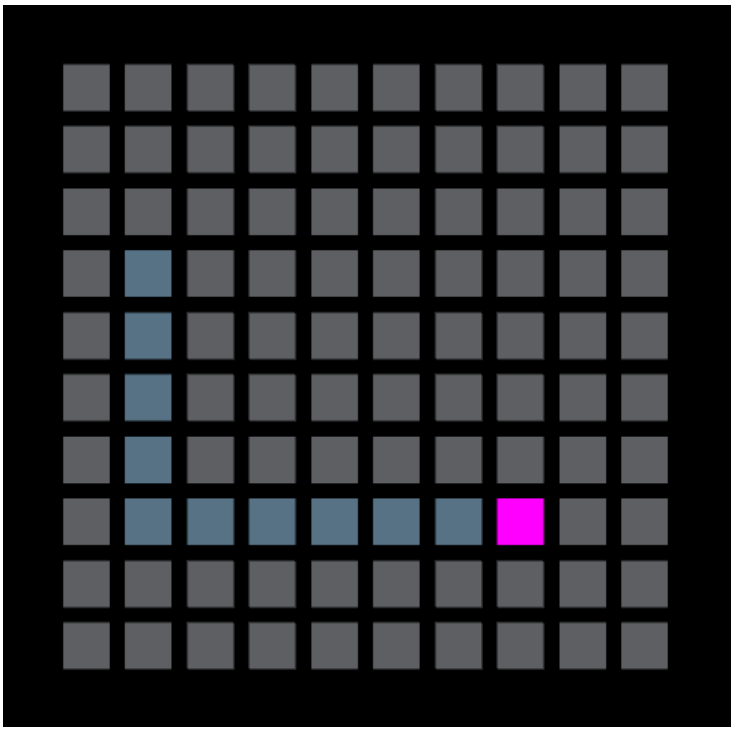


Figure C.1.3

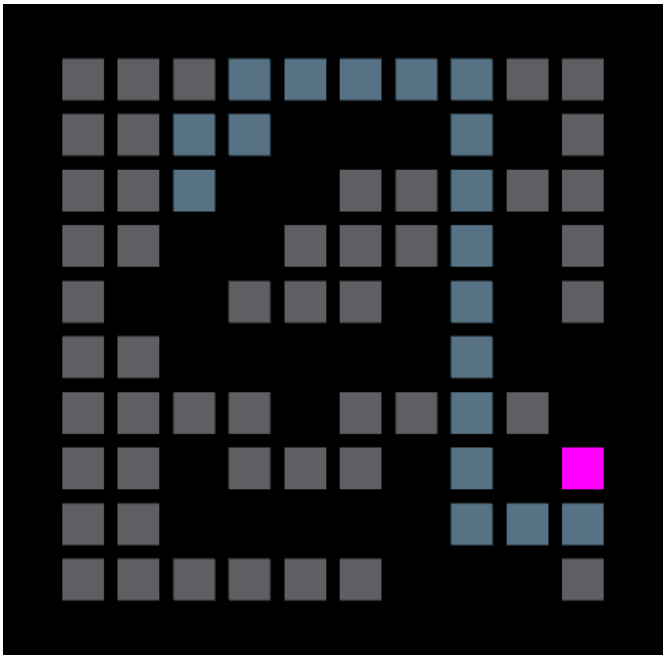


Figure C.1.4

