Project Rapport Mathias J. Kirkeby, WarGame:

Table of content:

Introduction

Trough the semester of Spring 2022 was the students within IDATx2001 tasked with creating a War Game application as part of their evaluation of the subject.

The War Game was given specification trough out the semester with PDF Files given trough obligatory tasks. Every assignment can be found within the git repository within the "docs\Assignment" folder.

Specification

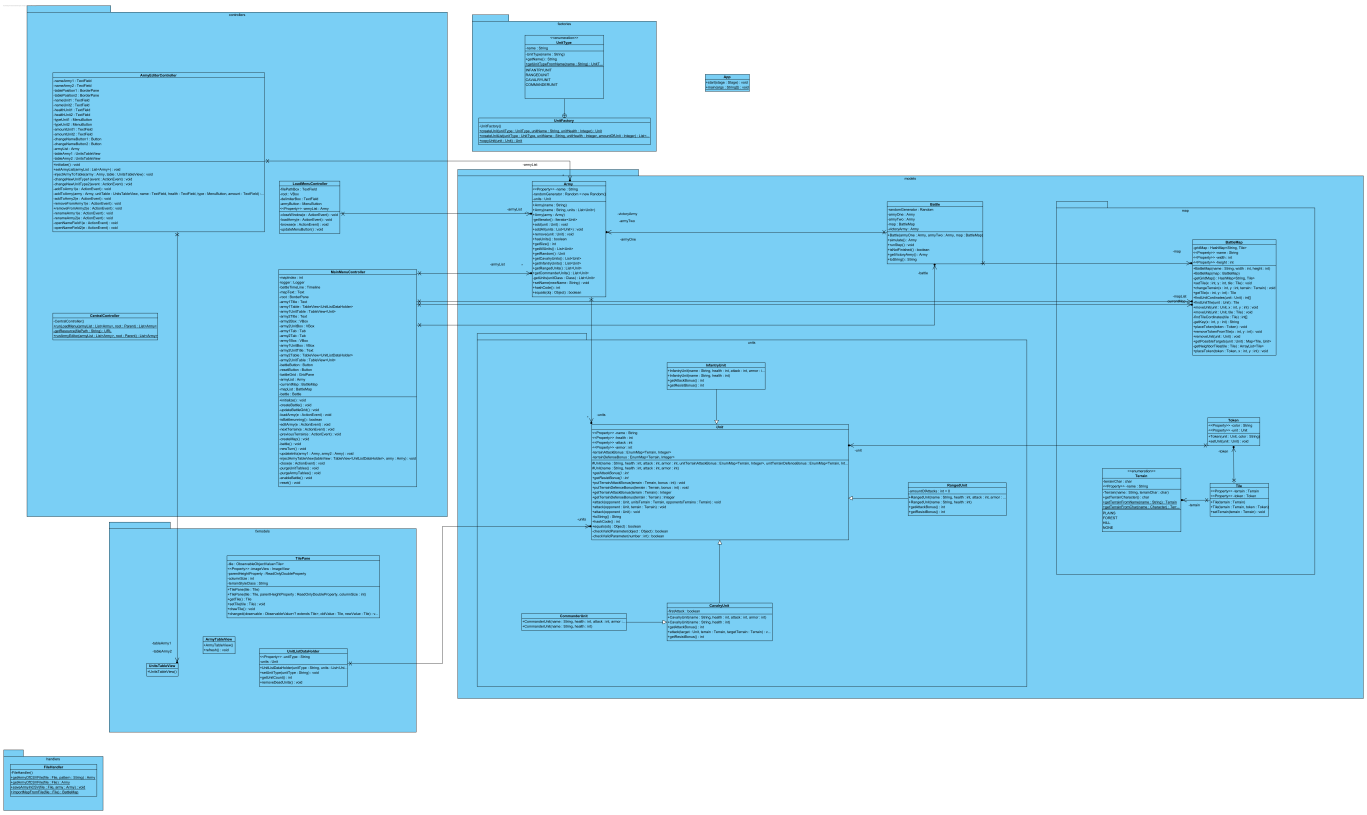
The features this application support is:

- Loading files:
- Editing armies
- Running a battle between armies
- Running a animation when the battle is running
- Tables that updates as the armies losses or gets soldiers

In addition was there multiple requirements give trough the assignments part 1-3. These has been fulfilled after the best of ability.

