Team AdjacentSeats APCS pd 5

Jonathan Quang, Noah Tang, Thomas Lin 1/05/2017

Final Project Proposal

Turn Based Military Sim

Overview

Our project proposal is a turn based player-versus-player military simulation. Each user takes the place of a general or admiral that issues orders to units on a battlefield (a grid of ASCII characters that is printed out based on information stored in a 2d array). Every turn, the player types in a command and a character will perform the action accordingly. Examples of commands would be north, south, attack, and charge. You can only issue one command per turn. Different units have different stats (such as strength). Battlefields may vary in terrain composition, with wooded, hilly, or flat land.

Each ASCII character will represent a group of units. This way there won't be a large number of characters on screen, providing clarity and avoiding overcomplexity. You win if your side successfully eliminates the enemy units before they do it to you.

Several pre-programmed scenarios from throughout history will be included. For example, the Battle of Hastings would see the user controlling either William or Harold, with units placed appropriately on the field, and hilly terrain for the English army to occupy, as they did during the real event.

Concepts/Materials Used

Other classes we will use include Keyboard, and a unit superclass, similar to YoRPG's Monster and Character classes. Keyboard.java will be utilized for reading terminal user inputs. This is used to determine the command the player will issue, a crucial part to playing the simulation.

The battlefield will be created using a large two dimensional array. In this array we plan to store objects that will represent various terrain features.

A number of objects will also be instantiated for each unit on the field. The units will have a coordinate location on the battlefield. When the screen is updated, the program will print the string representations of each terrain feature in the array, unless there is a unit occupying the area, in which case the unit's string representation will be printed. As the new battlefield string is constructed, a for loop will have to iterate through both an array of Terrain features and Units. In the case of the latter, the structure used will be similar to a linear search in that it is traversing the array in search of Units. If it finds one, the Unit will be added to the string-to-print.

In summary, our final project will make use of:

- Superclasses/subclasses
- Inheritance, Polymorphism
- Multiple data types, including arrays
- Iterables and Iterating, looping structures
- Abstract classes, such as the superclasses Unit and Terrain.

```
For example:
2D Array battlefield:
```

forest	plains	plains
plains	plains	plains

```
Unit Objects:
Infantry, at 0,1.

Key (not finalized):
# = Infantry unit
. = plains
, = forest

Printed to Terminal:
  123
A ,#.
B ...
```

Important Classes and Methods

Unit Class

Each different type of unit will have a class specifying its capabilities and properties, and each of these classes will be subclasses of the Unit.

Unit will have coordinates, to save its location on the map, and a 'strength' value which indicates roughly the number of soldiers in the unit. It must have accessor and mutator methods for these variables. It must also have an attack method, which is used to reduce an enemy unit's strength by a number proportional to its own, and movement methods, which allow it to change its coordinates. An array of Units with width and height corresponding to the Terrain array will allow us to easily identify Units based on their location.

Terrain Class

The battlefield will be represented in the program by a 2D array of Terrain objects. The Terrain class must have a name and a single-character symbol to be printed to the terminal. The move methods of Unit objects may specify whether or not they can pass through certain types of terrain. Lakes, for example, should not be crossed by cavalry, and hills should not be crossed by battleships.

Woo Class

The Woo Class will be executable by java, and will include the main method which organizes events and calls appropriate methods. It will also include some static methods which process user input. For example, if a player enters “A7 north”, the program must translate ‘A7’ into integers 0 and 6, and then call the movement method of the unit occupying the square at (0,6).

A method using Keyboard.java will print out a start menu which will allow the player to choose what mode they would like to run the simulator in. There will be a few modes replicating famous historical battles and one random. Choosing the random mode means terrain and units are not pre-set.

There will be a series of methods that will either generate a pre-programmed simulation with units and terrain representative of a specific historical battle or a new, random simulation. Methods will be included that translate each battle’s stored information and print it as a string to the terminal.

There will be an info method that can be called by user input via Terminal. It will print out the commands, descriptions of what they do, as well as a guide to what unit is represented by which specific ASCII character.

A stats method that functions similar to the info method will print out your units' characteristic values.

User Experience (UX)

The program will run on the default terminal size of 80 x 24. Users will be greeted with a start menu that presents several options: Play a preset scenario, play a random game, and view the game 'manual'. The manual may be accessed at any time during a game, and will print a page of notes on how to use the program, as well as a key to deciphering the printed battlefield representation shown during each turn.

Inside a battle, the top of the terminal window will always contain a visual representation of the battlefield grid. The bottom will include a space for the user to type and send commands to the battlefield, just below an indicator of the turn (ie "Player 1's turn" or "Player 2's turn"). After any command which updates or changes the battlefield, roughly 30 newlines will be printed, to clear the screen. The battlefield grid, now updated, will then be re-printed at the top, and the command-writing space below it. The interface will specify that it is now a different player's turn.

Stretches

There will be room for stretching in this project, provided that we have time to first comfortably implement and test all of the features mentioned above. Reading a text file would be useful, so that we could create a way to ‘save’ a battle and come back to it later, or perhaps to provide an easier way to add historical scenarios.

We would also be interested in trying to implement color text in the terminal, if possible. Colors might be a good way to visually distinguish different units and terrain features, and also might increase the program’s visual appeal.