Wargames project report

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 different 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 important 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. There is also a victory announcer that pops up on the screen after a battle’s completion.

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 that will be associated with the army file.

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 an important 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 user interface.

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 a unit to a specific army by right clicking the corresponding table view and choosing “Add new unit” to then modify its attributes accordingly.

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 for each class, which helps with
* 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 have a singular purpose. This ensures the context and purpose for 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 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. A good 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 important 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 it’s 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 possible 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.

When implemented correctly, animations 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 somewhat 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) had to be considered for the animations.

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.

The points mentioned above have all helped make Wargames a more user-friendly, maintainable, and extendable project.

|  |  |
| --- | --- |
| Badge 1 with solid fill  Badge with solid fill | Diagram  Description automatically generated sad  Class diagrams  Figure-1 shows the classes within the “units” package and their relationship. These are the unit-types offered by wargames.  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

|  |  |
| --- | --- |
|  | The following is 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 how the open/closed principle was used |

## Process

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

The process of working with wargames was like how I normally work with other projects. I try to be consistent with work hours and add Javadoc to every new peace added along the way. There were times where other classes had to be prioritized and I had to postpone my workhours for wargames. Coming back from such breaks can be quite challenging as the code I have written feels unfamiliar, but the Javadoc helped at getting me back into the project again.

Git, a version control software for tracking changes in files and folders, was used through out the entire project’s lifetime. It helped create a sense of safety knowing that I could revert to previous commits if something was to go terribly wrong. Such situations often arose and were not much of a problem since I committed with every little 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 tested 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 a lot through out 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. IntelliJ is the best IDE I have ever used and there is no doubt that it has saved me countless hours.

As an extra precaution, I had the project directly synced with OneDrive. This made it possible to commit a previous day’s work on a new computer, without worrying about having different versions of the project. This was important for me since I forget to commit and push to the central repo on GitLab before leaving and try to work on the same project on a different computer.

## 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 of low quality (Bad solutions for big projects).

## Conclusion