Warhammer 40000 Legiones Astartes Age of Darkness Army Selector/ List Printer

Ian Nevills
James Laubach
Nicholas Reid

Abstract

To condense the army selection book (hitherto: Legiones Astartes Book) and the rules neccessary for the play of that faction specifically from the Core Rules Book into an easily navigable format to allow for the quick construction of army lists/builds for veteran and novice players of Warhammer 40000 alike.

1. Introduction

Warhammer 40k is an expansive and very complex tabletop game and its rules can be daunting even for veterans of the game. For novices, navigating thick tomes of factions upon subfactions of units and variants to forge their chosen army list can be nearly impossible. Our goal is to condense the faction book of one faction (Legiones Astartes, "Space Marines") within the setting of the "Horus Heresy" expansion to base 7th edition Warhammer 40k to streamline army creation and hopefully streamline game play by providing an army-list document that provides reference to particular rules. With that in mind, however, we are not seeking to make it so that any person could use our program in lieu of buying the particular books in question. Our plan is for our program to: Allow the user to search/page through a catalouge of units pertinent to the faction in question, and select units (and eventually upgrades) within the bounds of standard army templates in the Legiones Astartes Book and point limits imposed on normal game play, and to output a formatted and easily navigated document that provides information (basic stat lines, a list of upgrades, list of rules (but no detailed text), and an index indicating where further information can be found to clarify each) to streamline game play.

1.1. Plan Goal 1.

None.

1.1.1. Plan Goal 2..

None.

1.2. Background

Warhammer 40000 Legiones Astartes Age of Darkness Army List
Warhammer 40000 Core Rules Book 7th Edition

Primer:

Warhammer 40k is a tabletop science fiction strategy game set in a dark and distant future. Within 40k there are many different factions with equally different game-play fact and origin stories. Warhammer 40k is a game literally buried in lore and reasons for the how and why its factions behave and interact the way they do. In fact, a more accurate description may be a tabletop science FANTASY game given that almost all the standard fantasy races anyone familiar with Dungeons and Dragons or Tolkien's novels would recognize. Though, with that recognition comes the next problem, each of these races is at first familiar but also very different than their traditional archetype. A further description though is beyond the scope of what I intend to cover at this time. For now, I will cover the subject of this project.

The Space Marines:

For our part, we have chosen the Space marines as our subject for this project, they are probably the most widely recognized, if not known to be a part of the setting, and are the likely poster children to introduce new players into the setting via their use in novels and games based on the setting. Space marines in the context of Warhammer 40k, one part warrior monk, one part Roman legion, one part crusading host, and one part Arthurian legend, wrapped in a ton and a half of powered armor and genetic engineering. In the lore of Warhammer 40k, they are the ultimate soldier and the peerless defenders of Humanity, sent into the harshest fighting and the most important conflicts.

The actual game-play is much less....stellar. At their core, the Space marines are a force dedicated to somewhat durable heavy infantry formations with a wide selection of weapons and upgrades and a fairly well rounded stat-line. Everything else in their lineup are either walkers intended for direct support of those infantry or various vehicles and tanks that almost universally double as battle taxis to ensure the infantry reach where they need to be and provide a large enough wall of guns to ensure they can do their job when they get there.

The Horus Heresy:

The Horus Heresy is one part expansion and one part prequel to the main Warhammer 40k setting, and is the scope of our project. The Horus Heresy is a civil war taking place in the 30th Millennium, ten thousand years before.

1.3. Challenges

Our challenges are that the more information we wish to add up to our full goal, the potential complexity and required time to complete the program grows exponentially. There is a lot of information in Warhammer 40K that must be processed for basic game play such as building an army and recognizing factions, unit types, and unit upgrades. Each part has different rules for game play as how factions interact, strength and weaknesses of unit types, and how each unit is benefited by upgrades. So with each new piece of information the complexity of the program increases.

2. Scope

Our bare bones level of functionality that we would call ”complete” is the ability to choose an army organization, choose units, and to force users to select army composition within the confines of the army organization. When choosing units our bare functionality would have the system display the unit name, the unit index number, the unit base stats and point value,a text box with stat definitions, and the unit composition.

A near term stretch goal would be to recognize the point limit imposed by the game and to take the point value of units and to tell the user that they have exceeded the point limit imposed by game play.

A stretch goal would be to allow the selection of upgrades, show how they alter the point count, and provide an index to the appropriate page for each upgrade to allow the user to find more detailed information needed for game play. A further goal down that road would be to list appropriate rules to each upgrade (and the base units) and allow another index to provide a similar ability to quickly find needed information for those as well.

The second main stretch goal would be to make the program be able to read back in its output list (or a code that would generate an identical list) to allow a user to edit a list after the fact rather than rebuilding it from scratch.

2.1. Requirements

2.1.1. Functional.

- User need to have container to hold information.
- User needs to be able to add and delete information as needed.
- User needs to be able to create, identify, and access information file.

