**The need for a GUI desktop application to aid in the building of an army roster list in Warhammer 40,000**

**Grant Rigby – P15223403**

**IMAT3451 – Computing Project**

**Computer Science BSc (Hons)**

**27th of February 2018**

****

Abstract

For a long time players of the tabletop wargame Warhammer 40,000, have been creating army rosters in a tedious and time consuming manner. This involves manually looking up the point values of the models and their equipment in the faction’s codex book. These point values are totalled to ensure they don’t surpass a chosen point limit. Traditionally this process is achieved by using a calculator and a pen and paper, which can be difficult due to the constraints of particular models. In order to alleviate this problem it has been decided to create a GUI program that performs the point calculation automatically, as well as ensuring all the constraints of the faction are met.

Table of Contents

[1. Introduction 4](#_Toc511305609)

[1.1 Program Overview 4](#_Toc511305610)

[1.2 The need for an army roster builder in Warhammer 40,000 8](#_Toc511305611)

[1.3 Objectives of the Project 8](#_Toc511305612)

[2. Analysis of Requirements 9](#_Toc511305613)

[2.1 Program Functionality 9](#_Toc511305614)

[2.1.1 Functional Requirements 9](#_Toc511305615)

[2.1.2 System Requirements 10](#_Toc511305616)

[2.2 Interface Elements 10](#_Toc511305617)

[3. Design Considerations 13](#_Toc511305618)

[3.1 Major Design Decisions 13](#_Toc511305619)

[3.1.1 Decisions regarding the structure of the code 13](#_Toc511305620)

[3.1.2 Decisions regarding the appearance of the user interface 14](#_Toc511305621)

[3.2 Armoury of Weapons 15](#_Toc511305622)

[3.3 MVC Structure 16](#_Toc511305623)

[3.3.1 Model Structure 16](#_Toc511305624)

[3.3.2 View Structure 16](#_Toc511305625)

[3.3.3 Controller Structure 17](#_Toc511305626)

[4. Implementation 17](#_Toc511305627)

[4.1 Unit and UnitSquad classes 17](#_Toc511305628)

[4.2 Controller class 18](#_Toc511305629)

[4.3 Weapons and WeaponList classes 19](#_Toc511305630)

[4.4 Testing 19](#_Toc511305631)

[4.4.1 Test Plan 19](#_Toc511305632)

[4.4.2 Test Results 19](#_Toc511305633)

[5.Conclusions 19](#_Toc511305634)

[5.1 Program Evaluation 19](#_Toc511305635)

[5.2 Project Management 19](#_Toc511305636)

[5.3 Tools Used 19](#_Toc511305637)

[5.4 Final Thoughts 19](#_Toc511305638)

[Bibliography 20](#_Toc511305639)

[Appendices 21](#_Toc511305640)

[Appendix 1A – Fields and constructor of the Unit super class 21](#_Toc511305641)

[Appendix 1B - Fields and constructor of the UnitSquad super class 21](#_Toc511305642)

[Appendix 1C – Example of a Unit sub-class with one weapon 22](#_Toc511305643)

[Appendix 1D – Example of a Unit sub-class with two weapons 22](#_Toc511305644)

[Appendix 1E – Example of a UnitSquad sub-class 23](#_Toc511305645)

[Appendix 3A – User interface create profile screen 27](#_Toc511305646)

[Appendix 3B – User interface create army roster screen 28](#_Toc511305647)

[Appendix 3C – User interface add unit section 29](#_Toc511305648)

[Appendix 3D – User interface modify unit section 29](#_Toc511305649)

[Appendix 3E – User interface display army roster section 30](#_Toc511305650)

[Appendix 3F – User interface display points section 31](#_Toc511305651)

[Appendix 3G – User interface warning message section 31](#_Toc511305652)

[Appendix 3H – User interface help menu section 31](#_Toc511305653)

[Appendix 3I – User interface file menu section 31](#_Toc511305654)

[Appendix 4A – Weapon list of map data structures 32](#_Toc511305655)

[Appendix 4B – Weapon list of map data structures continued... 32](#_Toc511305656)

[Appendix 4C – Weapon list of map data structures continued... 33](#_Toc511305657)

[Appendix 4D – Weapon list of map data structures continued... 33](#_Toc511305658)

[Appendix 4E – Weapon list get methods 34](#_Toc511305659)

[Appendix 4F – Weapon super class 35](#_Toc511305660)

[Appendix 4G – Heavy weapon sub-class 35](#_Toc511305661)

[Appendix 5A – eclipse model package structure 36](#_Toc511305662)

[Appendix 5B – eclipse weapon model package structure 36](#_Toc511305663)

[Appendix 5C – eclipse view package structure 36](#_Toc511305664)

[Appendix 5D – eclipse controller package structure 36](#_Toc511305665)

[Appendix 5E – Controller, remove unit member handler 37](#_Toc511305666)

[Appendix 6A – ArmyBuilderRootPane class 37](#_Toc511305667)

[Appendix 6B – ArmyBuilderMenuBar class 38](#_Toc511305668)

[Appendix 6C – CreateProfilePane class 39](#_Toc511305669)

[Appendix 6D – AddUnitPane class sub-panes 40](#_Toc511305670)

[Appendix 6E – AddUnitPane class sub-panes populating BorderPane 40](#_Toc511305671)

[Appendix 7A – Unit Size Boundary Test Cases 41](#_Toc511305672)

[Appendix 7B – Unit total points accumulation 43](#_Toc511305673)

[Appendix 7C – Interface functionality (Combo box) 46](#_Toc511305674)

[Appendix 7D – Interface functionality (Buttons / Menu Items) 46](#_Toc511305675)

[Appendix 7E – Interface functionality (Disabled Items) 47](#_Toc511305676)

[Appendix 8A – Literature Review 48](#_Toc511305677)

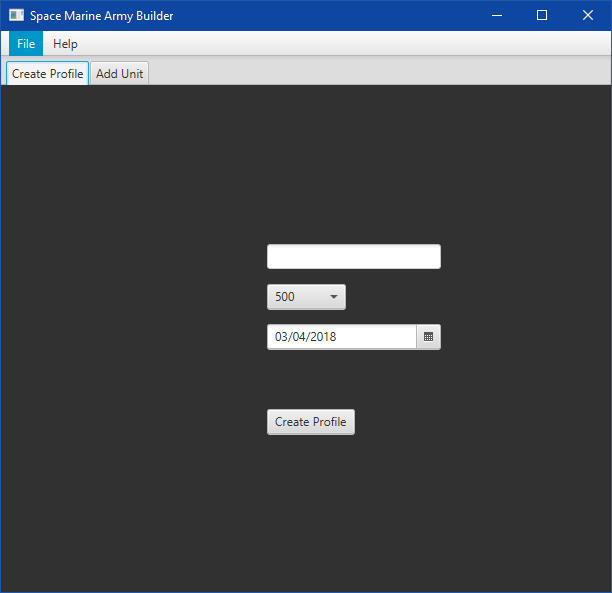
# Introduction

## Program Overview

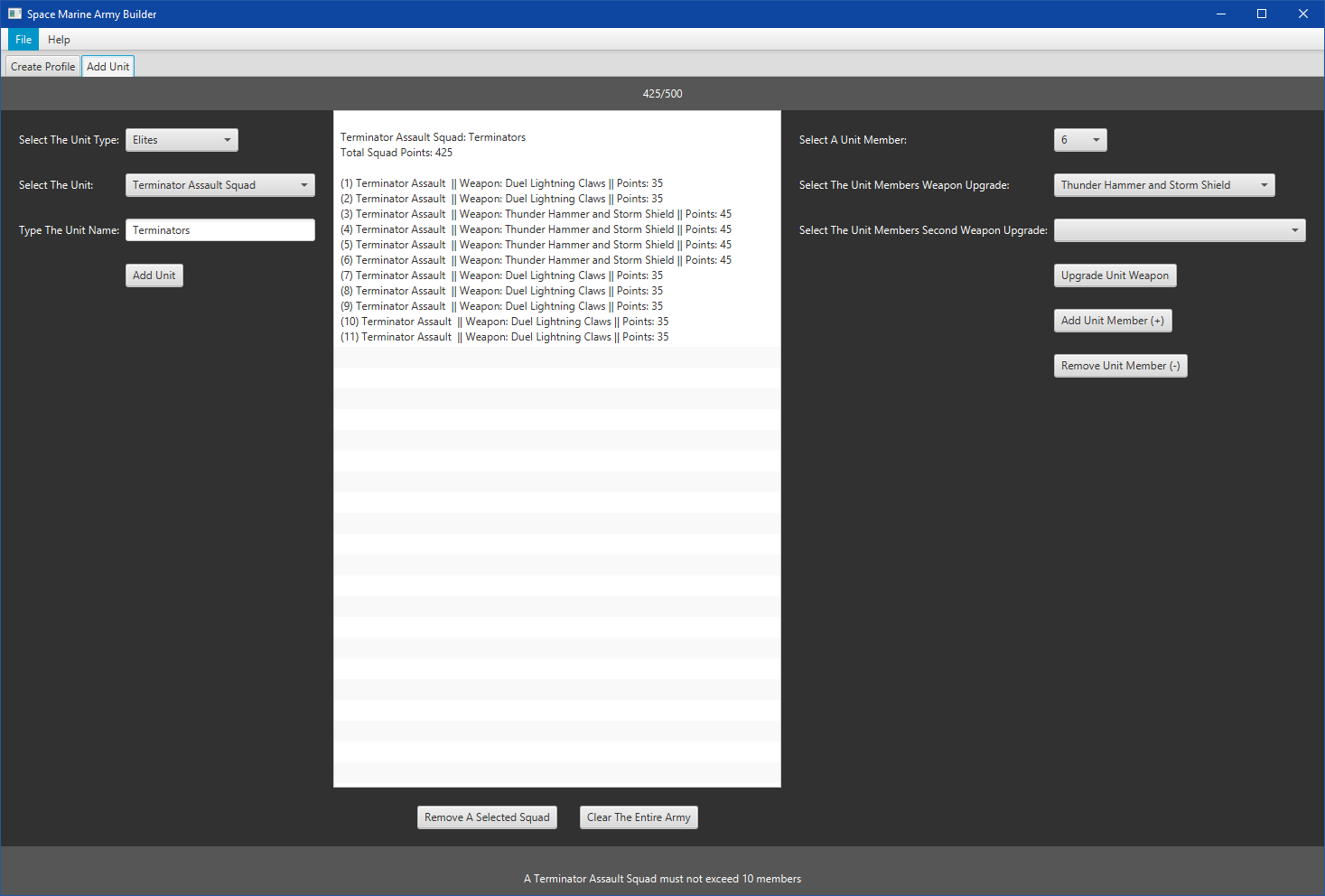
This project is to create a multi-platform desktop application that allows players of the Warhammer 40,000 tabletop game create an unbounded army roster for the popular Space Marine faction. The unbounded system is being used because there are no restrictions on the army rosters composition, meaning it is simple to use which is a favourable trait for a beginner hobbyist. The alternative was a battle forged army; however through the conclusions of the literature review (*section 4, paragraph 1)*, it was found that the battle forged approach added more complexity. The program will feature a rich and intuitive graphical user interface (GUI) using JavaFX, an interface framework for Java (*section 3.2, paragraph 1 of the literature review)* to ensure the user experience is simplified and allows a user to quickly create an army roster. Due to the nature and complexity of the Warhammer 40,000 tabletop game and in particular that of the Space Marine faction, there are a number of constraints and considerations that the program will follow to ensure accuracy and compatibility with the tabletop game. These constraints and considerations will form the basis of the programs functionality.

The main function that is in accordance to the rules of Warhammer 40,000, is the utilisation of the points system. This points system is implemented in a number of ways in the program, firstly that a user can set a point’s limit that must not be exceeded, this limit is an indication of the size of an army and ensures a user only has access to a finite amount of resources to build an army roster with *(refer to Figure 1).* Secondly the two major components of a Warhammer 40,000 army, namely the units, and equipment will each have an associated point’s value that is accumulated to reach the maximum points limit. A user can add units *(Refer to Figure 2)* with default weapons, or for additional point costs can upgrade an existing unit with weapon upgrades, as well as adding unit members, if a unit consists of multiple members. Squad members can also be removed in the same way that they can be added. The more members a unit has the more overall points it will cost.

There are also some other important functions that the program performs that are not strictly related to the Warhammer 40,000 tabletop game. These are essential convenience features that allow the user to display a temporary display of the army roster (*Refer to Figure 2*). There are also options to save the current configuration and to load the current configuration after the program has been closed. This allows the user to make changes at a later date. There is also an option to print out the most recently saved configuration on paper. Lastly, crucially as a user is adding or removing units, weapon and equipment upgrades, the changes in the total accumulated points will be visible in real time. This will be a point counter that constantly updates itself, and will give a user an indication of how many points are currently used out of the user set point limit. This will ensure a user won’t exceed the point limit.



*Figure 1 - Depicting the create profile pane, that allows a user to name and set the point limit for an army.*



*Figure 2 - Depicting a snapshot of the Add Unit panel. Here a unit type and a unit can be selected and the name unit can be specified, the display output of an army roster list is also present. Note that this is only a single unit squad for demonstration purposes. There are also options to upgrade a unit squads weapons as well as adding and removing squad members.*

## The need for an army roster builder in Warhammer 40,000

There are a number of reasons why implementing a GUI desktop application is necessary in order to build an army roster list for a Warhammer 40,000 army. Firstly because the traditional approach to building army rosters involved manually calculating the point cost by looking up the units, weapons and equipment in the corresponding faction’s codex book, and noting them down on a bit of paper. This is rather tedious and time consuming and can be quite daunting to newcomers of the tabletop game. It also prevents hobbyists from modifying their army easily. By using a simple GUI program with drop down menus and buttons that automatically accumulate points, it makes the process of army building much easier and less time consuming, as well as allowing a user to modify an existing army. This would also free the user from having to look up and remember values and constraints in the faction’s codex book.

As well as a consumer need for this program, there are also some personal reasons behind the project. Firstly it will be useful to me as a consumer, as I’m an avid Warhammer 40,000 fan who just so happens to collect the Space Marines faction, therefore I can use this program to help me build my own army rosters. I also want to be able to develop my object oriented programming skills in particular with Java, which is the language of choice for the project. The project also gives me an opportunity to implement a larger scale program compared to with what I’m accustomed too, which also gives a focus on the structure of the program rather than just the coding aspect. The planning and structure of a program is an important aspect of any large scale program, so this project should help me in my planning and design skills. It will also allow me to implement a design pattern, in particular that of the MVC (model, view, and controller) pattern, these design patterns can solve a great deal of many common programming problems in the object oriented paradigm, so exposure to them is a great asset to my personal development.

