Swamp Wars!
Software Development 3
Coursework

SET09101
40056761

1 Introduction

Brief The coursework specification gave students the task of developing a solution for the following problem statement:
Using design patterns, threads and a GUI, you are to write the Java code for a game involving an Ogre and some ogre enemies in a swamp.
The specific design patterns requested were the Factory, Observer, Strategy and Command patterns. The openness of this specification can be seen as a double edged sword - while it leaves a lot of options open in terms of creativity, the lack of a specific problem may cause a developer to be unsure of how to progress.

Method The program was created with a ground-up methodology, implementing the game-logic in its entirety before creation of a user interface. The reason for this was two-fold - it prevented any game-logic from being accidentally tied to the UI (poor object-orientated methodology), and meant that different graphical user interfaces could easily be imported, much like in the real world. For example, an engineer could develop the back-end code and a designer would build the user-end interface.
The program presented in this report can potentially be run with a choice of two interfaces: a Graphical User Interface - which employs windows and images to display game information - or a Text based User Interface - which runs in a computers command line and outputs all information as plain text. Both UIs have the same functionality, allowing a user to play the same game in two different mediums.

Variable Grid Size In order to implement a variable, asymmetrical grid size for the game, each actor keeps track of its own current position in the form of an X and Y coordinate. The actors move-method does not allow them to move beyond a limit, which can be changed depending on the size of the grid.

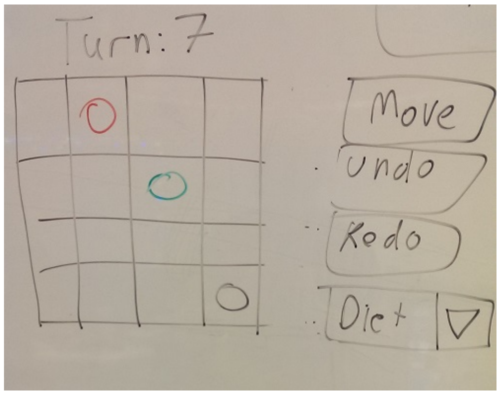


Figure 1: ***A rough plan of a possible GUI layout.***

2 Patterns and Threads

Factory Pattern The Factory Pattern allows a developer to generate an object, without specifying its class. The coursework specification suggested that at the end of each turn there is a 1:3 chance of a new, random enemy entering the swamp. Use of the factory pattern allows for this functionality, as each type of enemy is derived from a single base-class.
If the game was to operate without using the factory pattern, it would still be possible to generate random types of enemy through use of polymorphism. However, the code required would have to be embedded in the core game logic, rather than in a small class of its own.

Strategy Pattern According to the specification, an Ogre can have one of three diets at any time. The type of diet which is currently in use defines what types of food the Ogre can eat. Employing the Strategy Pattern allows an object to change its behaviour during runtime - a perfect solution to this requirement.
The Strategy Pattern allows for simple implementation of changeable runtime behaviour, but it would still be possible to implement without use of a pattern. By writing a method for each form of behaviour and providing booleans to determine which method would be called, the same effect could be achieved.

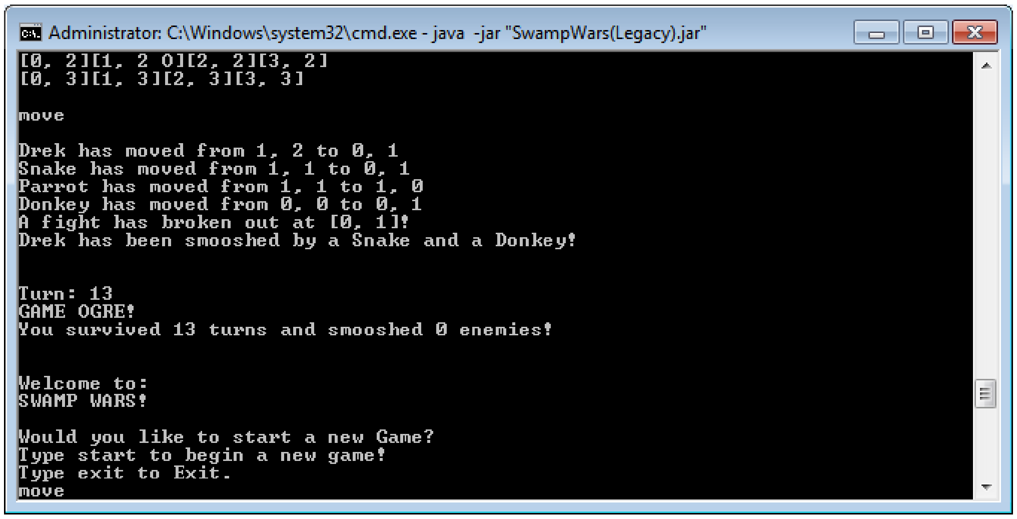
Command Pattern The Command Pattern allows an objects method to be executed by another class, or at a later time. In the game featured, the Command Pattern is used to generate a list of move commands for both enemies and the player, executing them all at once.
While the limited scope of this project may render this pattern somewhat unnecessary, there is no denying its usefulness. If there was a significantly large number of actors, or the move method was more complex, this pattern would certainly provide a runtime speed up.