2.1.2. Non-Functional.

- UI that the user interacts with is easily navigated and simple to learn.
- Make the program adaptable so that a different database (say after an update to the game) could be used instead of the one provided, as long as the format stays the same.
- Ensure that the database is accurate to the source material.

2.2. Use Cases

Use Case Number: 1
Use Case Name: Add Units
Description: User wishes to add a unit to their army. Look the list of units and Click add unit to list.

1) Navigate with button to list with the desired units.
2) User clicks add unit to list button.
3) The list is updated to reflect the new unit in the army.

Termination Outcome: The user has a new unit in the army list.

Use Case Number: 2
Use Case Name: Delete unit
Description: User wishes to delete a unit from their army. Look at users list and click Delete button.

1) Navigate to the users list through the interface
2) User clicks a delete button.
3) The list is updated to reflect the new army.

Termination Outcome: The user has lost a unit from their army.

Use Case Number: 3
Use Case Name: Save user list
Description: The user wants to save their unit list.

1) Navigate to the users list through the interface
2) Click the save list button.
3) The current list is saved.

Termination Outcome: The current user list is saved.

Agile use cases

1) As a user I wish to be able to add a unit to my list.
2) As a user I wish to be able to see my list of units.
3) As a user I wish to be able to see the stats of a selected unit.
4) As a user I wish to be able to create a new list.
5) As a user I wish to see the amount of points that I have available to me.
6) As a user I wish to be prevented from adding more units when I run out of points.
7) As a user I wish to be given basic instructions on how to use the program.
8) As a user I wish to be able to delete a unit from the list.

2.3. Interface Mockups

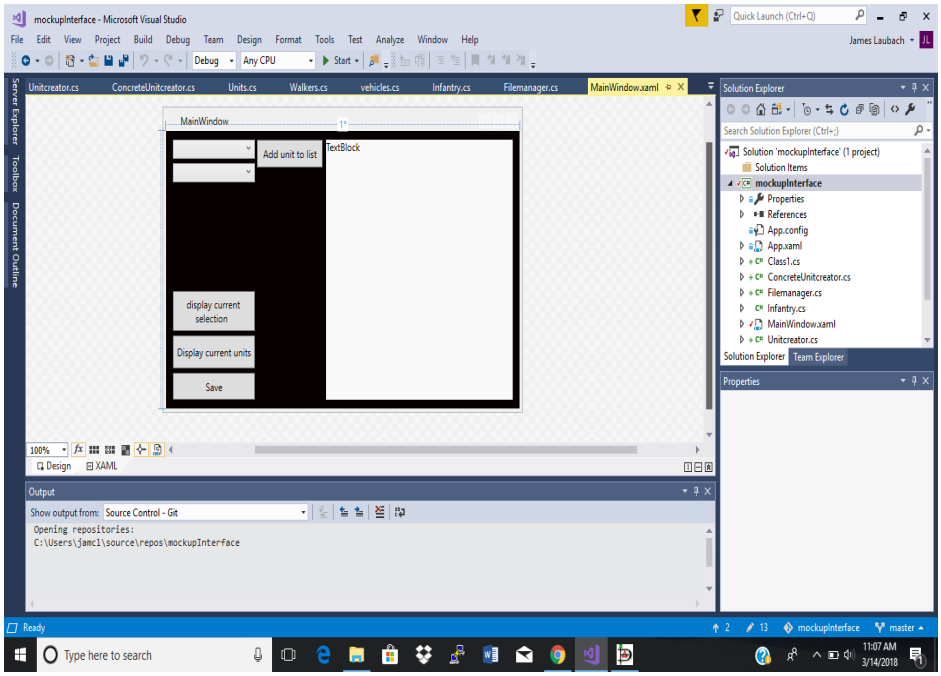

Figure 1. A picture displaying an early mock-up of our UI , in this early iteration the upper drop-down menu would allow the user to select which class of unit (infantry, vehicle, or walker) and then the bottom menu would display the appropriate units of that class. The user could then decide to display the stats of that unit (in the Textbox) and add the unit. The user could then choose to display the current list and save the list.

3. Project Timeline

2/15/18 —– Presented proposal to the class
2/28/18 —– Initial Database (Unit names, index, point values) finished and undergoing conversion to Access format.
3/2/18 —– Initial database fully converted to Access, UML finished, timeline finalized, design patterns chosen.