## Objectives of the Project

# Analysis of Requirements

## Program Functionality

### 2.1.1 Functional Requirements

The program is designed to be able to easily build an army roster for the Space Marines faction in Warhammer 40,000 7th edition. It will adhere to the rules and restrictions of the Warhammer 40,000 rules and will closely follow the corresponding faction’s codex book that lists all of the essential information of each unit within that faction. This information includes the point value of each member of a squad, or if the unit consists of a single model their point value, the maximum and minimum amount of members allowed if the unit is a squad, the available weapon upgrades for a unit or unit squad and their point values, also the amount of weapon upgrades a unit squad can have, as some squads are permitted to just one member being able to use a particular weapon (*Literature review, section 2.2, table 2.2 and section 2.1 paragraph 2, and section 2.2 paragraph 2*). Lastly a unit’s characteristics that can be found in the Warhammer 40,00 rulebook (Priestley, R. et al, 2004) *[12]* page 12. Some units are classed as vehicles and so have different unit characteristics (Priestley, R. et al, 2004) *[12]* page 58.

The entire basis of building an army roster in Warhammer 40,000 consists of utilising the point system. The idea is that when a user builds an army roster, they must impose a limit of how many resources an army can use. This ensures that two opponents on the tabletop have similar sized forces so the game is fair. This is determined with the point system, which is simply a number to represent the amount of resources that are available to construct an army roster with. The idea is, that as a user adds units, adds or removes unit members and what weapons upgrades a unit will have, this total point value accumulates until it’s as close to the set maximum points value as possible. (*Literature review, section 2.1 paragraph 1 and 2*) Point’s values are used for a number of different areas within a unit, which were described in the previous paragraph.

Now that the core functionality has been discussed utilising the point system, specific parts of the program can now be drawn out in more detail. Firstly the user should be able to name an army roster. This allows a user to add a personal touch to the army roster and doesn’t force the user to use a generic name imposed by the program. Secondly, a user must be able to specify the maximum amount of points for an army roster from a list of common values. A lower point army can be around 200, up to a larger point army of about 2500+. (*Literature review, section 2.1 paragraph 1).* Next, it may be useful to keep track of the date the user created the army roster. This could be important to see if the army roster is up to date with the current edition of the game as an out of date army roster may not comply with the current edition of the game, as new units may be introduced and rules may be changed.

Once the name, maximum points and the date are specified, the user can then add unit’s to the army roster. Adding units will accumulate points until it reaches the maximum points previously specified. However there needs to be some organisation of an army roster when adding units. Typically units in a faction in Warhammer 40,000 are categorised into different roles within an army (Literature review, *section 2.1, paragraph 4)*. As such these unit categories should be used within the program to ensure the army can be organised and divided when a user adds units. Once the unit category has been identified a user can then add the particular unit for that unit category into the army roster. However more considerations first need to be made before adding a unit, each of which were previously mentioned in the first paragraph. Upon choosing a unit category and a unit, the unit should have a minimum and maximum squad size and have options to choose a unit weapon upgrade per member of a squad. Finally the unit or unit squad should have a custom name, once again allowing the user to add a personal touch.

### 2.1.2 System Requirements

As well as the functional requirements that are directly involved with the tabletop game, there also needs to be some convenience features that allow the user to actually make the army roster useful. The most important of these features is to ensure the army roster can be saved to a readable text file, similar to BattleScribe (*Literature review*, *figure 2.3*) that saves to a html webpage, also the Warscroll builder (*Literature review*, *section 2.3 paragraph 2*), to a PDF. The text file should display the name of the army roster, the date of creation, total points and details of the unit’s custom name, a combined total of each squad members points value, the point’s value of individual members of a squad, a squad member’s weapons, and squad member number. Units consisting of a single model will have a very similar display, but with some of the squad specific display text omitted. Each unit or unit squad should also have its units characteristics (described in the first paragraph) displayed, and each unit should be organised into its unit category so the army roster display has a structure. Similar to BattleScribe, (*Literature review, section 2.3 paragraph 8* and figure 2.3*).*

A user should be able to remove units, both individually and also being able to clear an entire army roster. This should remove the unit from the model and army roster view. Much like BattleScribe (*Literature review, figure 2.3),* where clicking on a unit name allows the user to remove it, although BattleScribe has no option to clear multiple units at once, which is why it’s been identified as a requirement for this project. An army roster should also be able to be saved to a data file that can be loaded into the program again so a user can continue modifying units after the program has been closed and re-opened. Printing out the army roster display is also useful as sometimes a paper copy is required in certain settings such as a tournament (*Literature review, section 2.3 paragraph 9)*, where there might be no electronic devices available. Finally to represent the accumulation of points out of the set maximum, a counter should be displayed showing the current points value out of the set maximum. When a unit or squad member is added, the current points should increase, by the total units points cost, and when a unit is removed or unit member is removed the current points should be reduced by the amount of points cost the removed unit was. Just like how Warscroll does, by displaying the current points out of the set maximum points, and changing the current points based on if upgrades and units are added or removed (*Literature review, section 2.3 paragraph 4).* Lastlyclearing an entire army should reset the current points to zero.

## Interface Elements

The interface should be laid out in a simple and intuitive manner that is convenient for the user. In order to achieve this there are a number of elements the interface must have. Firstly the interface must have drop down menu’s that allow the user to choose items. Since this program is utilising JavaFX these are referred to as combo boxes (*Literature review, section 3.2 paragraph 2*). If a user wanted to choose a unit type in order to select a specific unit of that type, then two combo boxes should be used. The first one contains a list menu of all the unit types which was previously described in the program functionality section. Upon choosing a unit type, the second combo box should be populated with all the units of the previously selected unit type. So there are some cases where combo boxes are reliant upon on each other for population. Using combo boxes is convenient because it’s a common user interface element that also hides the content while it’s not focused which can help reduce clutter which may be overwhelming for some users.

The second element of the interface that will be widely used are buttons. Buttons are another common interface element that is used on all manner of software, and is also supported by the JavaFX framework (*Literature review, section 3.2 paragraph 2*). Buttons are simple as they require a simple click to perform an action. Often buttons are coupled with combo boxes to perform an action on the selected item in a combo box. Such as a user selecting a unit from a combo box and then creating that unit by pressing a button. Buttons will be used throughout the interface to perform different actions including, adding units, adding and removing unit members, upgrading unit’s weapons and equipment, removing a unit from the army roster and clearing an entire army roster.

Similar to buttons, menu’s and menu items are popular in software and are usually placed at the top of a programs interface. For example BattleScribe (*Literature review, section 2.3 paragraph 10*), has a top menu that has a number of actions, such as changing settings, saving and loading data and also providing information about the program. Having a separate menu bar at the top can be useful for these sorts of actions that are not directly linked to the main functionality of the program, because it separates the different areas of the interface. As such, a simple menu could be introduced into the interface of this program that gives options to save, load and print out a user’s army roster list. Information about the program, such as how to use it and the programs version can be placed in this menu.

There may be times where a user should have more control over a choice; this is where user input could be used in the form of text. Allowing a user to type into a text box gives the user more control than selecting a pre-determined item in a combo box menu. This can be particularly useful for giving things unique names. Having this option will be appreciated by the user as it can add customisability to the program, and gives an opportunity for the user to add their personal touch to the army roster.

Another important aspect is the ability to see any changes that may occur when creating and modifying an army roster. As such a real time representation of the army roster should be included within the interface. This could be an un-editable text box that shows a list of each individual unit within the army roster. When creating a unit, it is added to the text box, or when you add and remove members of a unit, a user can see the unit size grow and shrink, or see a certain member of a squads weapons change when upgrading that squad members weapon. This visual feedback is vital to the user, as it allows the user to make clearer decisions to change the army roster.

Text labels are also important on an interface. They can serve two purposes, first that they can give short instructions to a user. For example it can detail what to do when it comes to a combo box, such as telling the user to select a unit type. Otherwise a user may have no idea what the combo box is for. The second function is to provide visual feedback to the user. A text label can be changed upon an action being performed. For example if a unit has a maximum and minimum unit size, and the user adds too many or not enough members, then a text label can be used to inform the user that the unit’s maximum size is too large or minimum size is too small. A text label can also be used to update the total points of the current army against the set maximum, again giving the user visual feedback, this time regarding how close an army roster is to being complete. Finally a text label can be used to inform the user of any weapon upgrade violations, such as, if a user gives more weapon upgrades to a unit squad than is permitted, such as the example in section 2.2 paragraph 2 of the literature review.

The interface has a sense of progression where if a user skips ahead and chooses one option that is first dependent on another, then there may be unexpected consequences. For example if a user tries to add a unit by pressing a button without first choosing what unit type and unit to add to the army roster. To resolve this issue, certain portions of the interface such as buttons should be greyed out and not clickable until a user has selected the required choices in order to perform an action on a button. Although this does force the user to use the program in a uniform manner, it is necessary for certain parts of the program.

# Design Considerations

## 3.1 Major Design Decisions

### 3.1.1 Decisions regarding the structure of the code

The first design decision was in regard to structure of the code, since Java is predominantly an object oriented (OO) language (*Literature review, section 3.2),* the code should make use of common OO principles including inheritance, aggregation, making use of classes, objects and encapsulation.

Since a Warhammer 40,000 army consists of many units that have proprieties and rules, it would be good to represent these units as objects, as units represent the physical models of the tabletop game which are real life objects, which suits the OO paradigm *(Literature review section 3.2, paragraph 1).* Each unit object will have fields, including the unit name, weapon, second weapon, points value and the unit’s characteristics that were previously discussed in section 2.2.1. It also has two ArrayList data structures, namely unitWeaponList and unitSecondWeaponList. These fields are defined in the Unit super class (*appendix 1A*), and are actually initialised in an example Unit sub-class, (*appendix 1C*). However some units are considered to be vehicles and so have different unit characteristics. These vehicle unit characteristics are not defined in the Unit super class (apart from ballistic skill), but are rather defined and initialised in the corresponding Unit sub-classes that are vehicles, such as the Land Speeder Storm unit (*appendix 1D*). The purpose of these unit characteristics can be found on page 12, and page 58 for the vehicle unit characterises of Priestley, R. et al, (2004) *[12]* Warhammer 40,000 rulebook*.*

Each unit should also be initialised with a list of weapons that are taken from a pool of pre-defined weapons (that will be discussed later) that can be upgraded to. These weapons are initialised on all Unit sub-classes by adding items to the unitWeaponList ArrayList. An example of this occurring can be seen in appendix 1C for a Scout unit, note however that some units such as Scouts only have one weapon, so the weapon two field is ignored in unit sub-classes that only have one weapon. However for the units that have two weapons, the weapon two field is also initialised with a default weapon, and a second list of weapons is initialised by adding to the unitSecondWeaponList ArrayList, indicating what second weapons that unit can be upgraded to after creation. An example of a Unit sub-class with two default weapons, and two weapon lists can be seen in appendix 1D with the Land Speeder Storm unit.

One design problem that links to the size of the project is how many units there are in a Space Marine army. There are thirty nine different units, which mean that thirty nine classes need to be created with all similar methods and fields. In order to avoid writing out the same methods and defining the fields thirty nine times, inheritance can be used to write those methods and define the fields once in a unit super class (*appendix 1A*). Each individual unit is a sub-class that inherits from the unit super class, and can use the methods and fields from the unit super class, or if the unit sub-class has specific behaviour, then the unit sub-class can override any of the methods in the unit super class. Using inheritance in this manner can save on a lot of code from being repeated throughout the different units, it’s also more efficient and saves a lot of time. Appendix ???? shows a class diagram that clearly demonstrates the relationships between the various unit subclasses to the Unit super class.

Another design issue that needs to be considered is that most units in a Space Marine army are created with multiple members, often referred to as squads. Therefore when some units are created multiple instances of a unit’s class need to be created to form that squad. This is where the principle of aggregation can be used. For every unit class, another class can be created that is an aggregation of that unit. For example if you had a scout unit class, then a scout squad class would be an aggregation of many scouts, stored in an appropriate data structure. This aggregation class would be initialised with a user given squad name and have fields that specify the unit name and the maximum and minimum size of a squad, as specified in the Space Marine codex book (*appendix 1B*). The aggregation class would also contain methods. And like the unit classes, all thirty nine unit squad sub-classes will inherit from a super unit squad (aggregation) class where each unit squad sub-class will initialise all of the defined fields and use the methods in the super unit squad class (*appendix 1E*). Note the two overridden methods in this class, both of which come from the unit squad super class, but are overridden for each specific unit squad sub-class. All unit squad sub-classes will override the addUnitSquad method, as each unit squad will have to its own type of unit to add to the unit list to form the squad. However only certain unit squads with special weapon upgrade requirements override the squadWeaponsCheck method from the unit squad super class. Such is the case in appendix 1E where only one scout may upgrade to the weapons in the switch statement.

### 3.1.2 Decisions regarding the appearance of the user interface

The next phase of the design process was taking into consideration the layout of the interface. This design needs to be simple and intuitive for the user that follows a progression system. With the idea being that the user follows through the interface in a sequence, selecting various interface elements in order such as combo boxes and buttons. Typically users will follow left to right, so putting the first interface elements to the left side of the view will be intuitive to most users. An overall picture of the create army main component can be seen in appendix 3B which is made up of smaller sub sections, where the left most section, (illustrated by appendix 3C) allows the user to first select and add a unit, followed by a display that allows the user to observe and remove the recently added unit, (appendix 3E). Finally the right most side of this main component, where the user can modify an existing unit with weapon upgrades and the ability to add or remove squad members (appendix 3D). However each of the specific sub-sections just mentioned, adopt a top to bottom approach. For example combo boxes that require input from other combo boxes will be placed below one another so a user knows that the top most combo box needs to be selected first. So for every major component of the interface (referring to the create army section in this program) a left to right layout will be adopted, whereas smaller sub sections within the major interface will follow a top to bottom approach. This approach is similar to the BatteScribe program (*literature review, figure 2.3*). Another important element is the display of the current points out of the set maximum points. This display needs to be clear to the user so that the maximum points limit is never exceeded, if the current points exceeds the set maximum point limit, then the point display should change colour with a warning message, as well as prompting the user that the current points of an army should be reduced. The placement of this element should also be considered. Usually important aspects of a program are placed on the top, above all of the other major elements of the interface. The points display is another sub-section within the create army main section, and is placed at the top, illustrated by the top darker grey bar in appendix 3B, and a close-up in appendix 3F. Related to this also are the warning messages that occur when a user adds or removes to many or not enough unit members, or gives a certain unit squad more weapon upgrades than is permitted. These errors ensure that the unit the user adds and modifies are legal to the tabletop game. However adding these warning messages at the top along with the points display may appear rather cluttered, especially if the warning messages are long, so placing these unit specific warning messages at the bottom might be a good idea as then it doesn’t disrupt the left to right sequence of the other major elements of the interface, this warning message can be seen as the darker grey bar along the bottom in appendix 3B, and a close up in appendix 3G.

