Fate-Weaver

Age of Sigmar Assistant General

1-Intro:

The goal of Fate-Weaver is to provide is to assistant Age of Sigmar generals decision making in the heat of combat.

To achieve this, I would like Fate-Weaver to do the following:

* Calculate how much damage one unit can do to another and its kill percentage (how likely the friendly unit is to destroy the enemy unit in one round of combat)
* Calculate a unit’s value compared to the rest of its army.
* Use the above to suggest the optimal activation order for units currently in combat.
* Calculate likely hood to complete a charge based on where two units are positioned.
* Use the above to suggest where to engage the enemy.

But most of all, I want to use this project to challenge my programming capabilities and learn lots of new tricks along the way.

2-Loading the Armies:

2.1-First Draft:

I decided to begin my project by implementing a way to load armies into objects as this would allow me to already have data on units in the system before wanting to test any calculations. This would also allow me to define much of the program structure early on.

Diagram, table

Description automatically generated

*Figure 1 – Army ERD First Draft*

I started by drafting up an Entity Relationship Diagram using <https://cloud.smartdraw.com> to get an idea for generally how I wanted to structure the classes within an army. This can be seen in Figure 1 above.

2.2-Loading From Excel:

For this project I need to store lots of static data about unit stats and I believed the best way to do this was in an excel document. This also gave me the opportunity to learn something new – figuring out how to make my code interact with excel.

I looked into available packages for accessing excel with C#, specifically Microsoft Interop and IronXl, and decided to use the later. I made this decision as it had a more intuitive way of adding and reading values from cells and it processes much faster than Interop.  
  
2.3-Army Object:

The first object I created was Army. The role of this object is mainly to store the collection of units that make up an army as well as related variables. To begin with I created a constructor that takes a file path for an army list text file.  
The constructor then reads this text file to find a battletome (file that contains all information for a given army) and a list of units that should be contained with said army.

I used a try catch to throw an error message if a unit could not be found within the battletome, indicating something wrong with either the text file or excel spreadsheet when I debug.

Text

Description automatically generated

*Figure 2 – The First Bit of Code - Wooo*

2.4-Unit Object:

Next I created the Unit object. This object is responsible for holding all key information about a unit within the game that will be needed for calculations later down the line. The constructor takes a battletome WorkSheet and a string for the units name. It then searches the battletome for a unit with this name and saves the row number as unitlocation to use later.

I ran into my first error here. The console was receiving the error message that it could not find units I had added to my army list. I found this was due to the capitalisation not being an exact copy between the two files. I thought that if this was an error I made after just creating data for those files, then it would be a common error for users. So, to combat this, I used the simple String.ToLower() function to normalise capitalisation such that any variation could be translated between the two files.

Text

Description automatically generated

*Figure 3 – Pesky Capitals*

2.5-Weapon Object:  
A screenshot of a computer

Description automatically generated with medium confidence

*Figure 4 – Weapon Object*

The weapon object stores the statistics of a weapon carried by a unit. Populating the weapons proved to be my first big challenge. In the excel document, column J denotes how many weapons a unit carries.

To contain where I needed to use the IronXL package, I decided I would pass an array to the Weapon constructor generated when I open an excel document in the unit object. I know a weapon has 8 key statistics so I used to for loop to progress through each cell and every 8th cell it would create a new weapon. The challenge came from how I would move through cells and IronXL contains no in built function for finding the next cell in a row. To solve this, I created the flow chart shown in figure 5 to break down the problem. This allowed to find the next letter in the alphabet and, once Z is reached, add an additional letter e.g AZ which allows me to navigate across the spreadsheet. I was honestly surprised this isn’t a core functionality for IronXL.

----flow chart

2.6-Reinforced Units:

In Age of Sigmar, units can be reinforced which essentially means they can be doubled or tripled in size. The standard way of denoting a this is to add a \* or \*\* on the end to represent a unit being reinforced and double reinforced respectively. I simply made use of the String.Contains() function to check for a \* and multiplied the new units model count as required.  
Text

Description automatically generated

*Figure 6 – Reach for the stars*

2.7-Refactoring Army Compiling:

Fate-Weaver was now able to read from files to fully populate an army object so I could now move on to start making calculations on two armies fighting. However, first I thought it important to have a little break and then come back and try refactor my code so far to try reduce technical debt later down the line in the project. The changes I made a noted below:

* Changed WeaponType field to an Enum as should only ever hold the value “melee” or “missile”.
* Moved part of code that finds the next cell in a worksheet to an extension method so can be reused elsewhere if necessary.
* Reworked extension method to instead use a string builder as it is more memory efficient as only works on one instance of a string rather than creating multiple instances for substrings.
* Moved reinforcement code to within the unit class code as I felt it belonged there more than within the army class

3-Combat Calculations:

3.1-Average Damage Calc:

Now I can load units into FateWeaver, it’s time to start doing something with them. I decided my first step would be to write code to calculate some key values that FateWeaver can use to help make decisions.  
The first of these values I worked on was Average Damage - how much damage does an attacking unit do to a defending unit with average dice rolls?  
Text

Description automatically generated

*Figure 7 – Just an average amount of murder*

The first thing I needed was the probability of one dice rolling the to hit value, then rolling the to wound value and finally the enemy failing to roll high enough to save and block the incoming damage. The calculation for this can be found in the ChanceToDamage() method.

I then took this value and multiplied it by the number of dice being rolled (totalAttacks) and the damage each hit would inflict. I then looped through each weapon a unit has and totalled it for the average damage.

3.2-Percentage Kill Chance:  
Next I wanted a value to calculate how likely it is that a unit could be killed by another in one round of combat. This will be useful later for helping decide if it is worth engaging the enemy.

I started by creating a check to see if it it’s actually possible for the enemy to be slain in one round of combat as if not, there is no going any further which saves a lot of time. This can be seen below.  
Text

Description automatically generated

*Figure 8 – Looks like you suck*

I wanted to start off simple with figuring out this formula. I decided, to begin with, I would create a formula that just calculates the kills chance with a single weapon. This would be helpful in future as I could allow users to check what would happen if they chose to split their attacks. I decided to implement to this as an overload function for KillChance() that takes in a selected Weapon as a parameter.  
Text

Description automatically generated

*Figure 8 – Nice on the surface*

I had to learn some advanced stats for this one. I took to good old youtube to learn about binomial distribution. Once I felt confident, I had a go at implementing the formula into my code as shown below. After I had coded the formula, all I needed to do was plug in the numbers. The number of trials (n) would be the total number of dice rolled (number of models \* number of attacks). The number of desired successes (k) would be the minimum amount of attacks needed to land (enemy health / damage per hit). Finally, the probability of success of one trial (p) which I had already created the formula for with the ChanceToDamage() method. Just \* 100 and I had myself a kill percentage – wooo!

Text

Description automatically generated

*Figure 9 – Ew, Nerd*

3.2-All Out Kill Chance:

So now I had the probability of killing the enemy with one weapon, but what about if I wanted to use all the weapons a unit carries?   
  
P(AorB) = P(A) + P(B) – P(AandB)

where:   
P(A) = probability this combination or any previous combination does enough damage  
P(B) = any remaining combination does enough damage

Mcaffe – my protection – real time scan

1 – probability of min dmg

*Damage calc*

*Percentage kill*

*Unit value – aggressive and deffensve index*

*Best combat trades \* how likely these trades are to be pulled off*

*Parent + child for changing methods for different army types*

Text

Description automatically generated