Design

This project has used a maven architecture with javaFX as a application framework. As for the Class Diagram:



The application is split into mostly 2 parts, Front-end and back-end. This makes it so I can easily copy the backend code and build it a new application with another framework.

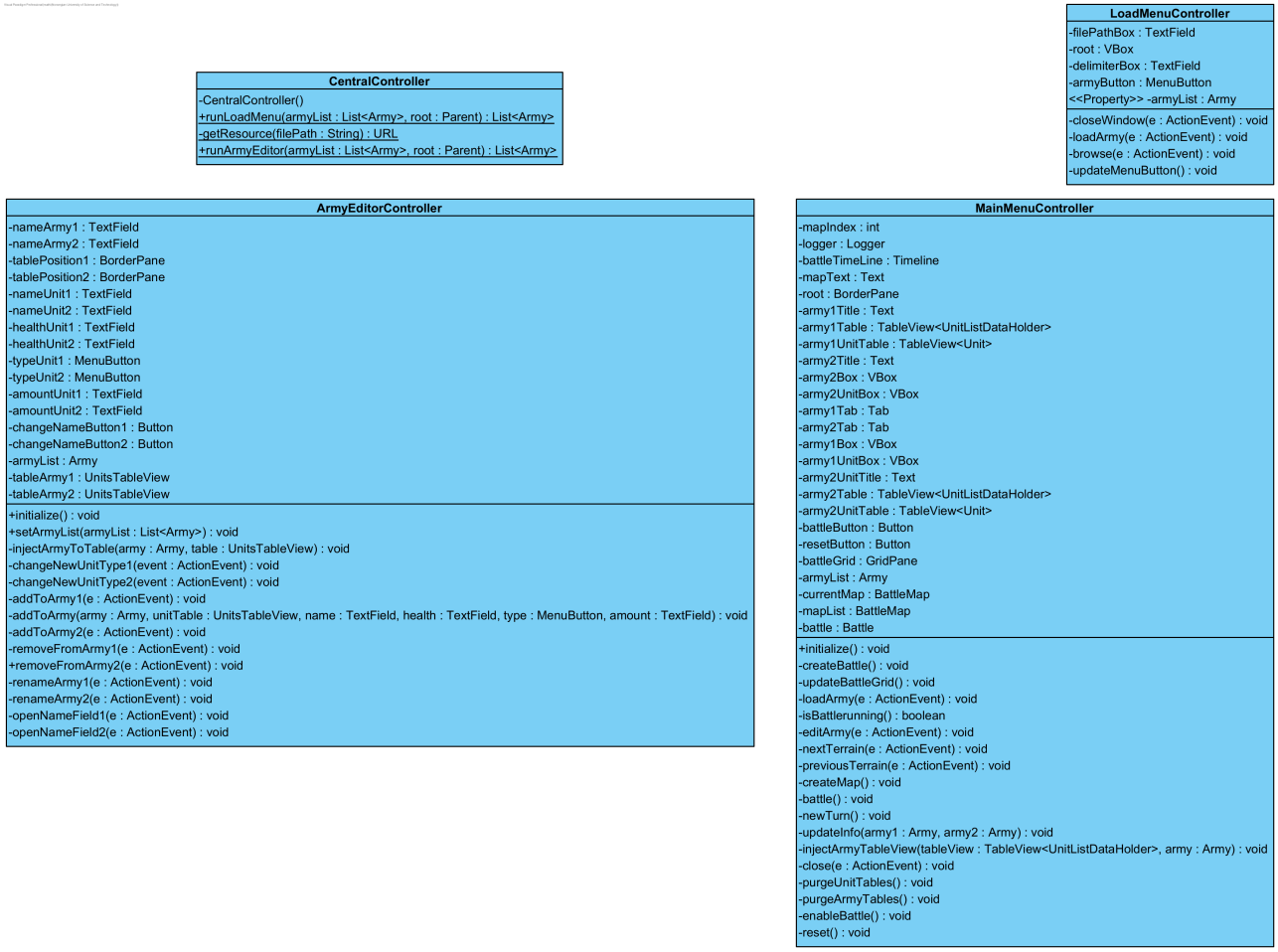
The packages used within the projects is the following:

Front-end

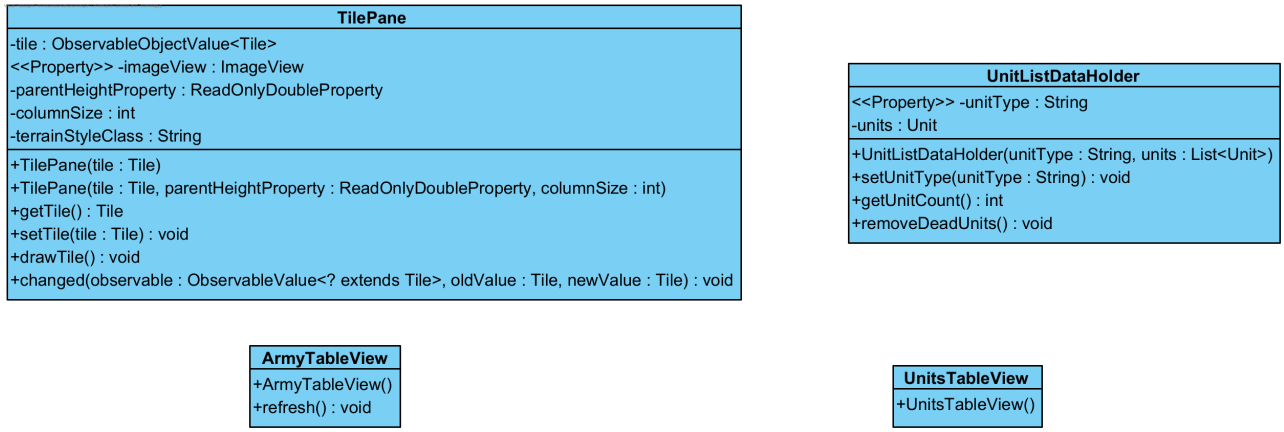
The front-end or gui is designed with a main menu as the central component. This is to make it simpler to add or implement other features. Since the main menu is very static in how it was created it also helps not to manipulate the main window.

- Controllers: The different controllers/facade models needed to implement functionality into to the application. Like mentioned this area is centered of the MainMenuController that calls the CentralController to open the different windows for loading, editor or later features like a map editor or

a custom map importer.

