Wargames project

By Beka Bonsa

## Project Background and Description

|  |  |
| --- | --- |
|  | Wargames is a project assigned for first year software engineer students who are taking IDATT2001 (Programming 2 course). The primary purpose of the project is to grade the student’s abilities based on how well the student integrates the technologies and principles covered throughout the course. |

The main task for the Wargames project is to make a war simulation game where armies, each with their own several types of units, can go to war and an army can come out victorious. The project also focuses on implementing tests, design principles and the use of various tools that make the workflow easier.

## Requirements

|  |  |
| --- | --- |
|  | The Wargames application has many features, most of which are stated in the subtasks of the Wargames assignment. Here is a use case diagram conveying the most prominent features in the Wargames application |

Diagram

Description automatically generated

|  |  |
| --- | --- |
|  | A screenshot of the Wargames user interface. |

Graphical user interface, website

Description automatically generated

As shown on the picture above, the user interface includes a menu-bar, selectable terrain types, Army info display and army tables. Placed in the middle of the screen are two troops, facing towards each other and representing their army, who are prepared to charge forward when the simulate battle button is pressed. The surviving troop after a simulation indicates which army was victorious.

Starting from the menu-bar, the user interface includes two different categories for menu options. One of them is File, in here lies an option to create an army file. When this option is clicked, a user is required to type in an army name and select file path right after. If done correctly, an army file with five troops will be generated and saved.

The second menu bar option, Edit, offers an option for the user to remove all troops from one of the armies.

Another crucial feature for wargames is the option to load armies from the user’s computer. The army file must have the extension “.csv” and must have comma separated values contained within. If the forementioned criteria is met, an arbitrary army file can be loaded by clicking on the load army button

Wargames is a battle simulation game and choosing terrain plays a vital role in the outcome of battles. Terrain can be chosen by clicking on one of the three displayed terrain types under the wargames’ logo on the GUI.

The table view is the most important feature for the GUI. After loading an army, the units in the army are displayed on one of the table views. It is possible for the user to modify and change the type, name, and health of a unit directly from the table view by double clicking a unit. There is also an option to add or remove a unit of a specific army by selecting the unit, then right clicking on the table view, to then choose “Add new unit” or “Remove unit”.

The save army button which is present on both sides of the screen is responsible for saving the current state of the army by overwriting the loaded file with the modifications added to it by the user.

The Simulate battle button highlighted in green is the button that kick starts the simulation and requires both armies to have at least a single unit.

The reset button is intended to reset all changes that have been made by the user and reset everything to the first state when the armies were first loaded.

## Design

|  |  |
| --- | --- |
|  | Described in this section are the design principles and design patterns used for the Wargames application |

The following are the main design principles/patterns used for Wargames.

* Single responsibility principle
* Open-closed principle
* Factory design pattern

Single responsibility principle is one the five SOLID principles for software development and is about making a class or a method serve a single purpose. This ensures the context and purpose of a class is not obscure or ambiguous and makes it easier to read and understand each class. When applying this principle, the cohesion of each module in the program will therefore increase. Increasing cohesion is the first step towards writing readable, elegant, and reusable code.

The Open/Closed principle is about extendibility and how easy it is to add new features without rewriting any previous work. An effective use of this principle results in fewer bugs when adding new features to a program and promotes efficiency for future extensions. For a product that is continually updated and maintained, the Open/Closed principle becomes critical to implement.

Factory design pattern is a creational pattern used for dealing with problems that arise when creating objects without having to specify the class of the object that will be created. A factory can be used to create an instance of a sub-class at runtime, a design pattern that came in handy when editing unit-type of units present in the tables of the GUI.

The GUI developed also followed common standards for front-end design. One of which was creating a responsive flow for user interactions. If we look at how hovering the cursor over certain UI elements, we see that some of the elements respond by resizing the text contained within or the element itself. This helps distinguish which buttons/elements are interactable and which are not.

One of the most key features a GUI should offer is, proper scalability. A program that offers no scalability harms the user’s overall comfort and experience. The biggest drawback of lacking scalability is the restrictions it could have for users with different sized monitors/screens. Ensuring scalability at an early stage helps the program reach a higher device coverage and gives the user the freedom to scale the application to their needs.

One of the other minor details present in the GUI is the coloring of buttons and sections. The most important button simulate battle is colored green to immediately capture the user’s attention. The simulate battle’s adversary, namely the reset button, is colored red to indicate that it is the opposite of simulating a battle. Since green is normally associated with safety the user is most likely to click on simulate battle (colored green) effortlessly compared to when clicking reset battle which is colored red (indicating danger, here loss of information). Aside from the two colors red and green, the GUI also feature a shade of orange to highlight important sections for the user. The table view is an example of that.

Animations, when implemented correctly, can be used to further immerse the user, and give a dynamic and lively feeling to the GUI. Lack of animations, especially in a simulation game, can make the whole experience dull. To make the simulating experience more engaging, animations for the troops representing their armies, were added. This proved to be a quite challenging task given that stage resizing (scalability) was of higher priority and applying both is troublesome.