Another important design consideration is the separation of different sections of the interface, such as the create profile and create army roster. Using tabs to accomplish this would be best, because tabs allow for the different sections of the interface to be on a different window entirely, thus reducing clutter and making it clear to the user what each section does. In this case both the create profile and create army roster major sections have been separated into two tabs. Appendix 3A for the create profile and appendix 3B for the create army.

Lastly, the menu will allow the user to save, load and print, as well as provide information about the program. BatteScribe (*Literature review, figure 2.3*) split the menu’s actions into two categories, file for all of the actions around the data, and a help category for items that contain information about the program. This approach will also be taken in this project, not only because existing programs adopt a similar style that most users are accustomed to, but because it separates related items into categories, which is clearer and more organised. The two menu categories can be seen across the top of the program in appendices 3A and 3B, and the items within each of those categories can be seen in appendices 3H and 3I, also note that each menu item has a short cut key, which speeds up the process of those actions.

## 3.2 Armoury of Weapons

There are many types of weapon upgrades that are available to a unit, ranging from huge vehicle weapons to smaller person sized weapons. There is also the distinction between melee and ranged weapons. Some units only use melee weapons where as some use ranged only and some use a combination of both. Being able to organise weapons into different categories is important. The Space Marine codex classifies different weapons into the following categories: Ranged, Melee, Heavy and Special. These categories are used by multiple units in a Space Marine army. There are also some specific unit categories for Dreadnoughts and Terminators only and additional categories for other units not listed in the Space Marine codex which makes it easier to supply weapons for certain units and gives more organisation and clarity for certain weapons for certain units. Each of these weapon categories can be used in the program, by writing a class that contains multiple map data structures that represent each of the weapon categories. This can be seen in appendices 4A, 4B, 4C and 4D. A map data structure is suitable because map items can be accessed with a key rather than an index, in which case the weapons name can be used as the key, which avoids having to look up the index of the map, rather a weapon can be accessed by using its name and corresponding get method, (*appendix 4E*) which is easier and less time consuming.

However before creating a map of weapons, a weapon itself has to be defined in a class. A weapon will have a name and a points value, and will be defined in a weapon super class which will have a number of methods (*appendix 4F*). All of the weapon categories will be sub-classes of the weapon super class, once again making use of inheritance. However these sub-classes contain no methods of their own, instead they use the methods in the super class, the inheritance structure is merely for organisation purposes to split weapons into certain categories which can then be used to form a map data structure. The Heavy weapon type sub-class example can be seen in appendix 4G. It should be noted that the weapon super class can be made abstract because a weapon will never be initialised to type weapon, but rather one of the subclass weapon types. Also notice how each of the map data structures and their get methods are static, this means that a weapon can be accessed directly by calling the class, an example of this can be seen in appendix 1C, where weapons are being accessed and added to an ArrayList.

## 3.3 MVC Structure

Now that the requirements and design for the both the underlying code structure and the interface have been considered, putting them together needs to be thought out. This can be achieved using the model view controller (MVC) design pattern, where the interface represents the view, the army roster data represents the model and the controller acts as the intermediary between the model and the view. Using this approach is good because it follows the separation of concerns principal (*Literature review, section 3.1, paragraph 1*). The idea is, that the user will interact with the view through the interface and perform an action such as pressing a button, this will then transform the action to the controller, where it will actually carry out the actions of the user, by first updating the model (the army roster) which in turn will also update a portion of the view, namely the army roster display. This sequence can be observed in figure 3.1 of the literature review.

### 3.3.1 Model Structure

The model aspect is by far the largest entity within the program, and will consist of many classes (*appendix 5A)*, including the Army class that will store the army roster data. It will also have the two super classes previously mentioned, UnitSquad and all of its sub-classes and the Unit class and its sub-classes. Note that the sub-classes for both a Unit and UnitSquad are contained within the *model.UnitType.Unit* sub package structure within appendix 5A. There is also an additional class called UnitType that puts each unit within its correct unit type category. In particular this class is used in the interface to populate the two combo boxes found in appendix 3C, it first populates a combo box of unit types, then when a user selects a unit type it automatically populates a second combo box with all of the units of the previously selected unit types. The model also contains all of the weapon types and the weapon list (*appendix 5B*).

### 3.3.2 View Structure

The next aspect is the view using JavaFX. A typical JavaFX layout will place the entire view within a main root pane, which contains a series of sub-panes. This is referenced by the ArmyBuilderRootPane class. The sub-panes within the root pane are represented by the menu bar at the top, which corresponds to the ArmyBuilderMenuBar class, the unit creation tab section of the interface, which corresponds to the AddUnitPane class and finally the create profile tab of corresponding to the CreateProfilePane class (*Appendix 5C*). Notice how each major segment of the interface has its own class, the menu bar, and the two tabs, one for creating the profile details and the other for creating, modifying and viewing the army roster. These view elements are separated because each of them have a separate layout within the main root pane layout. For example the CreateProfilePane class uses JavaFX’s GridPane layout, which lays its contents out in a series of columns and rows (*Appendix 6C)*, which is suitable for a form. The AddUnitPane class uses JavaFX’s BorderPane layout which places each segment at the top, bottom, centre, left or right (*Appendix 6D and 6E)*. In fact this class also has its own sub sections that contain JavaFX’s HBox, VBox and GridPane layouts (*Appendix 6D)* which are then placed within the BorderPane of the AddUnitPane class (*Appendix 6E).* The ArmyBuilderMenuBar class uses JavaFX’s MenuBar layout (*Appendix 6B)* and finally the main root pane uses the BorderPane layout which puts the ArmyBuilderMenuBar class at the top and the two tabs (which contain the CreateProfilePane and AddUnitPane) in the centre (*Appendix 6A).*

### 3.3.3 Controller Structure

Lastly the controller is represented by the ArmyBuilderController class (*Appendix 5D)*, which contains all of the event handlers that are contained in the view, and within each handler a series of actions that calls from the view and model, to update the model, which also changes the army roster display. For example if a user wants to remove an existing member from an existing squad, then the controller detects that the user pressed the remove unit member button in the view (*appendix 3D)*, so then the remove unit member handler in the controller performs a series of actions in order to fulfil the users request (*Appendix 5E*). In this example, these actions are that first the selected unit squad is identified by using a get method from the view, and then assigned to a variable of type UnitSquad. This new variable then calls the remove unit method from the UnitSquad class and gives it the parameter of the selected squad’s last index as the unit member to remove from the squad. Then the unit size check method is called to see if the unit size is neither too big nor too small, if the squad size is within the correct range then any warning messages regarding the size of the squad will disappear. The model is then updated by calling a method from the Army class with the selected squad minus one, which will in turn also update the army roster display view, due to the binding in the Army class model. Finally the Army class model will call a method to update the points of the army roster to reflect the changes of the selected squad having one less member.

# Implementation

## 4.1 Unit and UnitSquad classes

There were a few realisations whilst developing the Unit classes. The first being that some Units have more than one weapon. This is especially prevalent on vehicle units, where there are multiple areas of a vehicle unit that can house weapons. As such there needed to be some way of catering for those affected units. Since all of the unit classes inherit from a super unit class , the change should be made there. The solution was rather simple, in that a new variable with the name weaponTwo was introduced, and in each of the unit sub-classes the units that required two weapons simply initialised the second weapon variable. The units that only had one weapons simply ignored this variable. However this created another problem in that the units that had two default weapons as well as two weapon upgrades meant that in the Unit super class, another ArrayList of available second weapon upgrades needed to be created. And like with the second default weapon, the second ArrayList, named unitSecondWeaponList was initialised by all of the unit sub-classes by taking weapons from the WeaponList class. The creation of this second default weapon variable and second weapon list meant that now units that had two default weapons and two weapon upgrades were fully catered for in the model. However that did leave one more unforeseen issue in regard to the display toString method in the Unit super class. That being that this method only contained the first weapons variable, meaning that when the unit details were displayed only one weapon was displayed, despite some units clearly having two. The solution for this was to simply override the toString method in the super class in each of the affected sub-class units. This change meant adding the second weapon variable to the overridden toString method in each of the affected units, which fixed the display of units with two weapons, however one more fix was required in the controller class to fully implement units that has two weapons. This will be explained in that section.

Another issue that arose while developing was that originally the user specified how large a squad would be upon creation, by choosing a number from a combo box that started from one to the squads maximum size. However upon further inspection in the Space Marine codex, the units not only had a maximum size but also a minimum size which meant that a user could choose a value below the minimum value didn’t comply with the tabletop rules. Most units were instead created with the minimum size as default, which for a lot of unit squads was five. To fix this issue, the choose unit size combo box was removed from the view, and a new variable called min was introduced into the UnitSquad super class. Then each unit sub-class called the set method from the sueper class to set the min size. Then the overridden addUnitSquad method used the min variable in a for loop to create the minimum unit size as default. This prevented the user from choosing the size of the unit upon creation and instead created the default size, just as the Space Marine codex book states for each unit.

## 4.2 Controller class

One issue in the controller class actually links back to the problem stated in the previous section regarding some units having two weapons. When the upgradeUnitWeaponHandler in the controller is called, both the weaponUpgrade and secondweaponUpgrade methods are called in order to upgrade a units weapons. However not all units have two weapons, therefore calling both weapon upgrade methods for units that only have one weapon causes an issue where the weapon doesn’t upgrade on both the model and view. Only the units with two weapons should call both methods. This was solved by using a switch statement in the upgradeUnitWeaponHandler, that contained all of the units names that had two weapons. This solved the issue, so that only units with two weapons called the second weapon upgrade method, and didn’t affect the units that only had one weapon, like it did previously.

The next issue in the controller was in the AddUnitHandler. The issue was that since there are thirty nine different units in the army roster, when adding a unit, each unit had to have its own reference variable that was initialised to a new squad of that unit. Each variable then called the addUnitSquad method to add the correct units to that unit squad, as well as adding the unit squad to the model. This meant that there were thirty nine reference variables and thirty nine calls to the two methods. This was a massive amount of code, and needed to be reduced. This was achieved by having a single reference variable of type UnitSquad, which every unit squad sub-type inherited. A switch statement was then used that contained the names of all the unit squad types, and for each case the UnitSquad variable was initialised to a new squad of the sub-type, and then the reference variable called the required methods just once. This meant that the only element that had thirty nine repetitions was the switch statement, rather than having thirty nine variables and method calls. Although there is still a fair amount of code in this handler, the amount has been drastically reduced, by benefiting from the existing inheritance structure.

## 4.3 Weapons and WeaponList classes

## 4.4 Testing

### 4.4.1 Test Plan

A black box approach will be taken for the testing, in that the underlying code is hidden. This approach is taken because it more closely resembles how a user would interact with the program. A user isn’t going to be concerned with the underlying code, as long as the program functions as required. The testing will cover a wide range of areas, including boundary testing for a unit’s size, the accumulation of points for each unit, the interface combo boxes, the interface buttons / menu items and finally any element within the interface that are disabled to restrict user input. Each of these testing segments will now be explained in more detail. It should also be noted that a green box indicates a pass, a red box indicates a fail and an amber box indicates a partial pass.

In order to test the unit’s size (*Appendix 7A*), the first thing to consider is if creating a unit or unit squad will actually create the default size, (usually the minimum) as specified in the datasheet section of the Space Marine codex book. So for all thirty nine units within the Space Marine army an expected value is inputted and then compared with the actual result default size of that unit upon the unit being created for the first time. If the expected and actual result matches then the test case is a pass. As well as testing for the default size of all thirty nine units, the minimum and maximum size should also be tested. To test this, a boundary input will be set for each unit. For the minimum size, the boundary value will be one below the minimum size, in order to trigger the message prompt at the bottom of the interface that warns the user that the unit or unit squad size falls short of the minimum squad size, and for the maximum size the boundary value is set to one above the maximum size, to trigger the message prompt that warns the user that the unit size has been exceeded. So in the min size boundary input column the minimum boundary input is specified by taking the minimum size of the squad and taking away one. The expected result is set to the min warning message and if that matches the actual output then the test passes. The same applies for the maximum size, where the boundary value is the units maximum size plus one, and the max warning message is set in the expected result column.

The next set of tests is to ensure that the points total of each squad can add and subtract points when certain actions are performed (*Appendix 7B*). The first action is when a unit or unit squad has members added or removed using the buttons on the interface. A mixture of adding and removing will be used for the different unit’s with the pattern being, adding then subtracting and so on down all the units. The first column will have a random number of members in a squad and each member’s point’s value will be totalled (indicated by the bold number in brackets) to indicate the starting unit’s total points value. Then extra members are added or removed, and the points value of that member will be added or subtracted to or from the original unit squads total points value. The expected value will be the original unit squads total points plus the added member / members points value or the original total points value subtracted by the removed member / members points value. If the expected point’s value matches the actual point’s value then the test passes. As well as adding and removing members, a unit squads total points can be influenced by the upgrade of weapons, as weapons also have points values. So the previous unit squad point’s total that was confirmed in the actual and expected value columns will be used as the new total unit squad point’s baseline. In the adding weapon upgrades column a squad member or multiple members are specified (by number) with their new weapon name and the points value of the weapon, or if the member has the option to upgrade two weapons, the second weapons name and points value. The new expected total squad point’s value (column six) will be the original tests squad points’ value (column four) plus the points of the weapon or weapons. If the column six expected result matches the actual value (column seven) then the test passes. For example for a Scout unit the initial size is six members totalled to sixty eight points (6 x 11) then three members are added at 11 points each. The expected value is then (66 + 11 + 11 + 11) to equal ninety nine points. Then that ninety nine points value is then used to add any weapon upgrade points to, for the final unit squads total point’s value. This test is a two step process, testing both the adding and removing of members and the addition of better weapons for unit squad members, as these are both key elements in the program that modify the unit squad’s point’s value. Note that some units do not have any weapon upgrades or the unit might not be able to add or remove members, or both of these. In this case the columns are marked with N/A (none applicable) such as the Land Raider Units, where these unit’s have a minimum and maximum unit size of one with no weapon upgrades available, in which case the tests are simplly ignored.