3/7/18 —– Text describing stat-line figures and copyright information established, sample list format 1 ready for review. Prototyped classes,

3/15/18 —– Next check in ready, UI demonstrating partial functionality, at least able to access initial database

3/22/18 —– One or two main classes fully implemented and functional, with list builder fully functional with initial database

3/29/18 —– Adding information to initial database to form final database (Unit names, index, point values, composition, statlines), begin altering three complete classes to suit

4. Project Structure

Our program required a good deal of thinking and several iterations to decide on how best to break down our project. After some research we arrived at the conclusion that we had three basic classes of unit (our source material records six, but the actual differences in data displayed only support 3). One of our classes, infantry, also has a sub class where data is displayed differently as well. For our purposes here the actual differences in data displayed are unimportant, but we decided to arrange the three main classes in a factory pattern as sub classes of the class Unit with a UnitManager class acting as our assembler that the client (through our UI) interacts with. The user never directly accesses anything in the Unit sub classes.

Infantry's sub class (hitherto known as DualLine) is written as a decorator on the Infantry class that is called to record sub-unit information that makes up various infantry squads. Infantry records the overall squad stats, where DualLine adds the ability to record squad-member and squad-leader stats, both of which are separate and distinct from the overall.

In the program class "UnitManager" is setup to call a class "database" that will send a query for information in the database. The factory pattern setup allows the classes to call on "UnitManager" and through the "UnitManager" call on the database to the get the information the classes call for.

4.1. UML Outline

The UML outline for our program

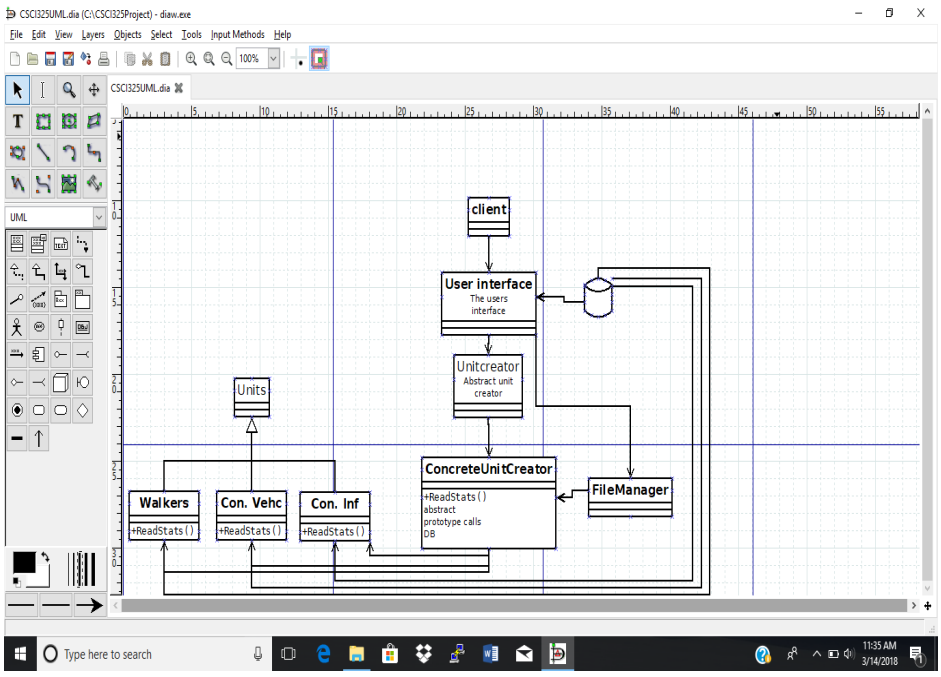

Figure 2. A UML that describes the interactions on a broad level between the client, our user interface, and our various classes that supply functionality

4.2. Design Patterns Used

We used a factory pattern in conjunction with a decorator to store our distinct unit information on three different classes and one sub class. Infantry,, Vehicle, and Walker act as products called through a UnitManager factory that is acted upon by the UI. DualLine is a decorator that adds the ability to record multiple statlines to Infantry's

5. Results

Though we have largely neglected this portion till now, that is largely because our progress was slow as we tried to finalize what we could accomplish and what was at best a pipe dream. As it stands now, we have several achieved several of our objectives. We have finalized our UI and have a working list builder, though our decorator in it is not. Our database is closing in on its final stages and is a work in progress. Within our UI and UnitManager (list builder) we have the ability to add units, subtract units, display individual unit stats, and to print our entire list. As yet we do not have the ability to print units with multiple stat lines, we do not have stats in our database (merely the framework information needed to navigate it), and we do not have the ability to save our list of units. We are also bound to the single database because unless we currently do not know how to proceduraly create buttons from information in our database. Currently our extra features are the ability to see both the list and our list.

5.1. Future Work

Where are you going next with your project? For early deliverables, what are your next steps? (HINT: you will typically want to look back at your timeline and evaluate: did you meet your expected goals? Are you ahead of schedule? Did you decide to shift gears and implement a new feature?) By the end, what do you plan on doing with this project? Will you try to sell it? Set it on fire? Link to it on your resume and forget it exists?

References

[1] H. Kopka and P. W. Daly, *A Guide to LaTeX*, 3rd ed. Harlow, England: Addison-Wesley, 1999.