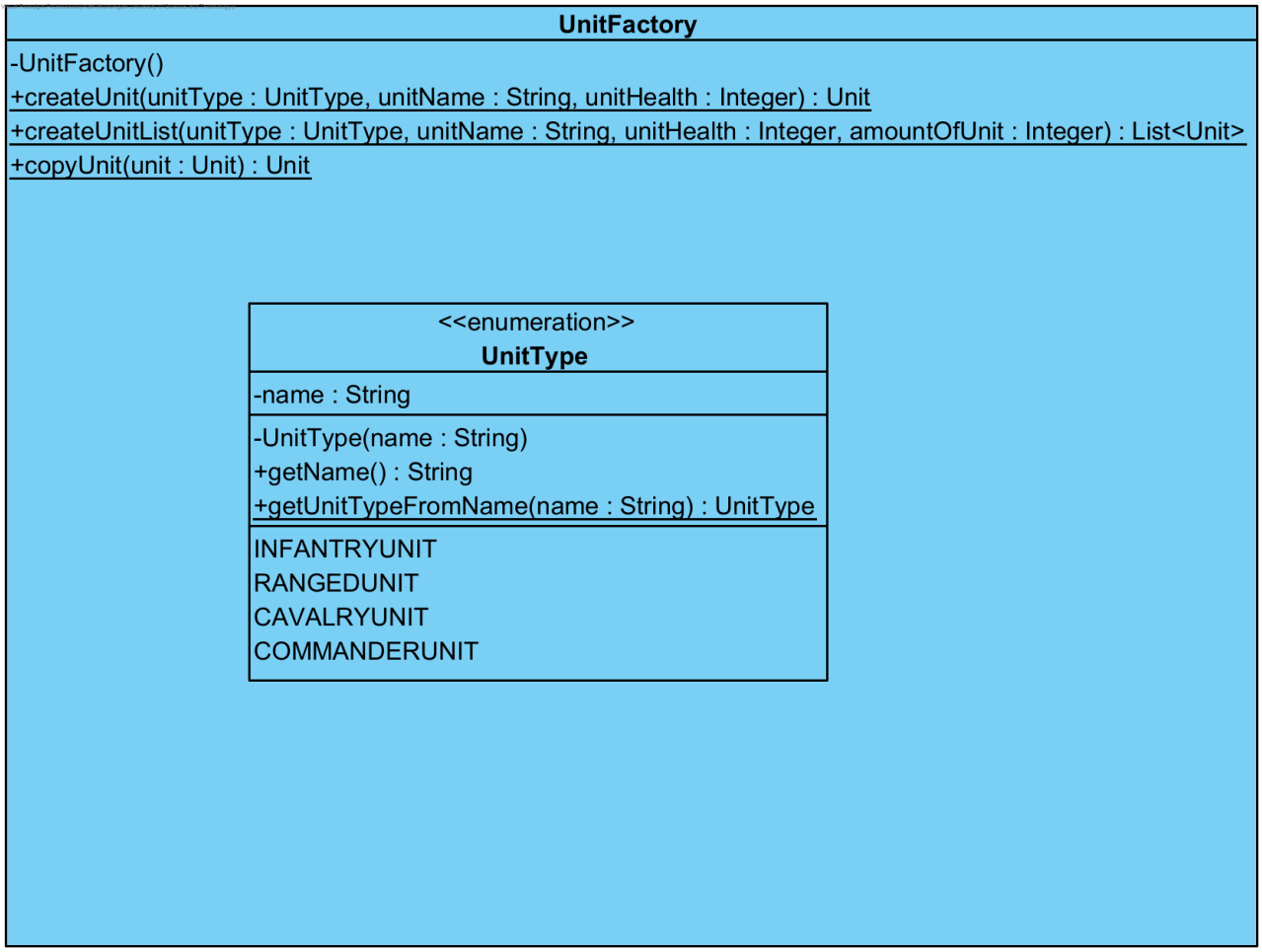


- fxmodels: This consist of objects or classes that is modified from the original javaFX library to allow for more a specific usage.