The next segment is regarding the contents of each combo box in the interface (*Appendix 7C*). This test is fairly simple, each test case will be for each combo box, and the expected value will be a short description of what should populate the combo box. If the expected result is the same as the actual result then the test passes. Another part of the interface that needs to be tested are the actions of the buttons and menu items (*Appendix 7D*). Just the same as with the combo box test, each button or menu item are specified for each test case and the expected result will be a short description of what the button or menu item does and then compared to the actual result to determine whether the test passed or failed. Lastly any item that is disabled on the interface will also be tested (*Appendix 7E*). Each test case will be an item that is disabled within the interface and the expected action will be a short description of how the disabled item can be enabled. This is matched with the actual action to determine the test result.

### 4.4.2 Test Results

# 5. Conclusions

## 5.1 Program Evaluation

## 5.2 Project Management

## 5.3 Tools Used

## 5.4 Final Thoughts

# Bibliography

[1] Docs.oracle.com. (n.d.) *JavaFX Overview (Release 8)*. [Online] Oracle. Available from: [https://docs.oracle.com/javase/8/javafx/get-started-tutorial/jfx-overview.htm#JFXST784](https://docs.oracle.com/javase/8/javafx/get-started-tutorial/jfx-overview.htm%23JFXST784%20)[Accessed 10 Nov. 2017].

[2] Hærum, E. (2015) *CBR-based Explanation-aware Army Builder for Warhammer Fantasy Battle*. Thesis (MSc), NTNU - Norwegian University of Science and Technology.

[3] Jailia, M., et al. (2016) Behaviour of MVC (Model View Controller) based Web Application developed in PHP and .NET framework. In: *International Conference on ICT in Business Industry & Government (ICTBIG)*, Indore, India, 18-19 Nov 2016. Rajasthan, India: IEEE, pp.1 - 5.

[4] Minor, J. and Gewali, L. (2004) Pedagogical issues in programming languages. In: *Proceedings of the International Conference on Information Technology: Coding and Computing (ITCC’04)*, Las Vegas, Nevada, USA, 5-7 April 2004. Las Vegas, Nevada: IEEE, pp.1 - 4.

[5] Reed, P. (2002) *A Comparison of Programming Languages for Graphical User Interface Programming*. Thesis (Undergraduate), University of Tennessee.

[6] Stocklein, J. et al. (2009) Poster: MVCE - A Design Pattern to Guide the Development of Next Generation User Interfaces. In: *IEEE Symposium on 3D User Interfaces*, Lafayette, Louisiana, USA, 14-15 March 2009. North Rhine-Westphalia, Germany: IEEE, pp.1 - 2.

[7] Syromiatnikov, A. and Weyns, D. (2014) A Journey Through the Land of Model-View-\* Design Patterns. In: *IEEE/IFIP Conference on Software Architecture*, Sydney, Australia, 7-11 April 2014. Vaxjo, Sweden: IEEE, pp.1 - 10.

[8] Warhammer Community (2017) *A Downloadable Warhammer 40,000 Army Roster - Warhammer Community*. [Online] Games Worksop. Available from: <https://www.warhammer-community.com/2017/06/30/a-downloadable-warhammer-40000-army-roster/> [Accessed 10 Nov. 2017].

[9] Warhammer Community (n.d.) *Warscroll Builder - Warhammer Community*. [Online] Games Workshop. Available from: [https://www.warhammer-community.com/warscroll-builder/](https://www.warhammer-community.com/warscroll-builder/%20) [Accessed 10 Nov. 2017].

[10] Zikic, N. (2016) *Explanation-Aware Army Builder for Warhammer 40k*. Thesis (MSc), NTNU - Norwegian University of Science and Technology.

[11] Codex: Space Marines. (2015)*Codex: Space Marines* 7th ed. Lenton, Nottingham: Games Workshop.

[12] Priestley, R. et al (2004) *Warhammer 40,000*. 4th ed. Glen Burnie, Maryland: Games Workshop.

[13] BattleScribe (n.d.) *What is BattleScribe?* [Online] BattleScribe. Available from: [https://www.battlescribe.net/?tab=features](https://www.battlescribe.net/?tab=features%20)[Accessed 13 Nov. 2017].

[14] Taylor, J. (2010) *BattleScribe*. [Software] Version 2.01.05.93.Unknown: BattleScribe.

[15] Bishop. E, et al.(n.d.) *Warhammer 40,000: 7th Editionv2040.* [Online Data File] BattleScribe. Available from: [https://github.com/BSData/wh40k-7th-edition](https://github.com/BSData/wh40k-7th-edition%20)[Accessed 13 Nov. 2017].

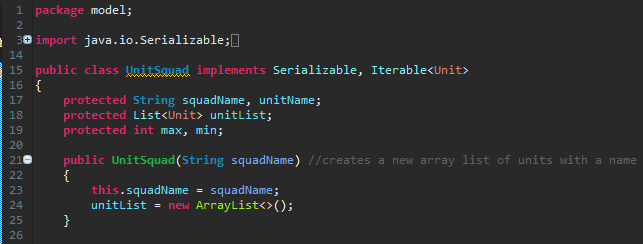
[16] Curry, E and Grace, P. (2008) Flexible Self-Management Using the Model-View-Controller Pattern. *IEEE Software*, 25(3), pp.88-89.

# Appendices

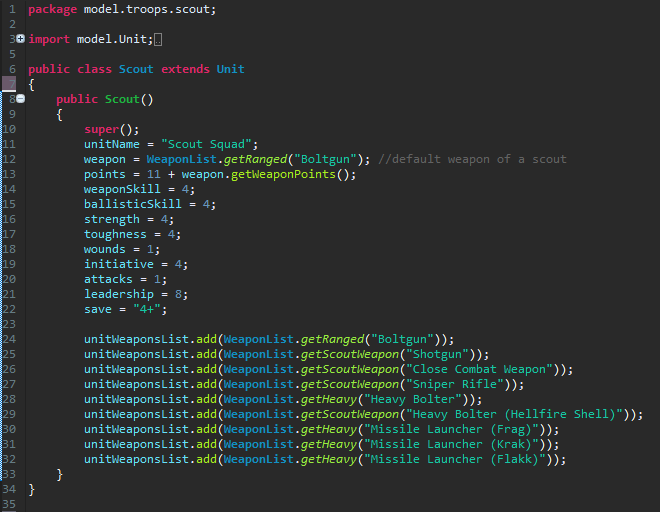
## Appendix 1A – Fields and constructor of the Unit super class

## fig(unit super class).PNG

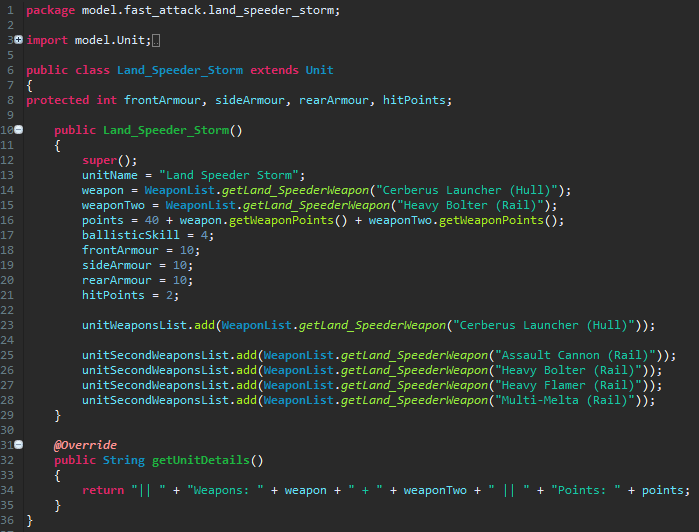
## Appendix 1B - Fields and constructor of the UnitSquad super class



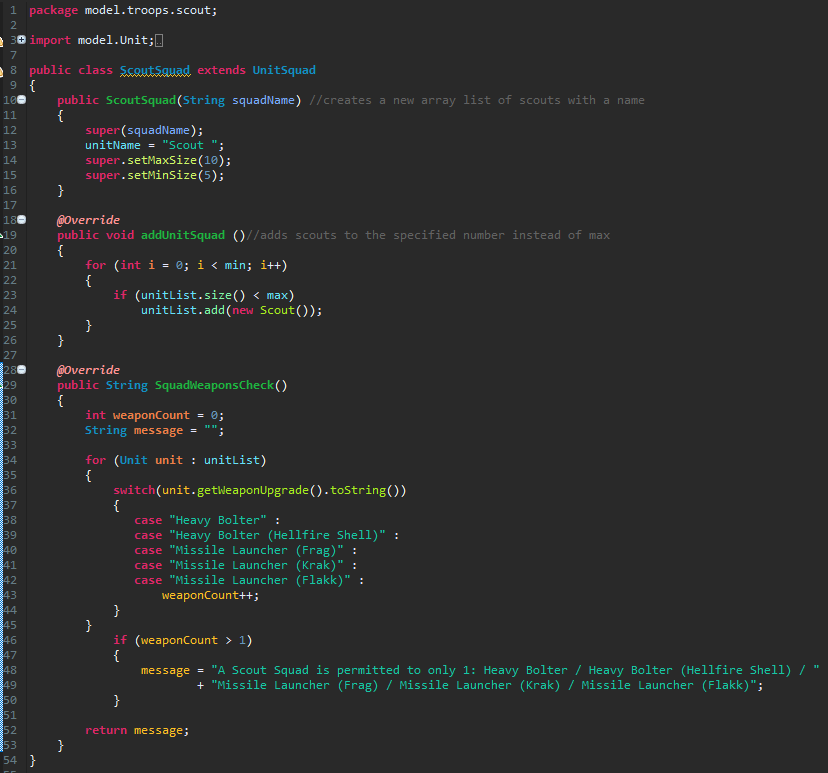
## Appendix 1C – Example of a Unit sub-class with one weapon



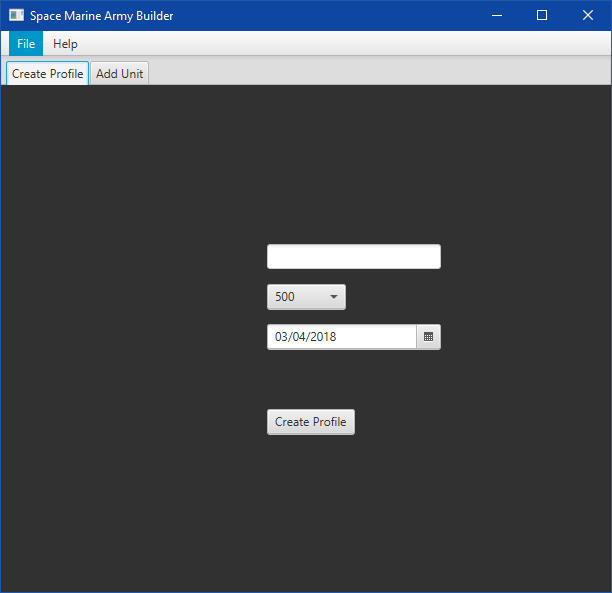
## Appendix 1D – Example of a Unit sub-class with two weapons



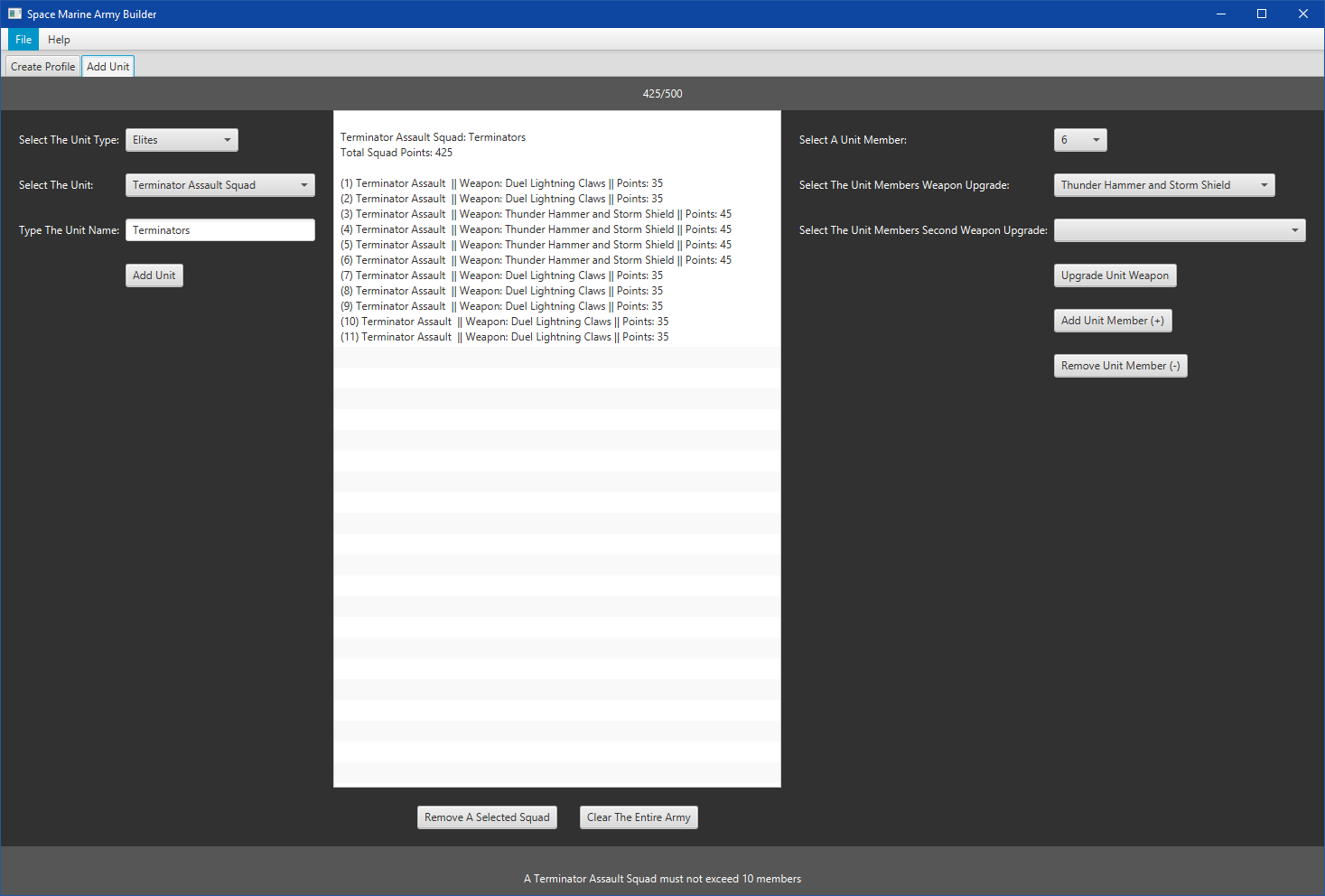
## Appendix 1E – Example of a UnitSquad sub-class