Observer Pattern While the documented program does not utilise the Observer Pattern, the utility that it provides when implemented correctly is highly efficient.
Ideally, this design pattern could have been used to allow the GUI to observe the game-state and update it accordingly when an event took place. While this may appear to be the best place to implement the pattern, in practise complexity was significantly reduced by having the GUI pull data from the Game-Control class when required.

Threads Even though the final product was a turn based game, it was still recommended that Multi-Threading should be implemented. By using multiple threads, it is possible to run several operation in parallel, allowing for faster runtime execution.
By threading the generation of commands, the game can create a move-command for each entity on the board in parallel, leading to a small, yet significantly noticeable enhancement in runtime.

3 GUI

Start Screen The Start Screen is presented to the player upon running the application. Here a user can input a size for the grid they want, as well as a name for their Ogre. The default name Drek Is already entered.
The Start Screen contains some simple logic to prevent a player entering an empty string as an Ogre name, and also limits the grid size to a number between 4 and 8. An example of the Start Screen can be seen in Figure 3.


Administrator: C:\Windows\system32\cmd.exe - java -jar "SwampWars(Legacy).jar"

```
[0, 2][1, 2 0][2, 2][3, 2]
[0, 3][1, 3][2, 3][3, 3]

move

Drek has moved from 1, 2 to 0, 1
Snake has moved from 1, 1 to 0, 1
Parrot has moved from 1, 1 to 1, 0
Donkey has moved from 0, 0 to 0, 1
A fight has broken out at [0, 1]!
Drek has been smooshed by a Snake and a Donkey!


Turn: 13
GAME OGRE!
You survived 13 turns and smooshed 0 enemies!


Welcome to:
SWAMP WARS!

Would you like to start a new Game?
Type start to begin a new game!
Type exit to Exit.
move
```

Figure 2: *The Game Over screen, as displayed in the Legacy Mode, in the Command-Line.*

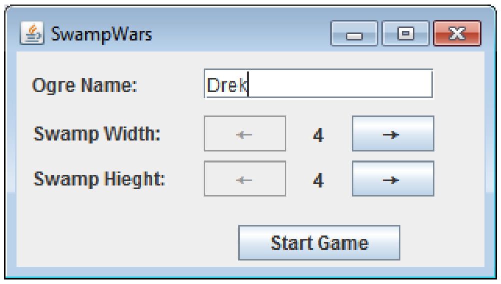


Figure 3: *Start screen presented to the player upon launching game.*

Game Screen Upon finalizing a grid size and Ogre name by clicking the Start-Game button, the Game Screen will be instantiated and shown. As well as the actual game grid, the Game Screen also features three buttons move, undo and redo, as well as a combo box containing various Ogre diets. Both turn and point counters are also present in this window.
While simple in design, the UI displays all information a user would require in a way that is easy to understand. Figure 4 is an example of the Game Screen.

Legacy Mode (Command Line) The game can also be played in a fully text-based mode, in a computers command-line console. While this mode was originally used to display information before a GUI was built, it was further implemented and included in the final version of the project.
While harder to display actors, the Text Based UI has the added benefit of being able to print out what happened each turn, allowing a user to see where each enemy moved from, as well as their final destination. An example of the Legacy mode can be seen in Figure 2.

