Pandemic Game User Report

Maksym Turkot

05/26/2021

Introduction

This document includes a discussion of how I developed a software solution to address aspects outlined in the design document, a user guide explaining how program should be run, a diagram of major components of the program, description of how program's functionality was verified, and a discussion of outcome data generated by the program explaining how it was run.

Software Solution Development

Setup:

The setup of the game board was simulated within the program in the constructor of Game class. All counters were set to their starting values, objects were created and different initializing tasks performed, such as shuffling cards and placing pawns in random cities, and infecting random cities. Each pawn was assigned two random cards from the shuffled deck. Once this was done, the game began.

The map itself is read from an adjacency list stored in map files. Each city has a TreeMap with pointers to other vertices on the game board.

Playing:

When the game started, each iteration was a sequence of pawn actions and map updates. Each pawn would first perform four actions. It would check if it had 5 cards with cities of the same color. If it did, it would cure the disease of that color, and continue. After that, each pawn checks if the current city has infections in it. If so, it will cure the disease in that city as long as it has turns left. If no diseases were found in current city, the pawn would move to a random city neighboring with the current one.

Once pawn used its four actions, it would draw two cards from the deck. If it ended up with more than 7 cards, excess cards had to be discarded. Now, the card discarding algorithm was actually an important one. It was crucial for pawns to try and collect 5 cards of the same color, otherwise they would never be able to cure any disease and would stand no chance at winning the game. When prompted to discard cards, pawn would check if it had cards of an already cured disease. These are useless at this point, so it would discard them first. If not, pawn would count how many cards of each color it had, and discard cards of a color with the lowest number. That way each pawn would try to get 5 cards of the same color as quickly as possible. As discussed later, this allowed pawns to cure in some cases 3 diseases, which proves that the algorithm works.

Endgame:

The loop of turns continues until Game detects one of the following conditions. If there are no cards left in the city deck, pawns ran out of time and they lose. If the number of cubes in stock becomes negative, the disease has spread too much and pawns lose. If all four diseases were cured, pawns win.

To manage these outcomes an endGame method was created, which prints data and results of the game into an output file.

User Guide

PROGRAM STRUCTURE:
In the project folder user may find the “data” folder. In it, there is a “maps” folder, containing adjacency list files, “logs” folder, containing program run logs, and “outcomes” folder, containing data about how game ended and relevant statistics.

CONFIGURING AND RUNNING THE PROGRAM:
In order to configure the program:
User may change configuration files in the “maps” folder described above. The format of map file should be as follows: each line should start with a name of a city, followed by a semicolon, followed by a comma separated names of neighboring cities.

In order to run the program:
1. Run the “package.bluej” file.
2. Right-click the “Controller” class.
3. Click on the “void runGames()” method.
4. The program runs, and once blue bar in the bottom-right disappears, generated log and outcome files may be found in corresponding folders described above.

Project Components and Diagram

COMPONENTS:

1. Class Controller

This class loads the maps and runs each configuration.

2. Class Game

This class runs the game itself, managing underlying tasks such as card shuffling and distribution, city infection, and keeping track of all counters and objects.

3. Class City

This class constructs a city (vertex) of a map, keeps track of all relevant information.

3. Class Pawn

This class constructs an individual pawn with its treating, moving, and action taking logic.

4. Class MapFileReader

This class manages data retrieval from map files.

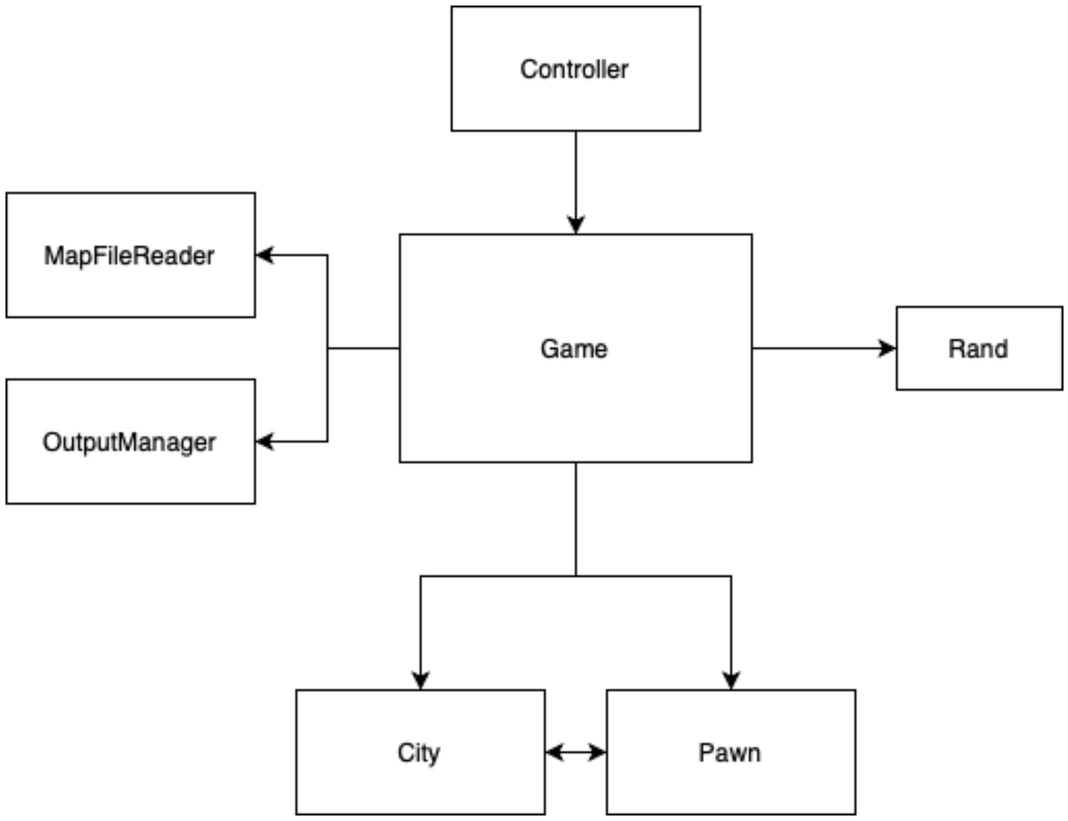
4. Class OutputManager

This class manages output file creation.

5. Class Random

This class generates random numbers to be used throughout the program.

DIAGRAM:



Verification of Program’s Functionality

Majority of testing was carried out through review of log files and checking if cities were read in from the file, if pawns moved from city to city, if diseases were cured, and if infections were treated.

There were also unit tests performed to verify key components of the program, such as vertex output, dealing cards, infecting cities, and treating those infections.

Data Analysis

When looking at the outcome data, we can see that pawns lost in all cases for the same reason: they ran out of player cards. This means that under current algorithm their performance is limited. Pawns already are able to treat diseases, as we see from the data generated. In some cases pawns cure 3 diseases, which is good progress. In order to win, pawns would need to be able to cooperate by splitting card colors each pawn is looking for and exchanging cards. The algorithm for treating diseases is also limited, since pawns move randomly, treating infections only if they happen to be in an infected city. However, under current conditions with an absence of outbreaks, infections are not the reason for pawn's losses.