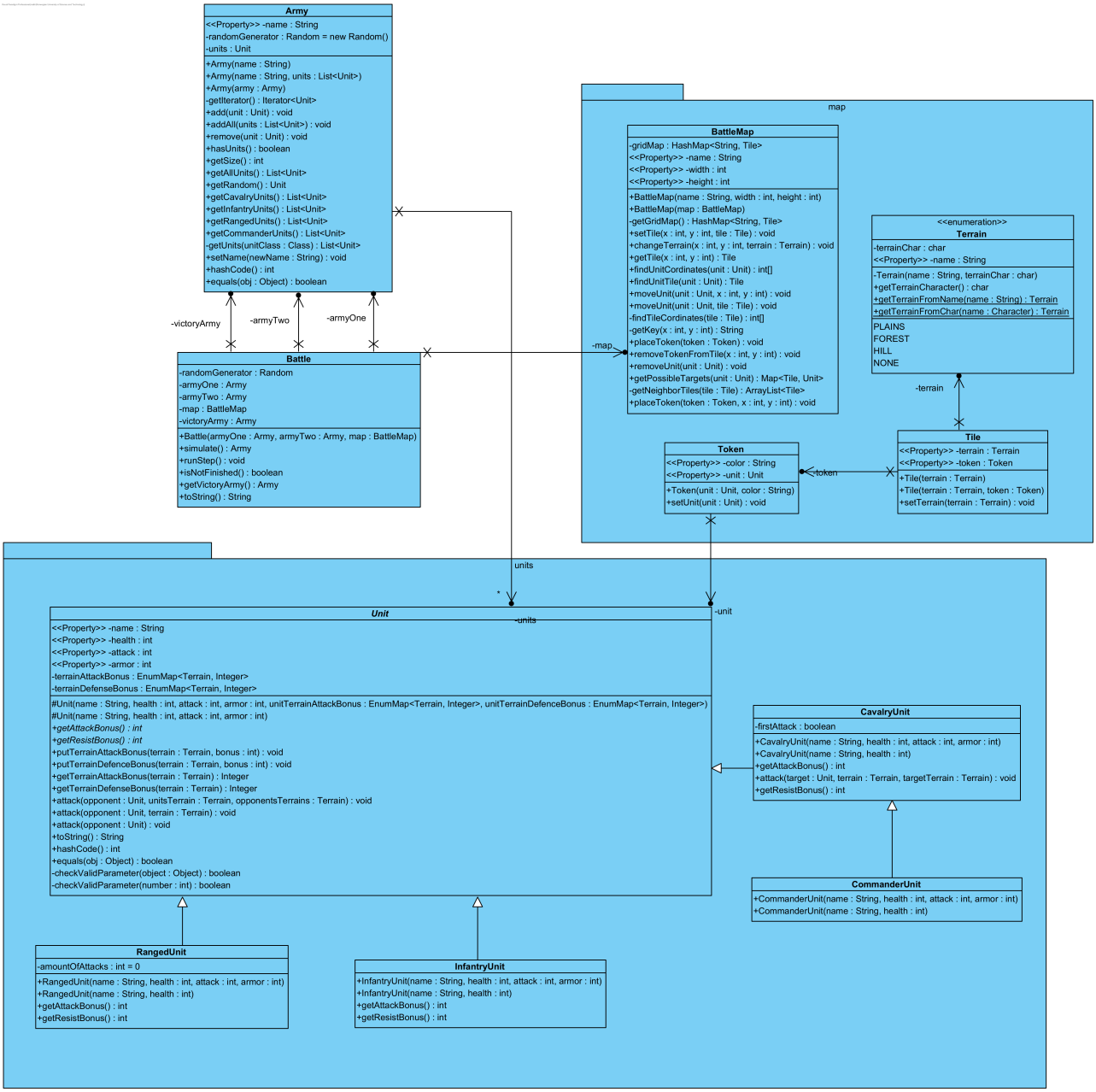


Back-end

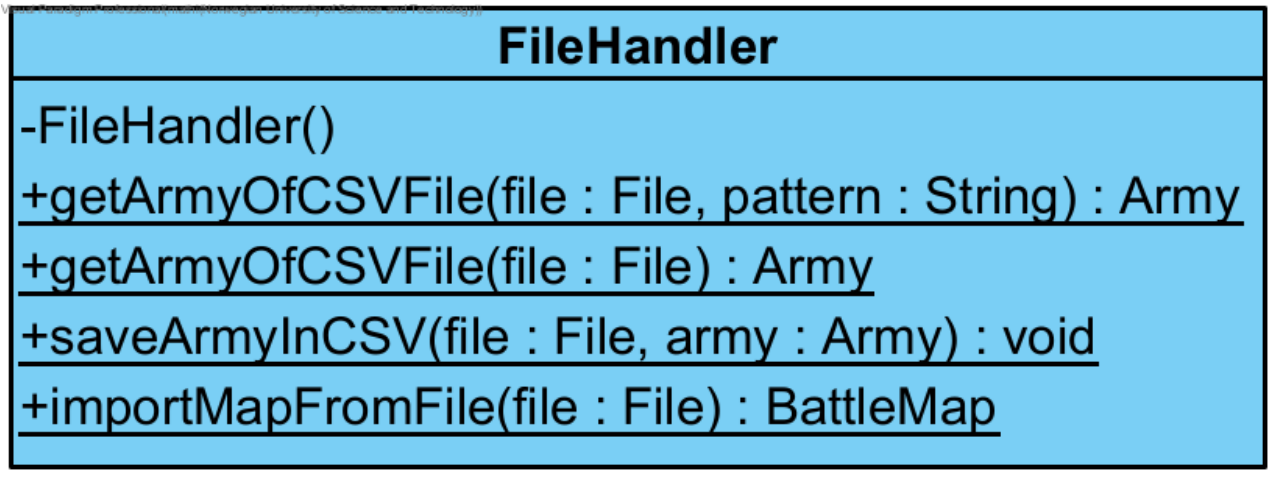
- Factories: This per days date consist of only the unit factory that follows the factory design pattern. It is used for creation of new and copies of units in the code.