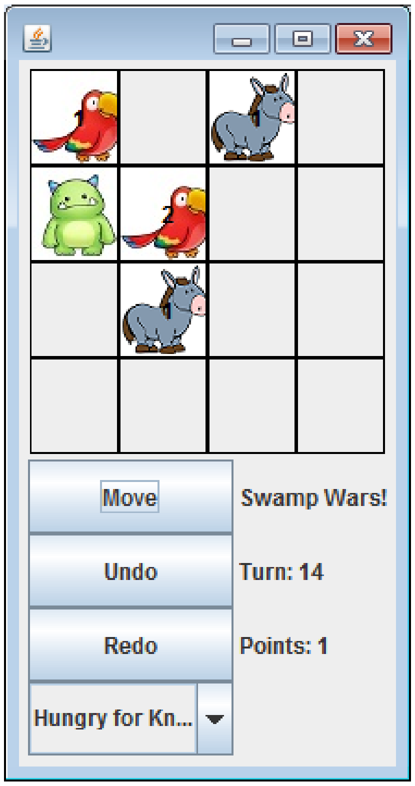


Figure 4: *The game screen after several turns.*

4 Conclusion

Summary The production of a game, with such an open-ended specification was an enjoyable challenge, allowing for creative and imaginative implementation of programming patterns and methods.
If this assignment was to be attempted again in the future, more time would be spent on developing the UI, as well as implementing more design patterns in order to create a more concise, modular game.

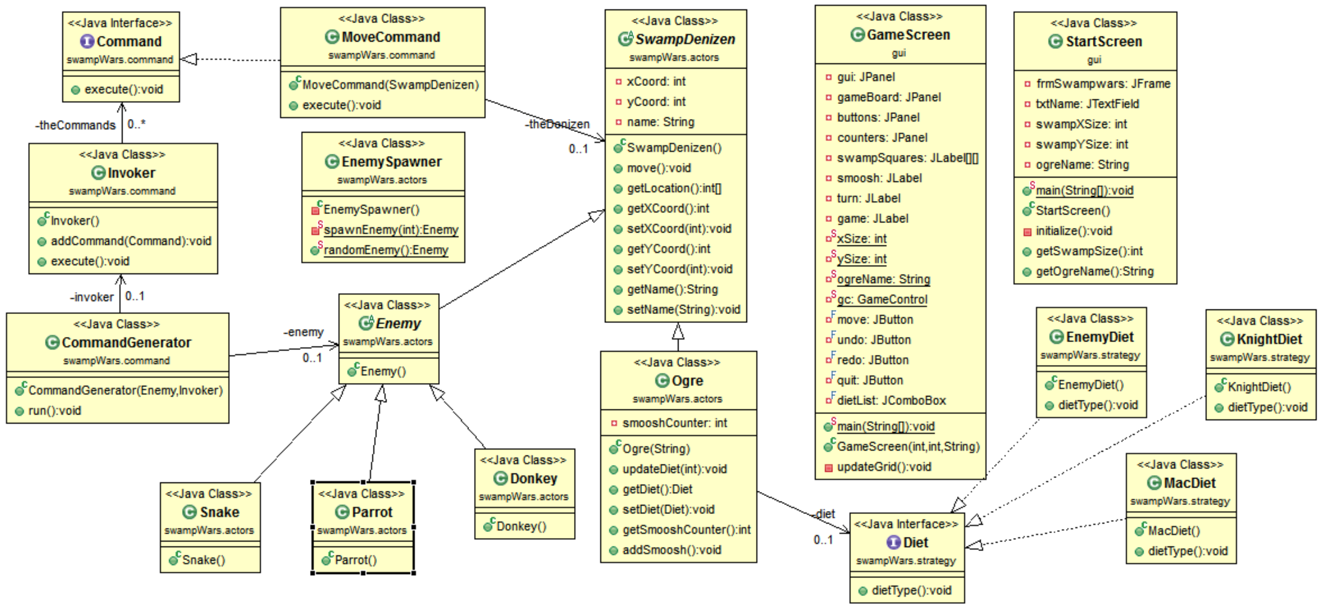


Figure 5: ***The Class Diagram of Swamp Wars.***