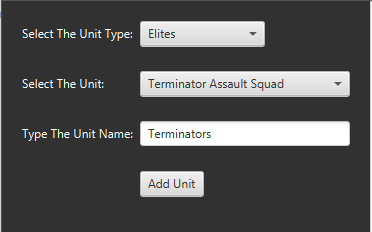
## Appendix 3A – User interface create profile screen



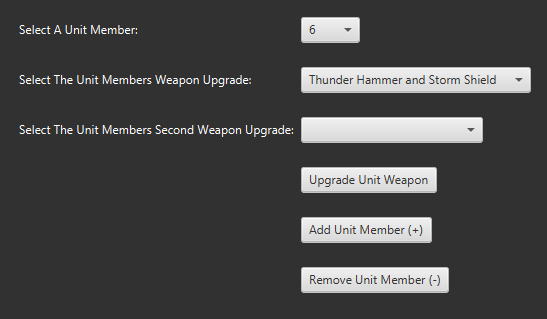
## Appendix 3B – User interface create army roster screen



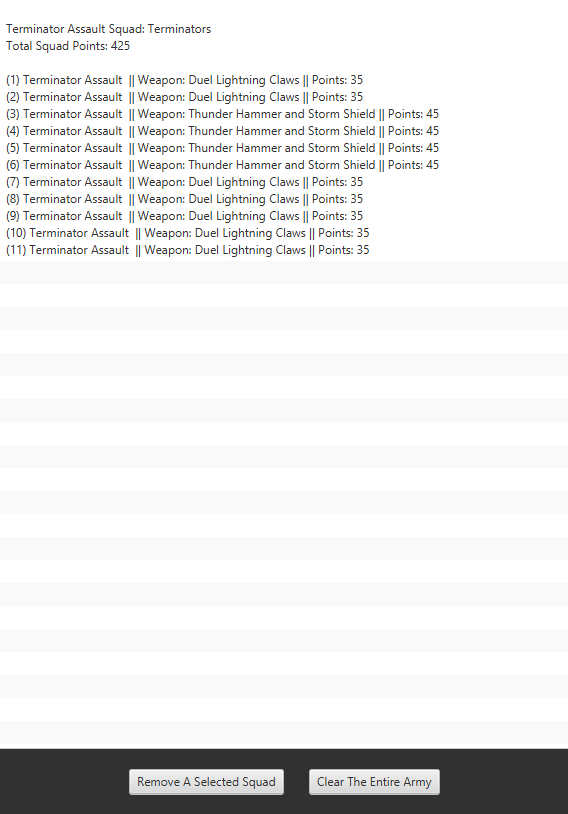
## Appendix 3C – User interface add unit section



## Appendix 3D – User interface modify unit section



## Appendix 3E – User interface display army roster section



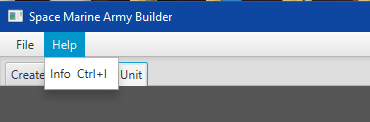
## Appendix 3F – User interface display points section

3F(points display top section).png

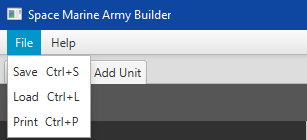
## Appendix 3G – User interface warning message section

3G(warning messages bottom section).png

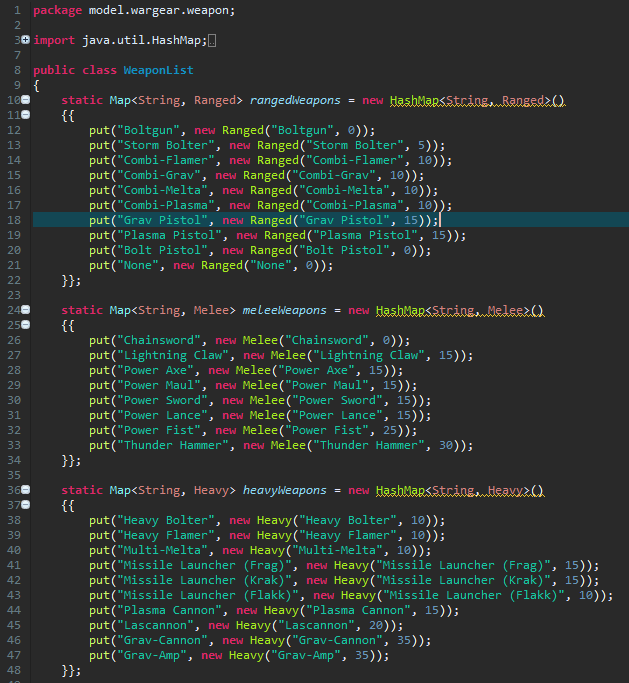
## Appendix 3H – User interface help menu section



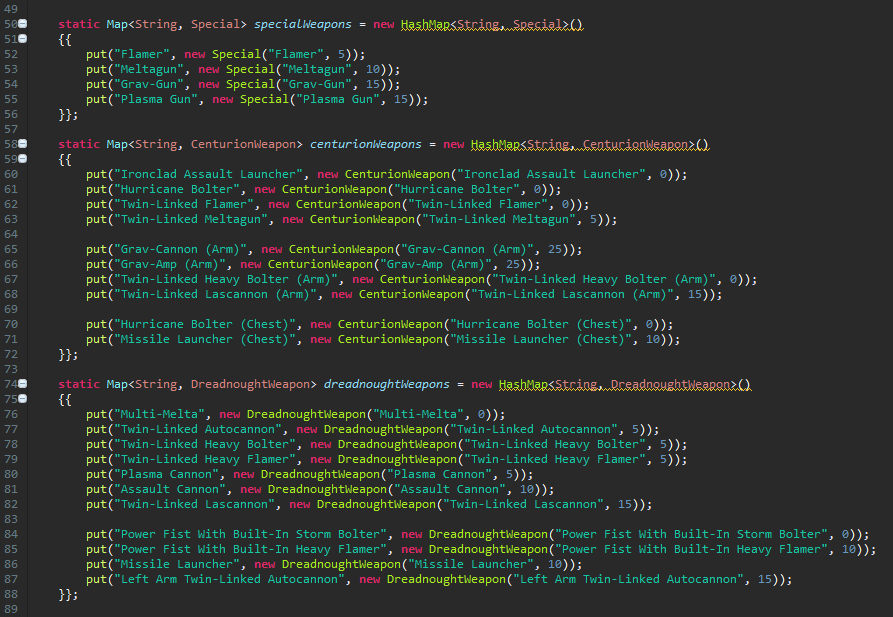
## Appendix 3I – User interface file menu section



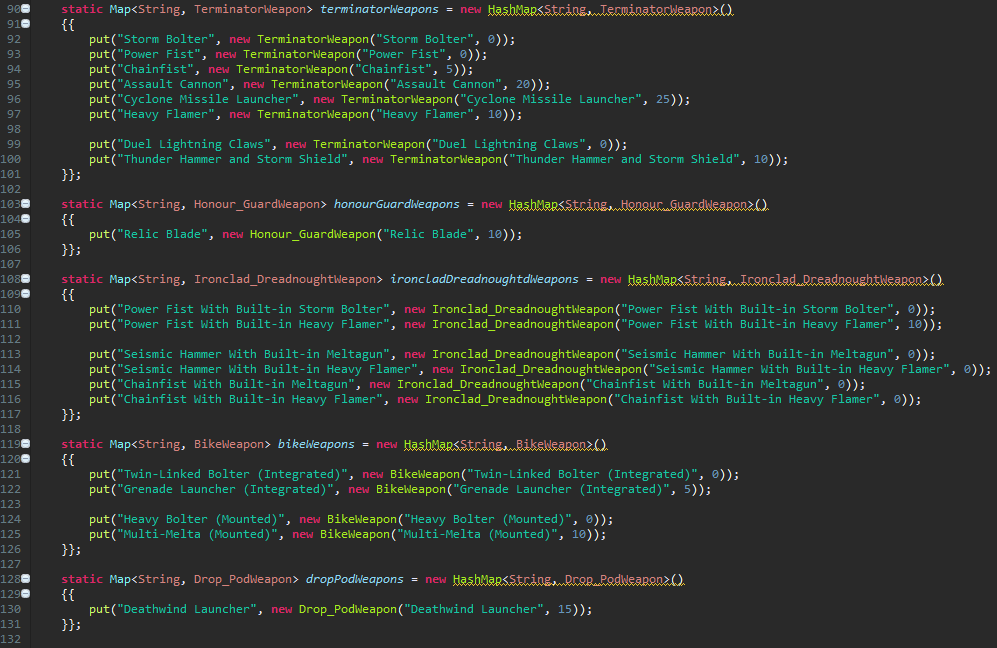
## Appendix 4A – Weapon list of map data structures



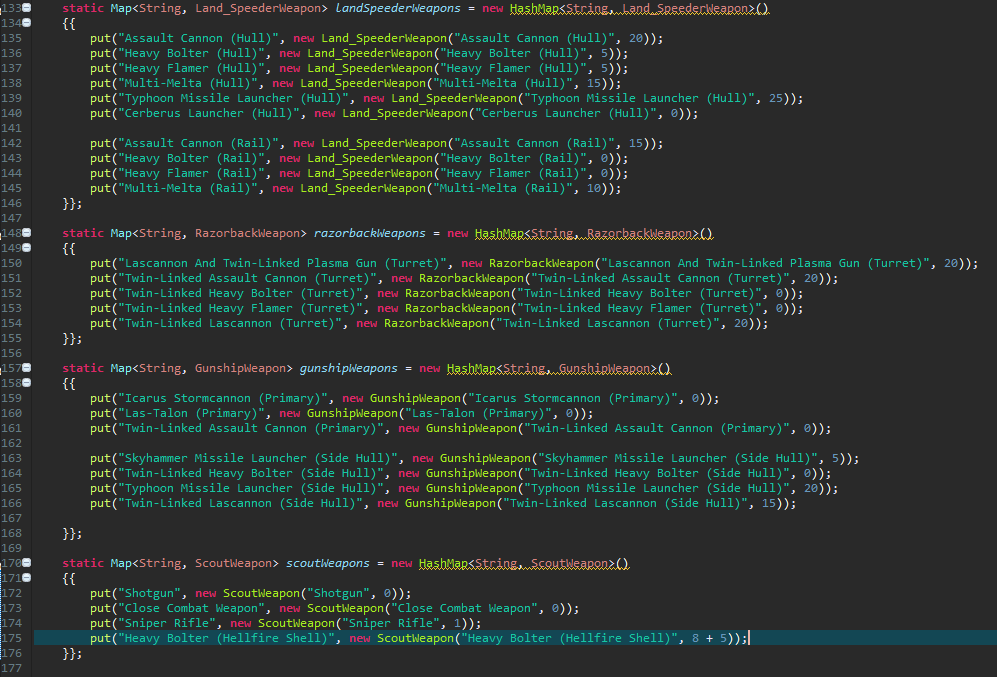
## Appendix 4B – Weapon list of map data structures continued...



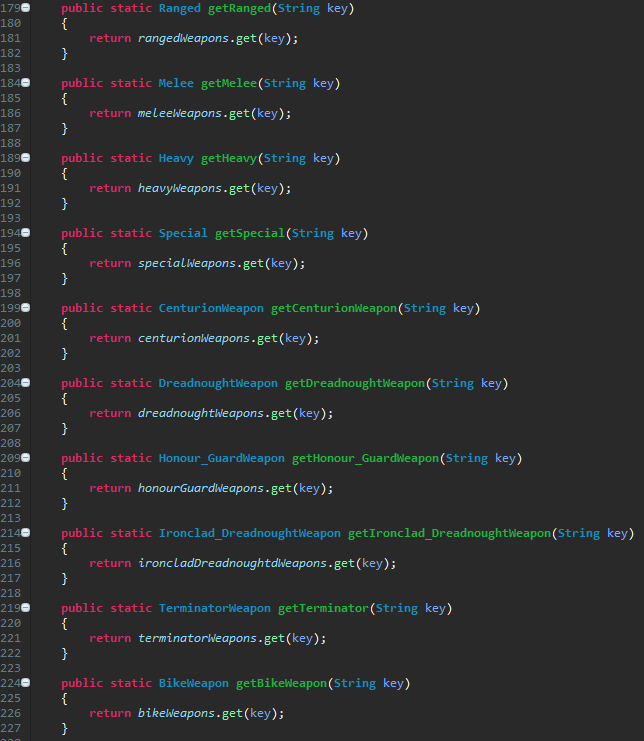
## Appendix 4C – Weapon list of map data structures continued...



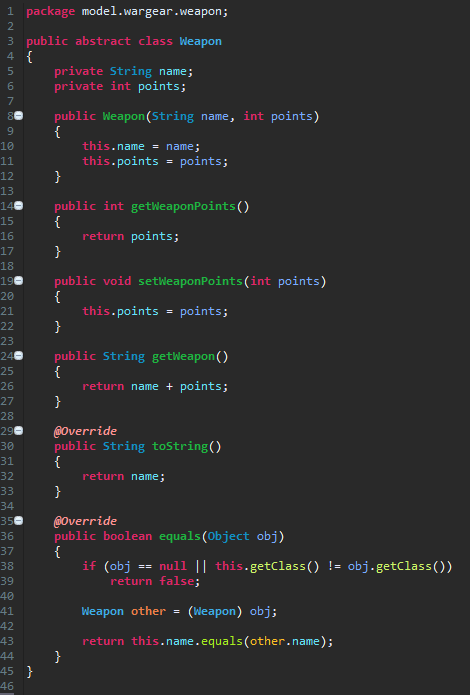
## Appendix 4D – Weapon list of map data structures continued...



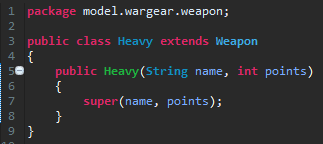
## Appendix 4E – Weapon list get methods



## Appendix 4F – Weapon super class

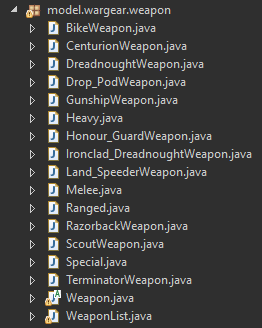


## Appendix 4G – Heavy weapon sub-class

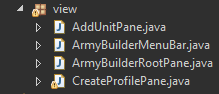


## fig(model package structure).pngAppendix 5A – eclipse model package structure

## fig(model package structure).pngAppendix 5B – eclipse weapon model package structure



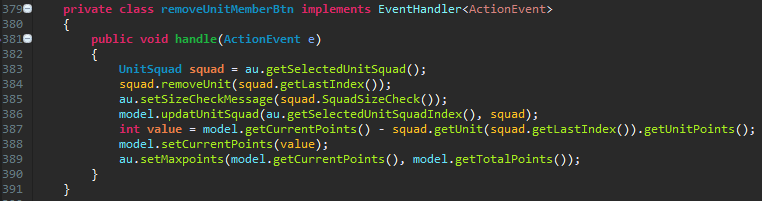
## fig(model package structure).pngAppendix 5C – eclipse view package structure



## fig(model package structure).pngAppendix 5D – eclipse controller package structure

5E(controller package structure).png

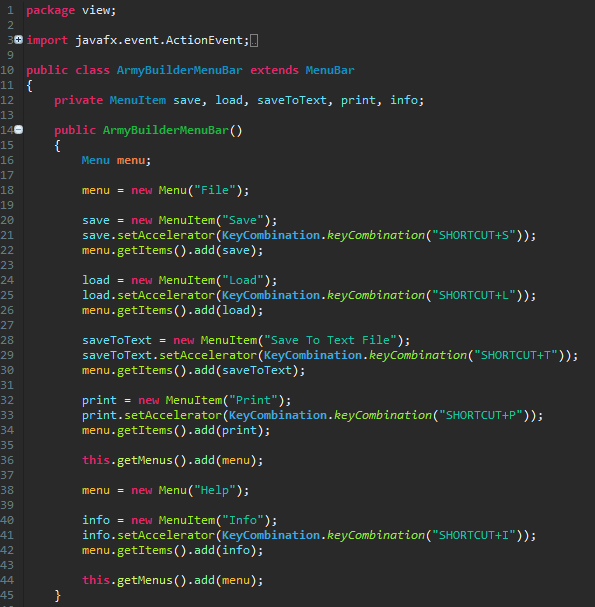
## Appendix 5E – Controller, remove unit member handler



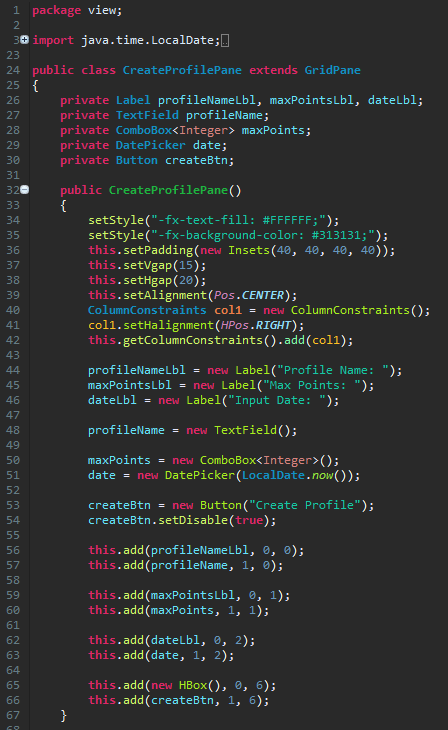
## Appendix 6A – ArmyBuilderRootPane class



## Appendix 6B – ArmyBuilderMenuBar class



## Appendix 6C – CreateProfilePane class



## 6D(AddUnitPane 1).pngAppendix 6D – AddUnitPane class sub-panes

## 6D(AddUnitPane 1).pngAppendix 6E – AddUnitPane class sub-panes populating BorderPane



## Appendix 7A – Unit Size Boundary Test Cases

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Test Case (Unit) | Default size expected output | Default size actual output | Min size outside boundary input | Expected output | Actual output | Max size outside boundary input | Expected output | Actual output |
| Centurion Assault | 3 members | 3 members | 2 members | Min size warning message | Min size warning message | 7 members | Max size warning message | Max size warning message |
| Command | 5 members | 5 members | 4 members | Min size warning message | Min size warning message | 6 members | Max size warning message | Max size warning message |
| Dreadnought | 1 member | 1 member | 0 members | Min size warning message | Out of bounds exception | 4 members | Max size warning message | Max size warning message |
| Honour Guard | 3 members | 3 members | 2 members | Min size warning message | Min size warning message | 11 members | Max size warning message | Max size warning message |
| Ironclad Dreadnought | 1 member | 1 member | 0 members | Min size warning message | Out of bounds exception | 4 members | Max size warning message | Max size warning message |
| Sternguard Veteran | 5 members | 5 members | 4 members | Min size warning message | Min size warning message | 11 members | Max size warning message | Max size warning message |
| Terminator | 5 members | 5 members | 4 members | Min size warning message | Min size warning message | 11 members | Max size warning message | Max size warning message |
| Terminator Assault | 5 members | 5 members | 4 members | Min size warning message | Min size warning message | 11 members | Max size warning message | Max size warning message |
| Vanguard Veteran |  |  |  | Min size warning message | Min size warning message |  | Max size warning message | Max size warning message |
| Venerable Dreadnought | 1 member | 1 member | 0 members | Min size warning message | Out of bounds exception | 4 members | Max size warning message | Max size warning message |
| Assault |  |  |  | Min size warning message | Min size warning message |  | Max size warning message | Max size warning message |
| Attack Bike | 1 member | 1 member | 0 members | Min size warning message | Out of bounds exception | 4 members | Max size warning message | Max size warning message |
| Bike | 3 members | 3 members | 2 members | Min size warning message | Min size warning message | 9 members | Max size warning message | Max size warning message |
| Drop Pod | 1 member | 1 member | 0 member | Min size warning message | No warning message | 2 members | Max size warning message | Max size warning message |
| Land Speeder | 1 member | 1 member | 0 members | Min size warning message | Out of bounds exception | 4 members | Max size warning message | Max size warning message |
| Land Speeder Storm | 1 member | 1 member | 0 members | Min size warning message | No warning message | 2 members | Max size warning message | Max size warning message |
| Razorback | 1 member | 1 member | 0 members | Min size warning message | No warning message | 2 members | Max size warning message | Max size warning message |
| Rhino | 1 member | 1 member | 0 members | Min size warning message | No warning message | 2 members | Max size warning message | Max size warning message |
| Scout Bike | 3 members | 3 members | 2 members | Min size warning message | Min size warning message | 11 members | Max size warning message | Max size warning message |
| Stormhawk Interceptor | 1 member | 1 member | 0 members | Min size warning message | No warning message | 2 members | Max size warning message | Max size warning message |
| Stormtalon Gunship | 1 member | 1 member | 0 members | Min size warning message | No warning message | 2 members | Max size warning message | Max size warning message |
| Centurion Devastator | 3 members | 3 members | 2 members | Min size warning message | Min size warning message | 7 members | Max size warning message | Max size warning message |
| Devastator | 5 members | 5 members | 4 members | Min size warning message | Min size warning message | 11 members | Max size warning message | Max size warning message |
| Hunter | 1 member | 1 member | 0 members | Min size warning message | Out of bounds exception | 4 members | Max size warning message | Max size warning message |
| Land Raider | 1 member | 1 member | 0 members | Min size warning message | Out of bounds exception | 2 members | Max size warning message | Max size warning message |
| Land Raider Crusader | 1 member | 1 member | 0 members | Min size warning message | Min size warning message | 2 members | Max size warning message | Max size warning message |
| Land Raider Redeemer | 1 member | 1 member | 0 members | Min size warning message | Min size warning message | 2 members | Max size warning message | Max size warning message |
| Predator | 1 member | 1 member | 0 members | Min size warning message | Out of bounds exception | 4 members | Max size warning message | Max size warning message |
| Stalker | 1 member | 1 member | 0 members | Min size warning message | Out of bounds exception | 4 members | Max size warning message | Max size warning message |
| Stormraven Gunship | 1 member | 1 member | 0 members | Min size warning message | Out of bounds exception | 2 members | Max size warning message | Max size warning message |
| Thunderfire Cannon | 1 member | 1 member | 0 members | Min size warning message | Out of bounds exception | 4 members | Max size warning message | Max size warning message |
| Vindicator | 1 member | 1 member | 0 members | Min size warning message | Out of bounds exception | 4 members | Max size warning message | Max size warning message |
| Whirlwind | 1 member | 1 member | 0 members | Min size warning message | Out of bounds exception | 4 members | Max size warning message | Max size warning message |
| Captain |  |  |  | Min size warning message | Min size warning message |  | Max size warning message | Max size warning message |
| Chaplain |  |  |  | Min size warning message | Min size warning message |  | Max size warning message | Max size warning message |
| Librarian |  |  |  | Min size warning message | Min size warning message |  | Max size warning message | Max size warning message |
| Techmarine |  |  |  | Min size warning message | Min size warning message |  | Max size warning message | Max size warning message |
| Scout | 5 members | 5 members | 4 members | Min size warning message | Min size warning message | 11 members | Max size warning message | Max size warning message |
| Tactical | 5 members | 5 members | 4 members | Min size warning message | Min size warning message | 11 members | Max size warning message | Max size warning message |

## Appendix 7B – Unit total points accumulation

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Test Case (Unit) | Adding or removing unit members | Expected result | Actual result | Adding weapon upgrades | Expected result | Actual result |
| Centurion Assault | 3 members (**165pts**) + 1 member (55pts) each | 220 points | 220 points | Member 3 with Twin-Linked Meltagun (5pts) + Hurricane Bolter (0pts) | 225 points | 225 points |
| Command | 5 members (**90pts**) – 1 member (18pts) each | 72 points | 72 points | Member 2 with Storm Bolter (5pts) | 77 points | 77 points |
| Dreadnought | 1 member (**100pts**) + 1 member (100pts) each | 200 points | 200 points | Member 1 with Assault cannon (10pts) + Left Arm Twin-Linked Autocannon (15pts) | 225 points | 225 points |
| Honour Guard | 7 members (**175pts**) – 3 members (25pts) each | 100 points | 100 points | Member 4 with Power Maul (0pts) | 100 points | 100 points |
| Ironclad Dreadnought | 1 member (**135pts**) + 2 members (135pts) each | 405 points | 405 points | Member 3 with Power Fist With Built-in Heavy Flamer (10pts) + Chainfist With Built-in Meltagun (0pts) | 415 points | 425 points |
| Sternguard Veteran | 5 members (**110pts**) – 3 members (22pts) each | 44 points | 44 points | Member 1 with Plasma Gun (15pts) | 59 points | 59 points |
| Terminator | 5 members (**175pts**) + 5 members (35pts) each | 350 points | 350 points | Member 5 with Chainfist (5pts) and member 6 with Chainfist (5pts) + Assault Cannon (20pts) | 380 points | 380 points |
| Terminator Assault | 8 members (**280pts**) - 2 members (35pts) each | 210 points | 210 points | Member 2 with Thunder Hammer and Storm Shield (10pts) | 220 points | 220 points |
| Vanguard Veteran |  |  |  |  |  |  |
| Venerable Dreadnought | 1 member (**125pts**) + 1 member (125pts) each | 250 points | 250 points | Member 1 with Plasma Cannon (5pts) + Power Fist With Built-in Heavy Flamer (10pts) | 265 points | 265 points |
| Assault |  |  |  |  |  |  |
| Attack Bike | 1 member (**40pts**) + 2 members (40pts) each | 120 points | 120 points | Member 1 and member 2 with Multi-Melta (10pts) each | 140 points | 140 points |
| Bike | 7 members (**147pts**) – 3 members (21pts) each | 84 points | 84 points | Member 4 with Grav-Gun (15pts) | 99 points | 99 points |
| Drop Pod | 1 member (**35pts**) | N/A | N/A | Member 1 with Deathwind launcher (15pts) | 50 points | 50 points |
| Land Speeder | 2 members (**90pts**) + 1 member (45pts) each | 135 points | 135 points | Member 3 with Typhoon Missile Launcher (25pts) + Heavy Flamer (0pts) | 160 points | 160 points |
| Land Speeder Storm | 1 member (**40pts**) | N/A | N/A | Member 1 with Assault Cannon (15pts) | 55 points | 55 points |
| Razorback | 1 member (**55pts)** | N/A | N/A | Member 1 with Twin-Linked Lascannon (20pts) | 75 points | 75 points |
| Rhino | 1 member (**35pts**) | N/A | N/A | N/A | N/A | N/A |
| Scout Bike | 9 members (**162pts**) – 5 members (18pts) each | 72 points | 72 points | Member 2 and 3 with Integrated Grenade Launcher (5pts) each | 82 points | 82 points |
| Stormhawk Interceptor | 1 member (**125pts**) | N/A | N/A | Member 1 with Icarus Stormcannon (0pts) + Skyhammer Missile Launcher (5pts) | 130 points | 130 points |
| Stormtalon Gunship | 1 member (**110pts**) | N/A | N/A | Member 1 with Typhoon Missile Launcher (20pts) | 130 points | 130 points |
| Centurion Devastator | 3 members (**165pts**) + 3 members (55pts) each | 330 points | 330 points | Member 2 with Grav-Amp (25pts) + Missile Launcher (10pts) and member 5 with Twin-Linked Lascannon (15pts) + Missile Launcher (10pts) | 390 points | 390 points |
| Devastator | 8 members (**112pts**) – 3 members (14pts) each | 70 points | 70 points | Member 4 with a Lascannon (20pts) and member 5 with a Heavy Bolter (10pts) | 100 points | 100 points |
| Hunter | 2 members (**140pts**) + 1 member (70pts) | 210 points | 210 points | N/A | N/A | N/A |
| Land Raider | 1 member (**250pts**) | N/A | N/A | N/A | N/A | N/A |
| Land Raider Crusader | 1 member (**250pts**) | N/A | N/A | N/A | N/A | N/A |
| Land Raider Redeemer | 1 member (**250pts)** | N/A | N/A | N/A | N/A | N/A |
| Predator | 3 members (**225pts**) – 2 members (75pts) each | 75 points | 75 points | Member 1 with Twin-Linked Lascannon (25pts) + Heavy Bolter (20pts) | 120 points | 120 points |
| Stalker | 1 member (**75pts**) + 2 members (75pts) each | 225 points | 225 points | N/A | N/A | N/A |
| Stormraven Gunship | 1 member (**200pts**) | N/A | N/A | Member 1 with Twin-Linked Plasma Cannon (0pts) + Hurricane Bolter (30pts) | 230 points | 230 points |
| Thunderfire Cannon | 3 members (**300pts**) – 1 member (100pts) | 200 points | 200 points | N/A | N/A | N/A |
| Vindicator | 2 members (**240pts**) + 1 member (120pts) | 360 points | 360 points | N/A | N/A | N/A |
| Whirlwind | 3 members (**195pts**) – 2 members (65pts) each | 65 points | 65 points | N/A | N/A | N/A |
| Captain |  |  |  |  |  |  |
| Chaplain |  |  |  |  |  |  |
| Librarian |  |  |  |  |  |  |
| Techmarine |  |  |  |  |  |  |
| Scout | 6 members (**66pts**) + 3 members (11pts) each | 99 points | 99 points | Members 1 , 2 and 3 with Sniper Rifle (1pt) each and member 4 with Heavy Bolter (8pts) | 110 points | 112 points |
| Tactical | 10 members (**140pts**) – 5 members (14pts) | 70 points | 70 points | Member 4 with Missile Launcher (Krak) (15pts) and member 5 with Meltagun (10pts) | 95 points | 95 points |

## Appendix 7C – Interface functionality (Combo box)

|  |  |  |
| --- | --- | --- |
| Test Case | Expected contents | Actual contents |
| Max points Combo box | List of numbers incrementing by 250 up to 5000 | List of numbers incrementing by 250 up to 5000 |
| Unit Type Combo box | List of unit types including; HQ, Troops, Elites, Fast Attack and Heavy Support | List of unit types including; HQ, Troops, Elites, Fast Attack and Heavy Support |
| Unit Combo box | List of units based on the unit type selected | List of units based on the unit type selected |
| Unit Member Combo box | List of numbers of the currently selected units size | Always lists 1 – 10 for all units no matter the size |
| Weapon Upgrade Combo box | List of weapon upgrades for the selected unit that can upgrade | List of weapons, when a unit is first created but doesn’t change if an existing unit is selected |
| Second Weapon Upgrade Combo box | List of weapon upgrades for the selected unit that can upgrade two weapons | List of weapons, when a unit is first created but doesn’t change if an existing unit is selected |

## Appendix 7D – Interface functionality (Buttons / Menu Items)

|  |  |  |
| --- | --- | --- |
| Test Case | Expected action | Actual action |
| Create Profile Button | Goes over to the next tab and displays the selected maximum points | Goes over to the next tab and displays the selected maximum points |
| Add Unit Button | Adds the selected unit into the display, with the custom name, unit characteristics, total squad points, and each member’s weapon and point value. Also updates the current points in the points display. | Adds the selected unit into the display, with the custom name, unit characteristics, total squad points, and each member’s weapon and point value. Also updates the current points in the points display. |
| Upgrade Unit Weapon Button | Updates a unit member’s weapon name and point’s value, as well as the total squad point’s value and current point’s value in the points display. | Updates a unit member’s weapon name and point’s value, as well as the total squad point’s value and current point’s value in the points display |
| Add Unit Member Button | Adds a new member of the selected unit, and displays its member number, weapon and points value. The total squad point’s value is updated and the current points are also updated. | Adds a new member of the selected unit, and displays its member number, weapon and points value. The total squad point’s value is updated and the current points are also updated. |
| Remove Unit Member Button | Removes the last member of the selected unit, and removes its member number, weapon and points value. The total squad point’s value is updated and the current points are also updated. | Removes the last member of the selected unit, and removes its member number, weapon and points value. The total squad point’s value is updated and the current points are also updated. |
| Remove Squad Button | Removes all details from the display of the selected squad and deducts the current points by the cost of the removed squad. | Removes all details from the display of the selected squad and deducts the current points by the cost of the removed squad. |
| Clear Army Button | Removes all details of all units in the display and resets the current points to 0 in the points display. | Removes all details of all units in the display and resets the current points to 0 in the points display . |
| Help – Info Item | Display a text prompt that details what the program is about and how to use it. | Display a text prompt that details what the program is about and how to use it. |
| File – Save Item | Save the current army roster into a data file. The data file should be the name of the army roster. | An error messages occurs when trying to save |
| File – Load Item | Load the data file containing the army roster information into the display. | Can’t load the data file as saving doesn’t work. |
| File – Save to text file Item | An external text file that displays the army roster name, the date of creation, the total point count of the army all of the units in the army categorised by their unit type. The text file should also have the name of army roster. | An external text file that displays the army roster name, the date of creation, the total point count of the army all of the units in the army categorised by their unit type. The text file should also have the name of army roster. |
| File – Print Item |  |  |

## Appendix 7E – Interface functionality (Disabled Items)

|  |  |  |
| --- | --- | --- |
| Test Case | Expected action | Actual action |
| Create Profile Button | Disabled until a user provides an army roster name in the first text field. | Disabled until a user provides an army roster name in the first text field. |
| Unit Combo Box | Disabled until an item from the Unit Type Combo Box has been selected. | Disabled until an item from the Unit Type Combo Box has been selected. |
| Add Unit Button | Disabled until both the Unit Type and Unit Combo Boxes have been selected. It also requires input from the unit name in the text field. | Disabled until both the Unit Type and Unit Combo Boxes have been selected. It also requires input from the unit name in the text field. |
| Weapon / Second Weapon Upgrade Combo Box and Upgrade Unit Weapon Button | Disabled until a unit member is selected from the Unit Member Combo box | Disabled until a unit member is selected from the Unit Member Combo box. |
| File – Save Item | Disabled if the current points of an army roster exceeds the set maximum points value | Disabled if the current points of an army roster exceeds the set maximum points value |
| File – Save to text file Item | Disabled if the current points of an army roster exceeds the set maximum points value | Disabled if the current points of an army roster exceeds the set maximum points value |
| File – Print Item | Disabled if the current points of an army roster exceeds the set maximum points value |  |

## Appendix 8A – Literature Review

**A literature review on the need for a GUI desktop program to aid in the building of an army roster list in Warhammer 40,000**

**Grant Rigby-P15223403**

**Computer Science BSc (Hons)**

**Abstract**

For a long time players of the tabletop war-game Warhammer 40,000, have been creating army list rosters in a rather tedious and time consuming manner. In that they have to manually look up the associated point values of both the models and their equipment in the chosen faction’s codex book. These point values are then totalled to ensure they don’t surpass a chosen point limit. Traditionally this process is achieved by using a calculator and a pen and paper, which can be difficult due to the other constraints of particular models. In order to alleviate this problem it has been decided to create a desktop GUI program that does the point calculation automatically, as well as ensuring all the constraints of that faction are met.

# Introduction

This report will consider the need for a desktop application GUI (graphical user interface) program in order to aid beginner players of the miniature tabletop wargame, Warhammer 40,000 to create an army roster list. The Space Marines faction has been chosen as the basis for the programs implementation because this faction is a good choice for beginners and is a very popular faction in Warhammer 40,000.

This report will have three sections. The first section will discuss the background theory behind some of the existing literature behind this topic area. This will include how building an army in Warhammer 40,000 works, utilising the point system. As well as understanding some of the constraints and rules that need to be applied when building a Space Marines faction army. There will also be some background on two existing table top army builders. With one being a website and the other being a fully-fledged downloadable desktop application.

The second section of the report will focus on the theory behind implementing a GUI program. Firstly a common design pattern known as MVC (Model View Controller) will be evaluated in regard to GUI architecture design. There will also be some discussion on the JavaFX framework *Docs.oracle.com. (n.d.) [1], which* is a framework, used for implementing GUI applications on the Java platform.

The final section will be a conclusion of the above literature and will discuss the choices that have been made in regard to the implementation of the program.

# Background theory

## 2.1 Warhammer 40,000 army building

Constructing an army list roster for Warhammer 40,000 can be a complex and tedious process. There are a number of constraints and restrictions that need to be met in order to construct an army list. These include, limiting the size of an army as well as limiting the amount of special units that can be used in that army *(Hærum, 2015) [2].*The point system is used to control the size of an army through a set point limit *(Hærum, 2015) [2]*. In fact the point system is the entire basis of army list building in Warhammer 40,000. The basic premise of the point system is that a point limit is set, this can be any value theoretically, however there are standard point limit values that can indicate the type of game that will be played. It can also give an indication to the duration of the game, with higher point limits resulting in longer games *(Hærum, 2015) [2].*There is a list of common point limit values that are shown in table 2.1 from *Zikic’s (2016) [10]* thesis, that are used for different types of games. For example a 200 point limit game would generally be a single squad or up to four squads. These are often referred to as ‘kill teams’. Beginners tend to use a point limit of 1000, which is considered a casual type of game. A1500 point limit is the lowest point limit for tournaments. A common point limit for tournaments would be 1850; this allows a more serious player to create a more varied army with more expensive units. Larger tournaments can go up to a 2000 point limit, these games usually last an entire day. Anything above 2500+ is quite uncommon and is usually an opportunity for a player to show off their entire collection.

When building an army list, the set point limit should not be exceeded. Each army contains a plethora of different units and equipment, which both have an associated point cost. Units can be represented as a single model or as a squad *(Hærum, 2015) [2].*A squad has a total point cost that is made up of the individual point cost of each squad member. A squad has a number of options, such as upgrading the default equipment, adding more members to a squad and replacing weaker squad members with stronger ones *(Zikic, 2016) [10].* Using any of these three options will result in the point cost of that squad changing. For example, upgrading to better equipment will increase the point cost of a unit by the amount that equipment costs *(Zikic, 2016) [10].*There are also additional rules and restrictions depending on the faction and particular squad or unit type. This will be discussed in more detail in the next section, using the Space Marines faction as an example.

There are two ways of organising an army list in Warhammer 40,000, these are unbound and battle forged armies. Unbound armies can be created with any combination of units and squads of that particular faction, with no restrictions. On the other hand, battle forged armies are more restrictive, because certain squads and units are needed to form various detachments within an army. The most common types of detachment are combined arms and allied detachments *(Zikic, 2016 [10]; Codex: Space Marines, 2015) [11].*

Lastly, units in a Warhammer 40,000 army are classified under different roles. At the top of the hierarchy are Headquarters units (often abbreviated to HQ) .HQ units are commanders and powerful leaders and are often composed as a singular model. Elites are powerful rare units, but are limited in their deployment. Troops are the most common unit in an army and are plentiful, albeit not as powerful as elites units. Fast attack units are very fast units that are usually responsible for reconnaissance and scouting. Heavy support units wield the most powerful weapons in the game and can also be the most powerful creatures (depending on the faction). Note that Heavy Support units are also the most expensive in terms of point cost *(Priestley et al., 2004) [12].*

## 2.2 Space Marine faction rules

There are a number of different rules and options that are specific to the Space Marines faction when constructing an army list. These rules and options apply to units that are composed of a single model, but there are also additional restrictions and options for units that are composed of multiple models (squads). These rules and options can be found in the datasheets section of the *Codex: Space Marines (2015) [11]* book. The rules and options within the datasheets include; Unit Composition, Wargear, Special Rules and Options. Each type of unit in a Space Marines army has an associated datasheet. For example table 2.2 shows the datasheet rules and options for a HQ Captain unit.

|  |  |
| --- | --- |
| **Unit Name:** | Captain |
| **Role:** | HQ |
| **Points Cost:** | 90 points |
| **Type:** | Infantry(Character) |
| **Composition:** | 1 Captain |
| **Wargear:** | • Bolt pistol  • Chainsword  • Frag grenades  • Krak grenades  • Iron halo  • Orbital strike (Chapter Master only) |
| **Special Rules:** | • And They Shall Know No Fear  • Chapter Tactics  • Independent Character |
| **Options:** | * May be upgraded to a Chapter Master...40 pts   • A Captain or Chapter Master may replace his chainsword with a relic blade...25 pts  • A Captain or Chapter Master may take a storm shield...15 pts  • A Captain or Chapter Master may take artificer armour...20 pts  • A Captain or Chapter Master may take items from the Melee Weapons, Ranged  Weapons, Special Issue Wargear and/or Chapter Relics lists.  • A Captain or Chapter Master may replace his bolt pistol, chainsword, frag and krak  grenades with Terminator armour, storm bolter and power sword...30 pts  • A Captain or Chapter Master in Terminator armour may replace his power sword with a relic blade...10 pts  • A Captain or Chapter Master in Terminator armour may only take items from the  Terminator Weapons, Special Issue Wargear and/or Chapter Relics lists. |

Table 2.2 – Depicting a Space Marine Captain and its unit properties, upgrades and options. Taken from the *Codex: Space Marines (2015, p.289) [11]* book.

As mentioned previously, there are some additional rules and options that apply if a unit is a squad. For example a Troops unit of Scouts has the same datasheet structure as the HQ Captain, however there are some additional rules and options that apply because a Scouts unit is a squad. Additional options for a squad involve a minimum and maximum unit composition. Usually the minimum is the default size. In a Scout squads case this would be minimum of four Scouts and one Scout Sergeant and a maximum composition of nine Scouts and one Scout Sergeant. There are also certain weapons and wargear that only one squad member can take for the entire squad. For example only one Scout in a Scout squad may take the following: Heavy Bolter at 8 points and a Missile Launcher (with frag and krak missiles) at 15 points. Refer to the *Codex: Space Marines (2015, p304 - 305) [11]* book on the datasheet for a Scout squad.