UnitFactory
-UnitFactory()
+createUnit(unitType : UnitType, unitName : String, unitHealth : Integer) : Unit
+createUnitList(unitType : UnitType, unitName : String, unitHealth : Integer, amountOfUnit : Integer) : List<Unit>
+copyUnit(unit : Unit) : Unit
<<enumeration>>
UnitType
-name : String
-UnitType(name : String)
+getName() : String
+getUnitTypeFromName(name : String) : UnitType
INFANTRYUNIT
RANGEDUNIT
CAVALRYUNIT
COMMANDERUNIT

- Models: Here lies most of the back-end code/objects like the units, army, battle and map.



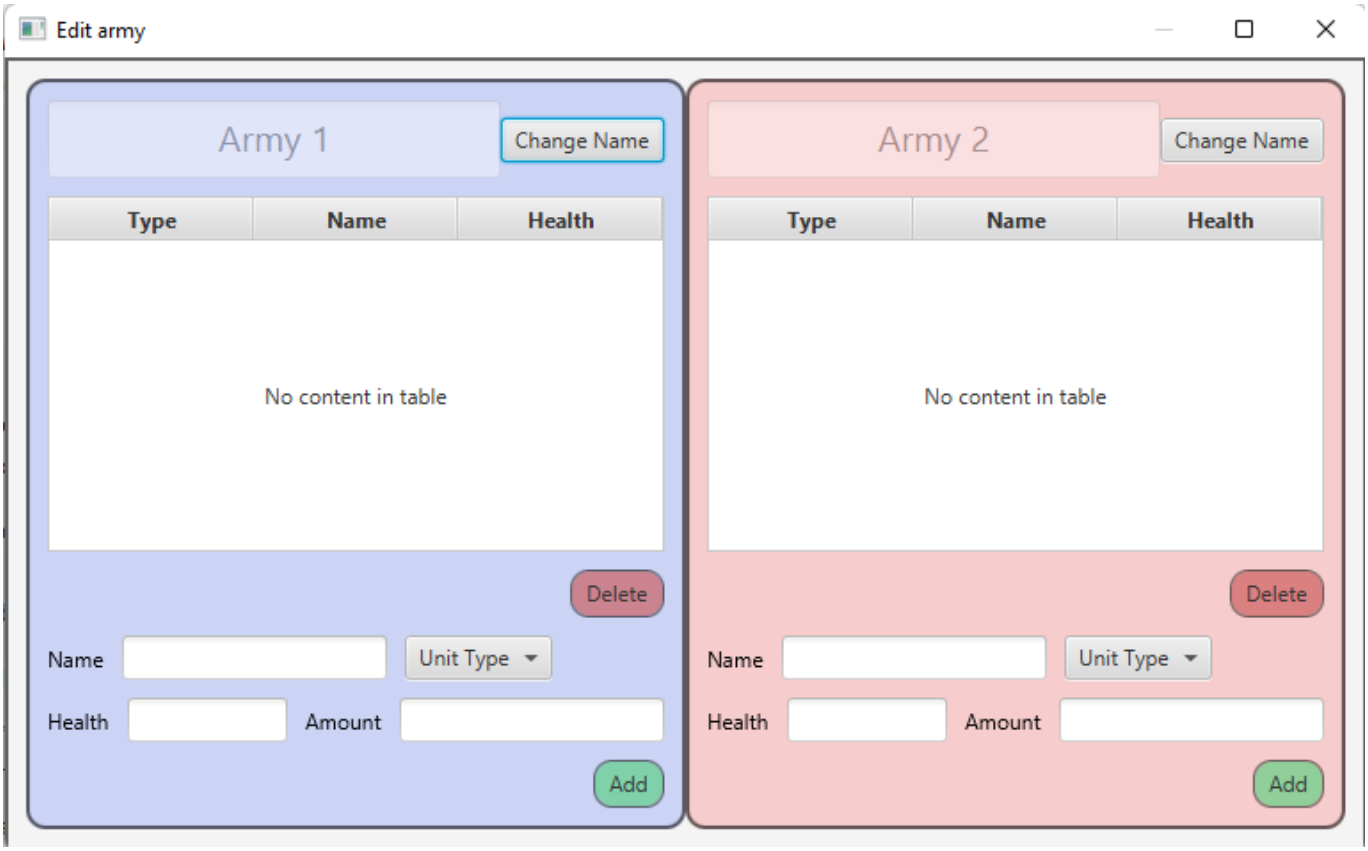
- Handlers: Here lies the different handlers that serves as a mediator to things outside of the application



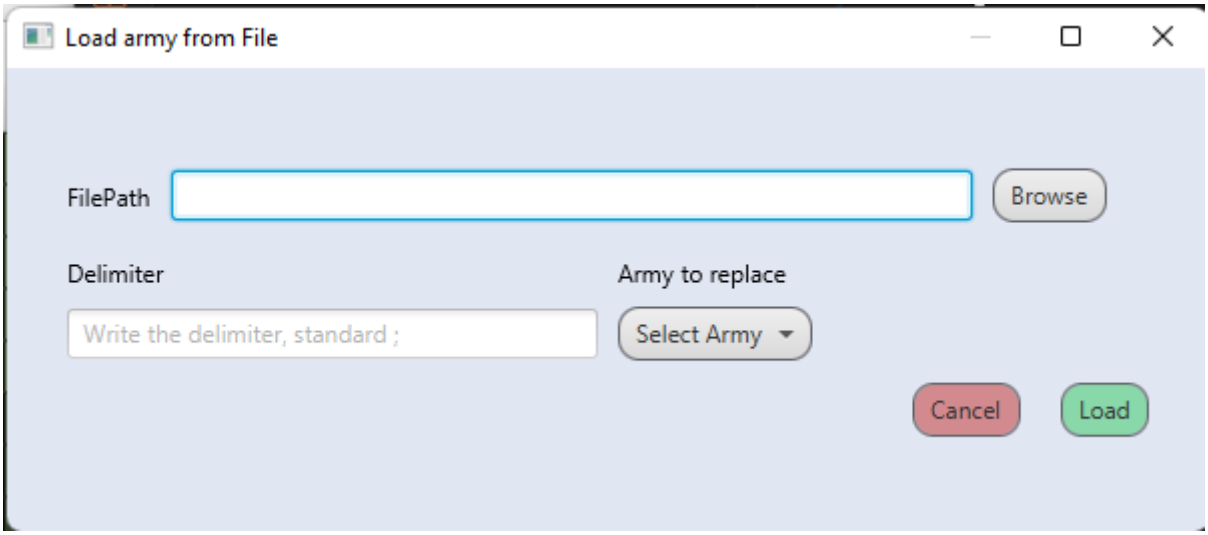
Implementation

Front-End

The front-end is a combination of java, css, png and map txt files. The application loads up FXML files that is linked to a CSS and a java controller file and if necessary creates and places objects. These are objects are mostly imported from the fxmodels package that consist of manipulated javaFX models like TableViews and BorderPane. The reason for why these objects are not just manipulated within the controllers is to ensure more readability within the controller and make it easier to access and/or modify for other controllers. As an example the edit army window uses 2 UnitListTableViews to allow for removing of selected units:



To ensure that we always refers to the same army, the armies is contained within a List of armies that is the size of 2. This makes it so we don't need to manipulate the current object to be exactly like another object, but can just replace it like when we are importing a army for another file.



While the GUI is poorly designed for accepting more then 2 armies because the use of static text elements, it would be entirely possible to increase the ability to add more armies by using the same transfer method between controllers.

Back-End

The back-end is pure java code with no extra packages implemented. This is to ensure that the code is really easy to copy over to another project without other requirements. Many of the tasks given trough out the semester has been solved within the model package. While the implementation of Units, Army and Battle was done relatively early trough the process, many of the classes has been modified to fit new criteria that came later.

Units

Every Unit within the application is a subclass of the Unit superclass. This was a requirement given in Assignment part 1 and is made this way since every unit has a default methods like attack and variables like health, attack, armor and etc. This is to make refactoring easier later when we were going to implement terrain.

The units also have a UnitFactory created with the requirement given in part 3. With its implementation a ENUM called UnitType was created. This was to hinder the use of String or another type of object as it provided static "variables" of the different unitTypes. This also means that every time a new Unit is introduced, the type needs to be created in the unitFactory. This UnitFactory was also given one extra method: CopyUnit(Unit unit).

```
    /**
     * Copies a unit
     * @param unit the unit to copy
     * @return a copy of the unit
     */
    public static Unit copyUnit(Unit unit) {
        if (unit == null) {
            throw new IllegalArgumentException("Unit can't be null");
        }
        return
createUnit(UnitType.getUnitTypeFromName(unit.getClass().getSimpleName()) ,
unit.getName(), unit.getHealth());
    }
```

The reason this is within the factory is because it allowed the units to be copied without specifically adding a clone method or a clone constructor to every subclass. This makes it so the Unit class is easier to implement functionality or refactor.