A huge factor that contributes to the usability of the GUI is displaying errors and making sure the user knows what the program does not allow. For example, simulating a battle without specifying an army will result in a warning message that says either army-1 or army-2 has no units. It does this by creating a pop-up window with the warning text that waits for the user to click on ok or close out of the pop-up. This way, the user is forced to know what has gone wrong and must complete the requirements before proceeding.

|  |  |
| --- | --- |
| Badge 1 with solid fill  Badge with solid fill | Class diagrams  **Figure-1** shows the classes within the “units” package and their relationship.  **Figure-2** shows the relationship between all the classes in wargames, excluding the test classes.  UnitType and Terrain were the two enums used for simplifying unit types and terrain. |

## Implementations

|  |  |
| --- | --- |
|  | Figure-1 The project structure for Wargames |

Graphical user interface, text

Description automatically generated

As shown in this figure, the folder structure of Wargames follows a Maven standard directory layout. Furthermore, the use of packages within the subdirectories of the project folder makes it easier to navigate and organize different classes. The test folder is also packaged in the same way the src/main folder is, this way it is easier to see which test corresponds to what class and vice versa.

|  |  |
| --- | --- |
|  | The following is an example of the single responsibility principle applied for the classes in the file management package. A class diagram of the classes contained within the package is shown below |

A screenshot of a video game

Description automatically generatedArmy tools is the parent class, and the file reader and saver are separated.

|  |  |
| --- | --- |
|  | The open/closed principle was used for creating distinct types of units |

Creating a new unit is as simple as inheriting the unit class and adding new attributes/methods to the newly created class. An example of this could be how the creation of infantry, ranged, cavalry and commander unit, did not require the modification of the parent Unit class. This makes creating new unit types much easier and does not require rewriting existing code.

|  |  |
| --- | --- |
|  | Factory design pattern is used in Wargames for converting unit-type at application run-time |

The following is the reference of the table column defined in the FXML file

@FXML private TableColumn<Unit, String> unitType

Highlighted in blue is the use of the UnitFactory’s unit maker method, it gets executed when a player commits an edit of a unit’s type in the GUI. The unit maker returns any one of the four units in wargames.

Text

Description automatically generated

|  |  |
| --- | --- |
|  | FXML node structure used for proper scaling. Here is a screenshot taken from scene builder |

*Text

Description automatically generated with medium confidenceA Border Pane is the root node.*

*V-Boxes under H-Boxes have*

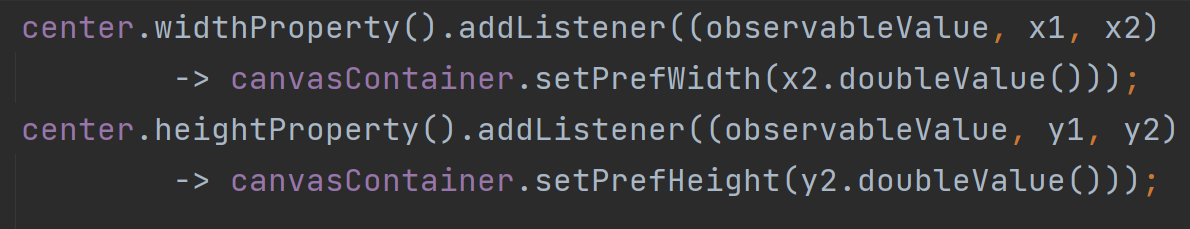
*h-grow set to always.*

|  |  |
| --- | --- |
|  | Solution for responsive/scalable animation |

Graphical user interface, text, application

Description automatically generated

Vbox fxid: center

AnchorPane fxid: canvasContainer

Scales the AnchorPane(canvasContainer) according to the parent (Vbox center).

## Process

|  |  |
| --- | --- |
|  | This section discusses the process of making Wargames. |

Wargames is a project that requires consistency and work discipline, the following tools made the development process much faster and improved the quality of Wargames overall.

Javadoc, Java documentation tool, made it easier to document my code and to get back into the project after a break.

Git, a version control software for tracking changes in files and folders, was used through the entirety of the project. It helped create a sense of safety knowing that I could revert to previous commits if something was to go wrong. Such situations often arose and were not much of a problem since I committed with every slight change made to wargames and I could revert to any one of those commits as desired.

Maven, a build automation tool for java programs, was used to download and manage dependencies that were crucial for this project, such as Junit dependency (covered down below).

Junit, a unit testing framework for Java, was an integral tool throughout the entire project. All major features included in wargames were thoroughly evaluated with the help of Junit tests. This was the safest way to assure myself that the code I have written still works after updating previous code and when adding new features. Testing before committing in Git has minimized the chances of creating bigger problems later in the process.

IntelliJ, an IDE for Java, also helped throughout this project. Its warnings, smart error detection and support for CSS highlighting (ultimate edition), made it easier to solve most of the problems myself without having to look for resources online.

## Reflection

Most of the time used working on this project was for research, looking for ways to create the most efficient solutions for the subtasks in wargames. The most challenging one to research about was JavaFX related topics. There were no clear standards and the content found in most places were not suited for big projects.

If I were to do this project again with the knowledge I have now, I would have implemented multiple scenes for the GUI and worked on more unit types.

## Conclusion

The Wargames project covered many topics, such as unit testing, design principles/patterns, project structuring, File management, and creating a functional GUI. With 80% entire curriculum for the programming course being evaluated on one project, the Wargames project incentivizes the use of best practices and teaches a valuable lesson in how to create a robust and functional program.