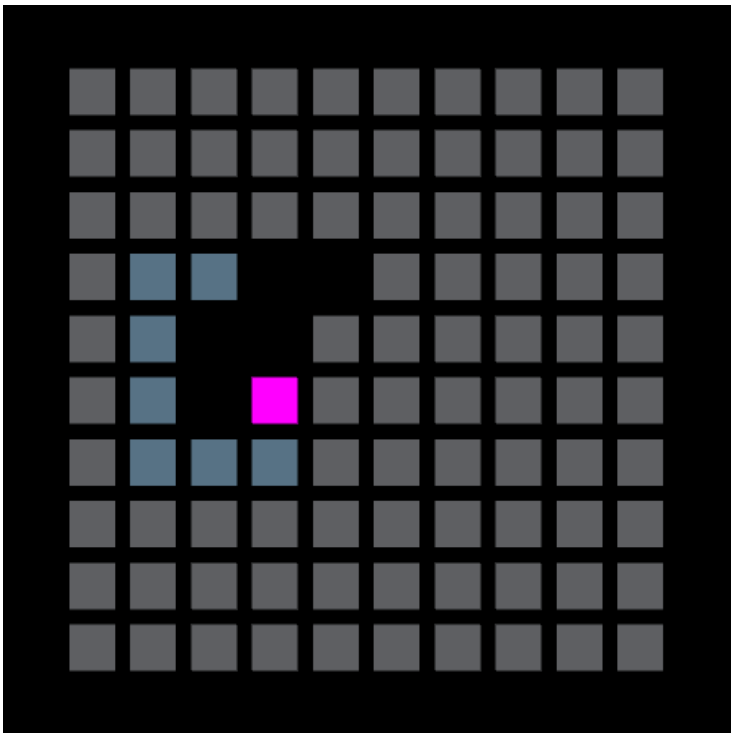


Figure C.2.1

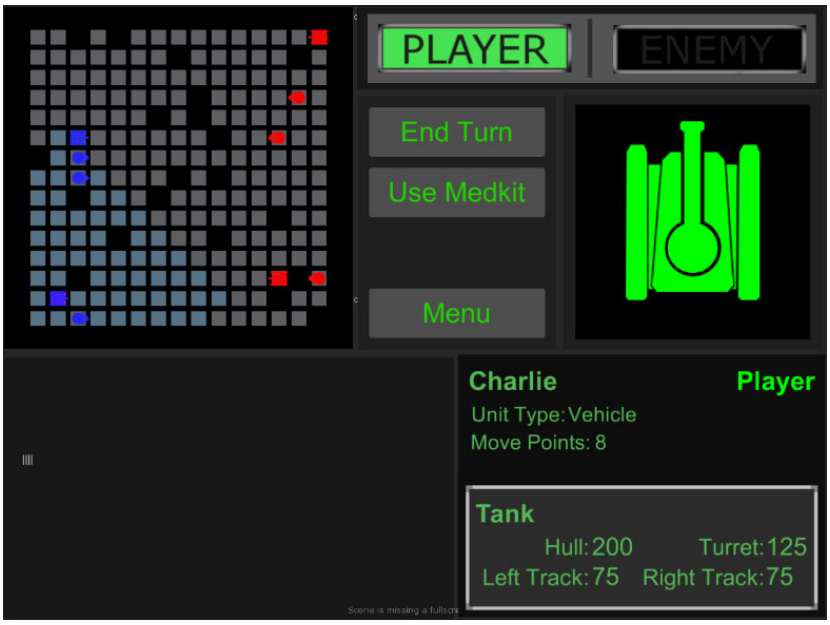
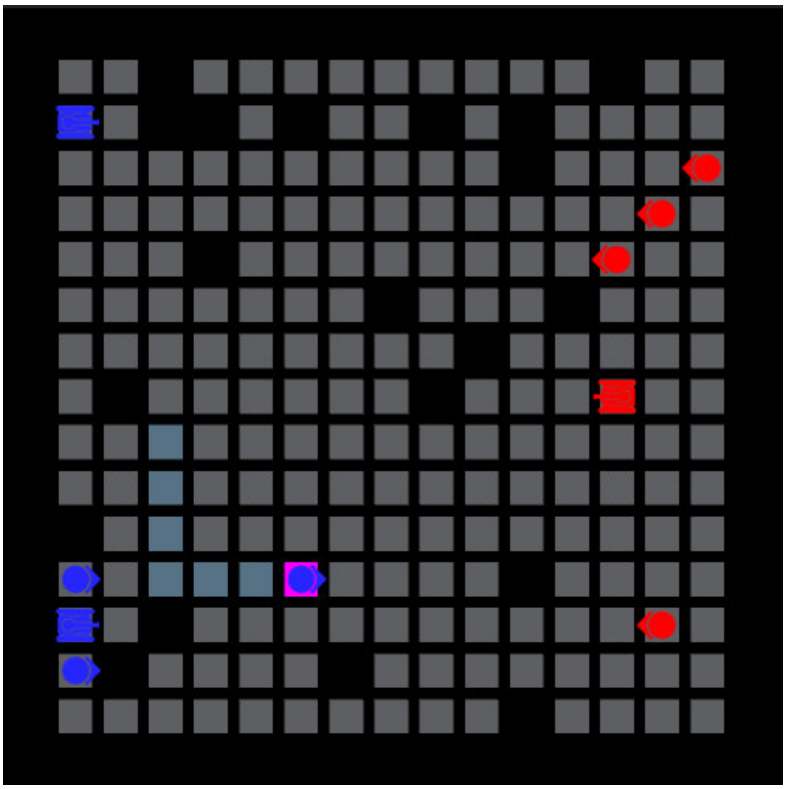