It should be noted that there are many different variations for the rules and options of each unit in a Space Marines army. There are far too many to cover in detail in this report. But the overall structure of the Captain and Scout Squad example units can give an indication to the complexity of different units in the Space Marines faction, particularly the difference between a squad unit and a unit that is an individual model.

## 2.3 Existing table top army builders

The traditional approach to building army lists in Warhammer 40,000 is achieved by simply looking up the rules, options and point cost for each unit, in the factions corresponding codex book. Then each units configuration would be hand written on a piece of paper and the totalling up of all the different units point costs would be done manually with a calculator, to ensure the point limit hadn’t been exceeded. Building army lists in this manner can be a tedious and time consuming process, especially with larger armies. In fact this method of army list building is still use. The *Warhammer Community (2017) [8]* provides a downloadable Warhammer 40,000 army roster template that can be printed out to create a hand written army list. Although this is slightly easier than building from scratch, the hand written element is still prevalent. This particular template also appears to be only designed for people creating battle forged armies, so unbound armies aren’t covered.

There are also a number of existing applications and online websites that allow army lists to be built for various tabletop games involving miniatures. These eliminate the problems of hand written army lists, as they automatically calculate the point cost as you add more units and also ensure the strict rules and options are met for different factions and their units. There is no need to check the codex for every unit in an army. The process is much quicker and probably more accurate too. In the next section two digital army builders will be presented and discussed.

Warscroll is a Games Workshop official, online webpage based army list builder for Warhammer Fantasy: Age of Sigmar, which has close ties to Warhammer 40,000, but is set in a medieval era fantasy setting as opposed to the 41st millennium. This online army builder can be found on *The Warhammer Community (n.d) [9]* website. Warhammer Fantasy works very similar to Warhammer 40,000 with the point system.

After exploring this army builder for a while, it appears to allow a maximum point limit, it also allows a user to choose different factions under four subheadings that correspond to a Grand Alliance. These grand alliances are; Chaos, Order, Destruction and Death, each with their own list of factions. Upon choosing a faction there are different categories of an army that can be chosen, including; Leaders, Units, Behemoths, War machines, Battalions and even Scenery that can be used on the tabletop. These options are presented as a pre-configured drop down list and give options to alter the equipment of a unit. Adding a unit automatically adds its point cost to the total point limit. There are also options that allow a current army list to be saved, and can continue being built at a later date with a load option. However the save and load features only work with the use of cookies in a web browser, which is somewhat limiting. There is another save option that allows the created army list to be saved to a PDF file, which can be printed and will be useful if a physical copy of an army list is required.

BattleScribe*(Taylor, 2010) [14]* is another army list builder that is used not just for Warhammer 40,000, but a variety of other tabletop games that involve building an army roster. This army builder is a local downloadable application, rather than relying on a web browser. It’s also multiplatform and has versions for IOS, Android, Kindle Fire, Windows, Apple Mac and Linux. This report will be focusing on the Windows desktop application version.

According to *Hærum (2015) [2],* this program is a very general tool that provides a framework for other users to create more specialised rule sets. This means that technically any game that revolves around tabletop gaming can be catered for. The idea is, that a user creates their own rule set for their desired tabletop game, and then shares this with the wider community, allowing them to use their rule set as a template. The program doesn’t provide any pre-defined data of any particular tabletop game or faction, but merely the framework for a user to build upon, or use the rule sets created by other users (*Hærum 2015) [2].* This is also stated on the BattleScribe website *(BattleScribe, n.d.) [13].*

Upon using a Warhammer 40,000rule set, provided by *Bishop, et al. (n.d.) [15]* for this program, it allows a user to choose any faction within the Warhammer 40,000 domain. It covers both unbound and battle forged rosters, with all the units within a chosen faction being made available. The game system file ensures all the rules of Warhammer 40,000 are met by every available faction, and each faction has their own specific set of rules, for each unit (*Hærum 2015) [2].* This is much like an inheritance structure, where the game file is the super or master class, and the factions are subclasses of the game file master class, and inherit all the rules for Warhammer 40,000 in addition to their faction’s specific rules.

Figure 2.3 shows how the desktop version of BattleScribe can add a Scout Squad from the Space Marines faction. The windows are split into four panels. The left most panel labelled Available Entries shows all of the available units. Units are classified by their roles (mentioned previously in section 2.1 of this report), such as a Scout squad being classified by the Troops role. The next panel labelled as Roster Selections displays the current roster of an army. Each unit within the roster view can be expanded to see

Each individual member of a squad and their details including point cost and equipment. A squad member or an entire squad can also be removed from this panel. The right most panel, displays the currently selected unit and displays their options for upgrades. For example selecting an entire Scout squad will bring up the options that apply to the entire Scout squad. Selecting a squad also allows you to add additional squad members. Ifa squad member only is selected, then the options that only affect the individual member are shown instead. The bottom panel shows warnings of any incompatibilities of the army roster list. For example in figure 2.3, a warning appears stating that the maximum size of a Scout squad has been exceeded by six.

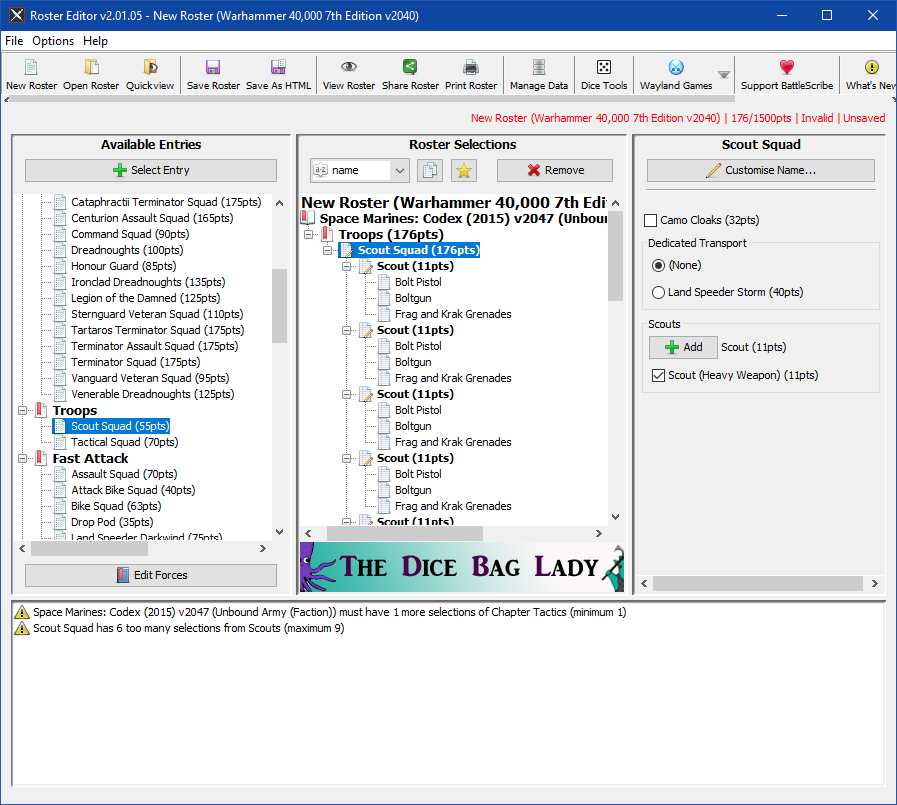


Figure 2.3 – Depicting a snapshot of BattleScribe in use to create a Scout squad (*Taylor, 2010) [14].*

BattleScribe also possesses a number of options that allow a user to view, share and print their army roster list. Viewing the roster opens a separate window and provides an overview of the army list in a more readable format. Sharing the army roster involves sharing to various social media sites and forums. Printing allows you to print a physical copy of the army list which can be useful if no electronic devices are available. There are quite a few other options as well but they are locked behind a pay wall, meaning that to get the full experience you have to pay an annual supporter fee. This fee also removes the advertisements that are prevalent on the application *(BattleScribe, n.d.) [13].*

# Implementation

## 3.1 MVC architecture

Applications that have a GUI (graphical user interface) or even any form of user interface needs to be able to be integrated with the rest of the program. This can be a major software development problem, with the solution often relying on a design pattern *(Syromiatnikov and Weyns, 2014) [7]*. One proposed design pattern for this domain is the MVC (model view controller) design pattern and its variations. The model aspect represents the data that will be used and updated within the program. The view is essentially the user interface (what the user see’s) and the controller is what drives the updates to the model, which is then translated to the user through the user interface *(Syromiatnikov and Weyns, 2014 [7]; Stocklein, J. et al. 2009) [6].*A crucial aspect of MVC architecture is the separation of concerns principal. Within the context of MVC, this means that the model component is completely unaware of the view component, and thus relies on the controller to relay information back and forth between the model and view. This principal can allow for a clear modular design and can make code easily maintainable *(Syromiatnikov and Weyns, 2014) [7]*. This is mainly due to the fact that if an update is made to the model code for example, then it won’t affect the views functionality, due to the model and view being separated from each other *(Stocklein, et al, 2009) [6].*

In figure 3.1, this concept of the separation of concerns can be illustrated. It shows how a user interacting with the view (for example a user entering text into a textbox and pressing submit) updates the model through the controller. Then the updated model uses the controller to change the view for the user to see. In this case it could be the text the user inputted being displayed to them

Note, how at no point are the view and model directly communicating with each other, this is achieved through the controller instead.

## 

## 

**The controller updates the model, with the captured user input**

## 

**The controller captures the input from the view**

## 

## 

## 

**The controller updates the view**

## 

## 

**The model notifies the controller of the change**

**The user see’s that change**

**User interacts with the view**

## 

## 

Figure 3.1 – Depicting a typical MVC design pattern, adapted from *Jailia’s, et al. (2016) [3]* MVC diagram labelled Fig 1.

Another advantage of using the separation of concerns principal for MVC architecture is that it enables the application to have multiple views and even multiple controllers. These can be created on the fly and altered without interfering with the model component of the program. Usually applications would have multiple views of the same data*(Curry and Grace, 2008) [16]*, with the MVC design pattern along with the separation of concerns principal facilitating it.

## 3.2 Java and the JavaFX framework

In order to implement the MVC design pattern that was discussed in section 3.1, a programming language platform needs to be chosen. A proposed language for this task could be Java. Java is a very popular programming platform, mainly due to the fact that it’s cross platform. Java code can run on any hardware platform and still execute the code, providing the JVM (Java Virtual Machine) is installed on the host system *(Minor and Gewali, 2004 [4]; Reed, 2002 [5]).*Java is also a purer object oriented paradigm (although not one hundred percent), unlike one of its main competitors C++, which can actually write a program without the need for any classes. Whereas any Java code needs to be contained in a class, in fact, even Java’s standard library is essentially a large set of classes *(Minor and Gewali, 2004) [4].*Using a purer object oriented language such as Java would be good for this domain, as the miniatures in Warhammer 40,000 can easily be represented as objects that are created by classes. Another advantage of Java is that it’s a very high level programming language, making it easier for the developer to focus on the actual implementation of the program, rather than the low level details of pointer manipulation and memory management. In Java, memory management is taken care of automatically through the garbage collector, which runs on the Java virtual machine. The only downside to this is that it makes programs run slower, than if a language employed manual pointer manipulation and memory management. This creates a trade-off between the ease of use, but slower performance using Java, and the extra effort required to manage memory, but have faster performance, using languages such as C++*(Reed, 2002) [5].*

Now that a programming platform has been established, a framework designed with GUI’s in mind need to be discussed. A popular Java framework for GUI design, is the JavaFX framework. This framework has been designed from the ground up as a Java API, which means it can make use of any other Java library, including the standard Java library. Another good thing about JavaFX is that the style of the application can be achieved using external CSS files. This keeps the styles separate from the implementation. JavaFX also integrates seamlessly into the Java runtime environment and Java development kit, making any JavaFX application cross platform, just like Java. JavaFX also supports a number of common UI controls such as check boxes, combo boxes, buttons, radio buttons and dozens more*Docs.oracle.com (n.d.) [1].*

# Conclusions

The purpose of this report was to gain an insight into how Warhammer 40,000 works in regard to army building. The point system was discussed and how this plays a vital role in the army building process. The entire application will be based around the accumulation of points, so this was a crucial part of the report. Another important aspect was around the idea of using a battle forged or unbound approach to army building. Considering this application is designed for a beginner player of Warhammer 40,000 I think it would be justified to take the unbound approach when designing the application, this is because for a novice, the battle forged style of organising an army into detachments could be overwhelming. It may also complicate the implementation of the program, if a battle forged approach was taken.

The report also discussed the specific restrictions for the Space Marines faction of Warhammer 40,000. This is also important to the design of the program because I intend to provide data for this faction only and to ensure the restrictions of this faction are met, despite there being many other factions in the Warhammer 40,000 universe. This is because implementing all of the available factions would be a colossal undertaking, and I feel as though I wouldn’t have the skill set to do a project of that scale, therefore I chose the Space Marines faction only. Also, this is a faction I play myself personally, so I’m fairly familiar with them, also they are the most popular faction in Warhammer 40,000, and the faction that a beginner is likely to start collecting.

Another aspect of this report was concerned with how existing tabletop army builders have implemented their solution. Two were chosen, Warscroll and BattleScribe. Both of these had desirable features that would be useful for my application, such as saving, loading clearing and also crucially, printing. I also like some of the UI controls such as drop down menus for unit selection on Warscroll.

The second part of the report discussed the technology behind the implementation of the program. The MVC (model view controller) was discussed, and why it’s an excellent choice for GUI design, mainly because it allows for code maintainability as well as a clear design, by separating each function of the program among the model, view and controller.

Finally Java and the JavaFX framework were discussed. Java is a very popular language mainly due to the fact that it’s cross platform. This is a favourable trait because it allows me potentially develop the application on other platforms later on. Java is also easier to develop on compared to some languages that don’t have automatic memory management. This would give me more time to focus on the application itself rather than having to deal with low end details of the program. Java is also the language I’ve had most experience with. Considering I’m using Java as the language of choice, it would be logical to choose a Java framework for the GUI aspect of the program. In which I chose the JavaFX framework. This framework makes use of a lot of the Java libraries that I already know and would therefore be suitable for this project. It also has a lot of rich features that would enhance the look and feel of my application.