Army

Within the assignment part 1 there was the requirement of a army class that will hold every unit. This class was to be used later within battle to simulate a battle between two armies. The army needed to contain a list of units and a name for the army as specified by the requirements. As a part of assignment part 2, the army class was to be implement 4 new methods. Every method was supposed to get every type of unit within the class. As part of this there were created a 5th method to apply for every of this functions: getUnits(Class unitClass).

```
    /**
     * A private function meant to filter out some units
     *
     * @param unitClass the unit class that the person wants to filter
     * @return a list of the specified class of unit
     */
    private List<Unit> getUnits(Class unitClass) {
        return units.stream().filter(unit -> unit.getClass() ==
unitClass).collect(Collectors.toList());
    }
```

This was to allow easier methods to create when a newer UnitClasses was created, and if there were plans to change from the list object it could be easier refactored.

Battle

As the final part on assignment part 1, was the creation of the battle class. This class was going to take 2 armies on the constructor and simulate the battle using the simulate function that would give the victory army. As of the introduction in assignment part 3 with terrain and adding the functionality of a animating the battle this Battle function got bigger. The battle could no longer just simulate and needed to be called step wise to ensure that the application could update between each turn.

Terrain (Map Package)

As part the Assignment part 3, Terrain was introduced as a factor within battle. This also gave me a excuse to work on it as part of a bigger project "BattleMaps".

A BattleMap contains several Tiles that consist of a Token (a "visualized" unit) and a Terrain. The map is always created with the mind of a grid in the form of squares much like a board game. BattleMap ended up as a meditator for battle class to the tile class to allowing for placing, moving and removing units from those.

FileHandler

The FileHandler was a part of assignment part 2, to add functionality for loading and saving a army to or from a CSV file. As the assignment specific used commas for splitting the elements, I found it to be simplistic. While many application saves a CSV with commas, like Excel, it could ruin a import if a name of unit contain several titles, with a splitting at commas like one of the unit within the test.csv: "Petter the almighty, demon slayer, master of mystical magic and gods messenger". While this is a really extreme example it also serves as a valid point to use a more rare character like semicolon.

Process:

Trough the semester I have actively worked with this project when other assignment or subjects had a priority. With the first priority in the project always to finish the assignments given out as it was obligatory. After each assignment I could allow myself to implement methods or features that would fulfill some of my vision of the game.

When working with this project I would often find myself pushing code to master because I found a fix or I had missed something when I last pushed. Big refactoring or implementing of new features would mostly be created as a branch to ensure that I had a working code before deadlines or if I created something I didn't feel satisfied with I could easily return to the main code.

While the Git IssueBoard is a good way to document tasks and other things needed todo, I would mostly code a TODO task within the java code to remember. This was since the project was individual I didn't need a reason to document changes other then commits and pull requests. If the code was a team effort the issues would have worked as notification when a task was finished and give people a general overview on how much was left, was in work or done.

Under the process I had SonarLint installed within my IDE to reinforce good code practice. While it does ensure that the code is "good" it at least provides some help. GitHub Copilot was also installed but was mostly helpful wih documentation as many of it's solutions did not fit within the project.

Reflection

Trough this period I have had many ideas on how to implement or add more functionality into the project. While this has worked in my favor to work with the project, it has also reflected negative towards me with creating documentation and tests. It has also made that some of my work seems "rushed" in the hope of it being finished. This can be mostly been seen in the Map package where I did not have a clear idea how I would want object to interact with each other.

While I have made many choices trough the project, many of them has been created trough instinct or because I was lacking a functionality I wanted. This also means that much of my code does not follow proper design pattern as it should. Much of my code or the model of package has been refactored and been trough a couple iteration before it became this final product, that could have been avoided with proper planning or using more to time to reflect on the code.

Conclusion

I have in conclusion fulfilled with the best of ability the requirements of this project. The project has been both a fun and a good learning experience as it provides some points that I can be better at. Given more time I am sure the I would have added more features within the application. If this task was given with a team in mind, github tasks would have been used more efficiently.