Figure C.2.2



Appendix – D: Questionnaire and Results

Questionnaire – Page 1: General Enquiries

This questionnaire will aid in gaining feedback from the participant's experience with the product, the data collected will not include any personal information and cannot be linked to the participant in any way.

Question 1:

How often do you play computer/video games?

- Often
- Sometimes
- Rarely
- Never

Question 2:

On what platforms do you play computer/video games? (Check all that apply)

- Computer/Laptop
- PlayStation 3
- PlayStation 4
- Xbox 360
- Xbox One
- Nintendo Wii
- Nintendo Wii U

- Mobile Phone
- Other (Please specify)

Question 3:

How would you describe your skill/experience with games? (Please check one)

- Experienced
- Average
- Unsure
- Novice
- No skill

Question 4:

What genre of games do you play, if any?

- Action
- Puzzle
- Strategy
- Role-playing
- Racing
- Adventure
- Other (Please specify)

Questionnaire – Page 2: Gameplay

Question 5:

Did this game function properly from start to finish for you?

- Yes
- No (please specify)

Question 6:

The controls were easy to understand.

- Strongly agree
- Agree
- No opinion
- Disagree
- Strongly disagree

Question 7:

The Graphical User Interface was well designed.

- Strongly agree
- Agree
- No opinion
- Disagree
- Strongly disagree

Question 8:

The game was fair and well balanced.

- Strongly agree
- Agree
- No opinion
- Disagree
- Strongly disagree

Question 9:

You understood what events were occurring at any given time.

- Strongly agree
- Agree
- No opinion
- Disagree
- Strongly disagree

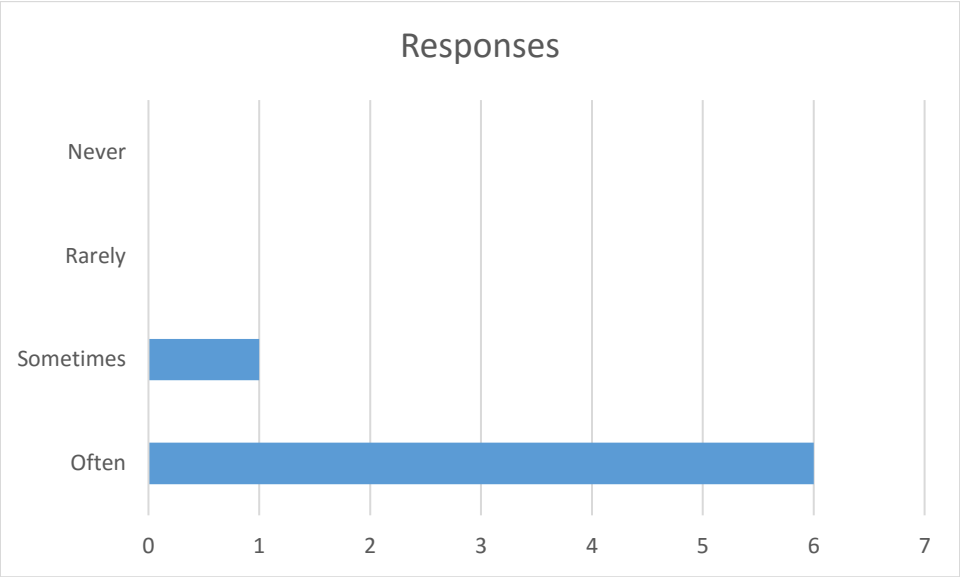
Question 10:

Do you have any suggestions in which improvements can be made?

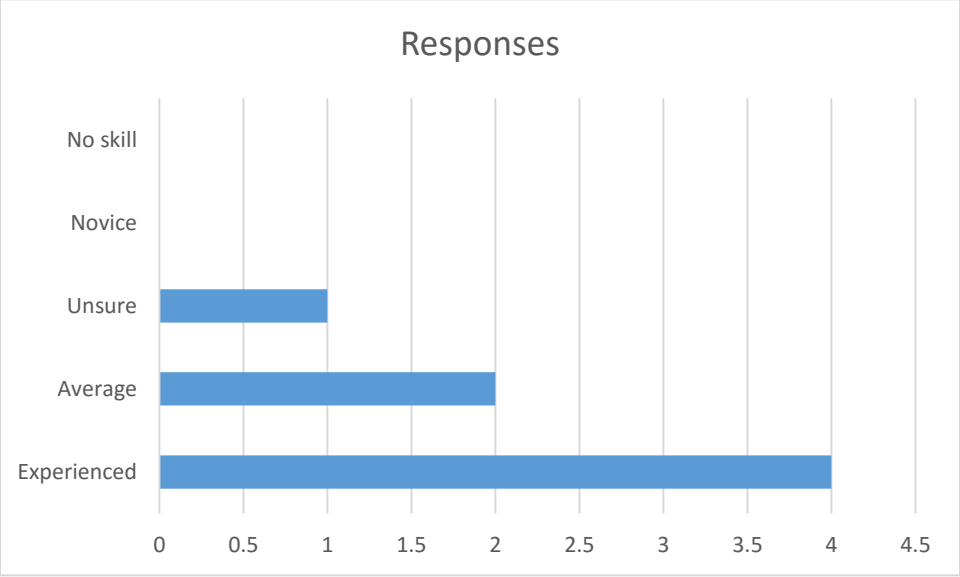
- No
- Yes (Please specify)

Questionnaire Results

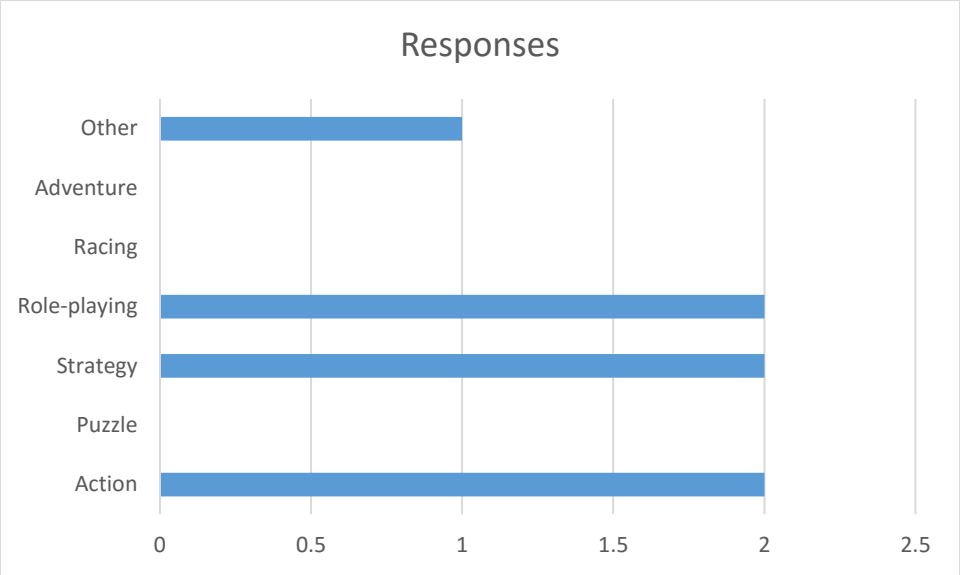
Question 1:



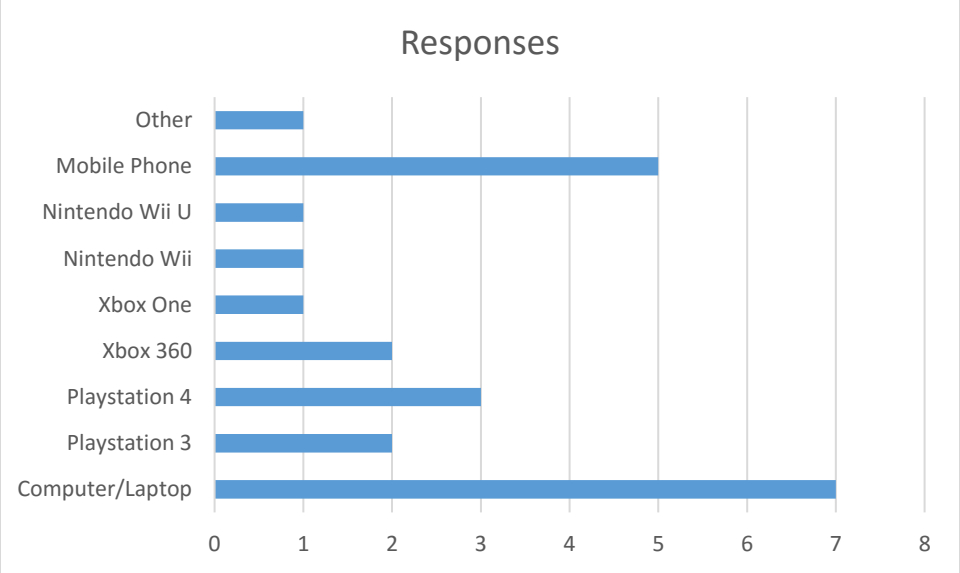
Question 2:



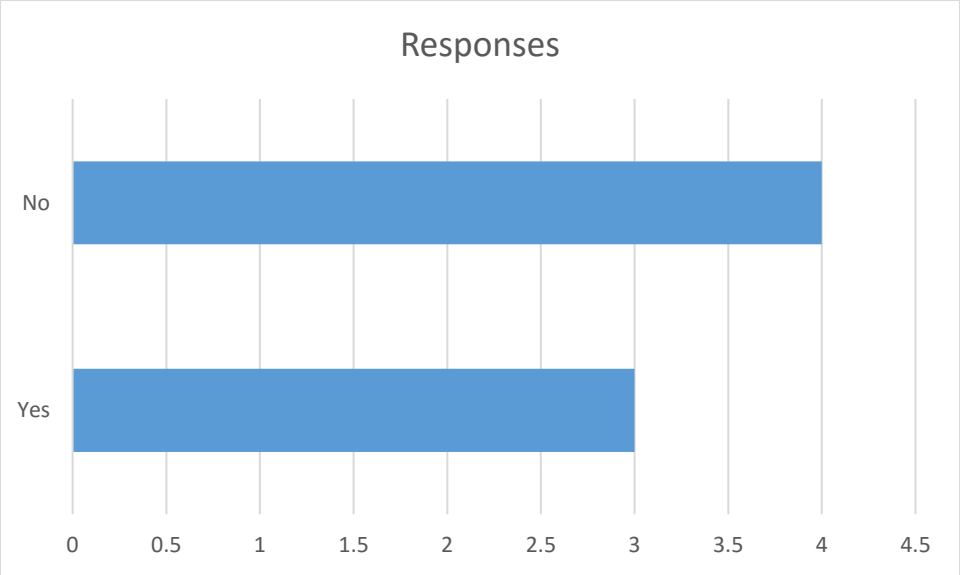
Question 3:



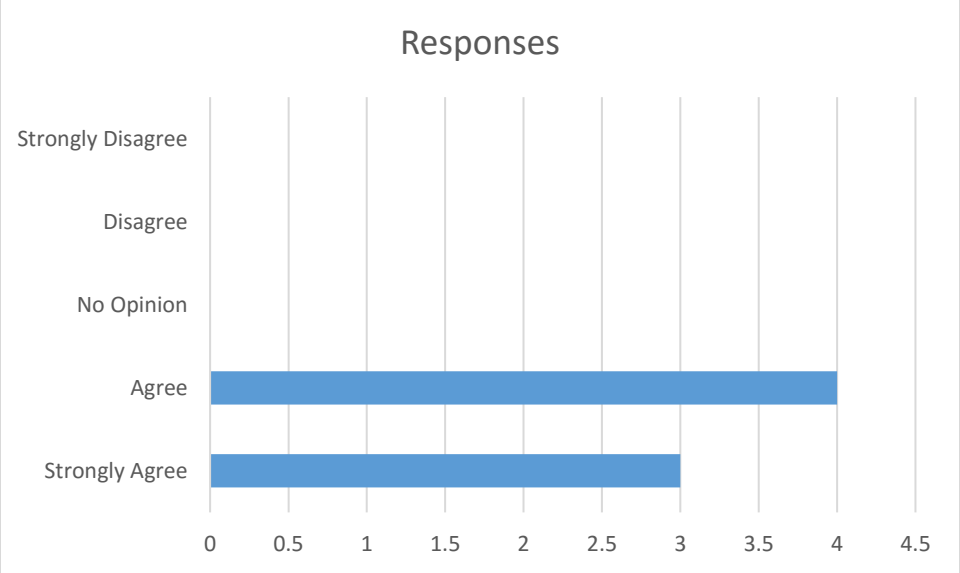
Question 4:



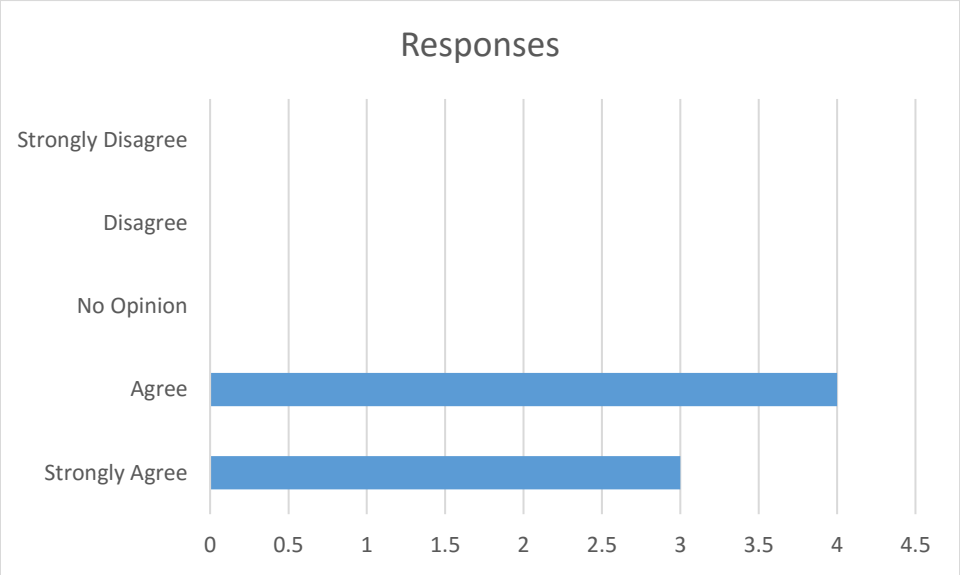
Question 5:



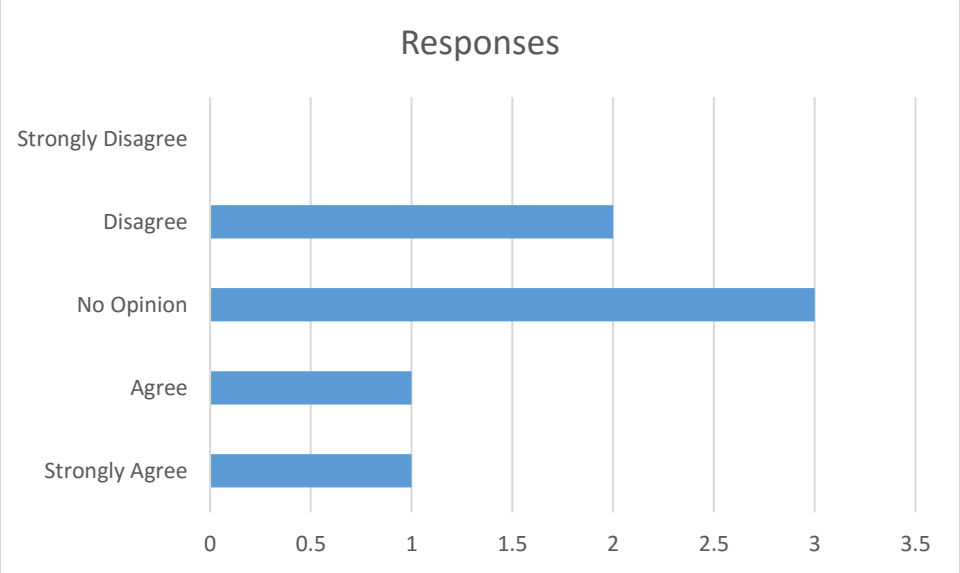
Question 6:



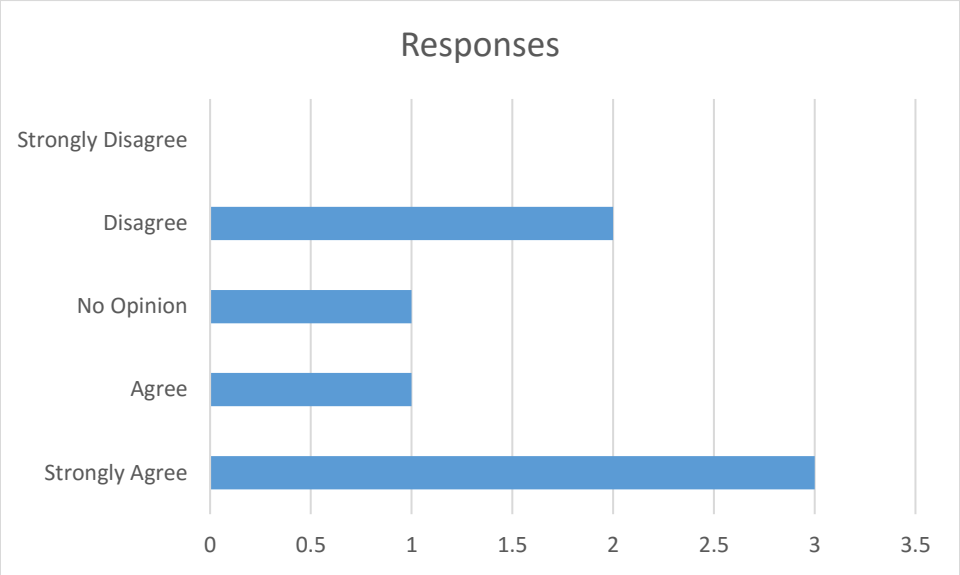
Question 7:



Question 8:



Question 9:



Question 10: