SYSC 3110 Final Project

Team Group

Author: Baillie Noell

Data Structures Explanation:

1 Class: Map

1.1 countries: LinkedList<Country>

We chose a LinkedList because they are more efficient at addition and deletion than an ArrayList. The operations needed to loop over a LinkedList are also only slightly less efficient then with an ArrayList. The countries in the list all get added one by one, shuffled, and then removed one by one. With so many countries we decided that our data structure needed to be more efficient at additions and deletions as that is what it is primarily used for.

1.2 continents: ArrayList<Continents>

The continents in the map are added once and never removed. The continent list is used primarily for retrieval operations which are more efficient with an ArrayList. The less efficient adding is insignificant as there are only 6 continents.

1.3 final String: CONSTANTS (for each country)

Instead of creating an individual global variable for each country we chose to use constant strings for each of the country names, and to create the countries as they are being added to the countries LinkedList. This reduces the creation of “useless” objects, and also removes using “phantom strings” as the country names, as the names will always be the same and will not change throughout the game.

2 Class: Player

2.1 countries: LinkedList<Country>

Each player holds a collection of the countries that they occupy. We chose a LinkedList again as during gameplay countries can change which player occupies them, and therefore will be added and deleted from different players country lists often. LinkedLists are more efficient at addition and deletion, while the lists are only looped over when printing the state of the map and listing which countries are occupied by which player.

2.2 eliminated: Boolean

We considered just removing a player from the Game list of players when they were eliminated but decided on a Boolean instead for two reasons. One, it will give us the ability to include an option during and at the end of the game to restart or start a new game with the same players, without them all having to be added again. Two, when we develop the GUI in milestone 2 this will allow us to display stats and to display which players have been eliminated.

3 Class: Game

3.1 players: ArrayList<Players>

We needed to be able to store all the players in the game. We chose an ArrayList as the players are added at the beginning of the game, and then stay in the game until the very end, as we decided not to delete them when they are eliminated. Most of the operations involved with players are therefore retrievals, which is more efficient to do using an ArrayList than with a LinkedList.

3.2 currentPlayer: Player

When storing which player is currently playing we considered two options, storing the index of the player in the players ArrayList, or storing a player object. We chose the latter because it avoids unnecessary method calls that would come from storing only the index. If we used the index, instead of being able to call currentPlayer.method() to use the method of that player, we would instead need to call players.get(index).method() which gives us smelly code under the Law of Demeter.

4 Class: Country

4.1 neighbors: ArrayList<Country>

A country can only be attacked from its neighbor, so we decided to have each country store their own neighbors. The neighbors are added at the beginning of the game and never get deleted, as the neighbors of a country do not change. Therefore, operations on a countries neighbor list are always looping through the list to check if the chosen country to attack is a neighbor. This is more efficient with an ArrayList.

4.2 troops: int

Though troops technically belong to players, we chose to have countries store them. This is because countries are always owned by a player, and a player will not have troops that are not on a country that they own, as well as a player will always have at least one troop on every country that they own. By storing the troops on the country when we attack, we do not need to care about which players are involved in the attack unless an attack is won and a country changes hands. During the attack method we can focus on the countries, and then determine later which players to move a country between only if an attack is won. If an attack in lost then we only need to remove troops from the countries involved in the attack.

5 Class: Continent

5.1 countries: ArrayList<Country>

Each continent stores a list of the countries in it, which is used to give out extra reinforcements when a player owns an entire continent. A continent will never lose a country, so the list never has deletions. The list is mainly used to loop through and check if a player owns all countries in a continent, which is more efficient to do with an ArrayList.

5.2 final int: reinforcements

Each continent stores the number of extra reinforcements a player gets when they own the entire continent. This number will never change, so it is designated as final.

6 Class: CommandWords

6.1 validCommands: String[]

The CommandWords class holds an array of Strings that contains the valid command words for the game. The command words are stored as Strings so that they can be easily checked for validity using the .equals() method. They are stored in an array to facilitate easy modification, as well as easily being able to search through the entire array to confirm valid commands.

7 Class: Command

7.1 commandWord: String

The command word is the word which is the “main” word expressed by the user via the command line. If the command word matches one of the valid command words in the CommandWords class, this command is then used by the game to decide the next action to take.

8 Parser

8.1 reader: Scanner

Imported from ‘java.util.Scanner’, the scanner takes input from the command line. This allows the players to type commands into the game via the command line interface. The Parser then takes the commands from the reader and transforms it into command words that can be used by the game.

8.2 commandWords: CommandWords

Uniquely created for this game, the list of command words is used by the parser to determine if the user has typed a valid command. If they have then the command is created and executed accordingly.