Homework 2: Legend of SeePlusia Reloaded

CS 224 Object Oriented Programming and Design Methodologies
Habib University
Fall 2019

Due: 18h on Monday, 30 September

The object oriented approach advocates encapsulation of related information in a class. A program proceeds by creating instances of the classes and having the instances interact with each other.

Object oriented *design* refers to deciding which information to group together and how. Alternately, it refers to deciding the classes for your program and their attributes (characteristics) and methods (behavior). This is not so much a science as it is an art—there are many possible designs for a given problem and all of them are *correct*.

In this assignment, we will redesign Legend of SeePlusia in an object oriented manner. We will encapsulate all information related to game logic in a `Game` class. Information related to a location will be represented in a `Location` class. We will model the information related to traveling from one location to another in a `Road` class. We will implement a `Map` class to model a map consisting of `Location` and `Road` objects. Finally, we will create a `Player` class to hold information and behavior relevant to the player.

Setting up the Map

Declarations of the classes `Location`, `Road`, and `Map` are attached. Write a corresponding implementation file for each class.

Use the above implementations to implement `Map::set_map()` so as to create the map of SeePlusia given on Page 2. This is the same map as in the previous assignment.

One way roads, e.g. from *Swamps of Despair* to *Sands of Quick*, are modeled as follows. The corresponding `Road` object contains pointers to both `Location` objects. The `Location` with the missing direction has the corresponding pointer set to `NULL`. For example, the `south` pointer of the `Location` object corresponding to *Sands of Quick* stores `NULL`.

Game Logic

Game logic is encapsulated in the `Game` class. Declarations of the `Player` and `Game` classes are in the attached files. Implement them with the same logic as in the previous homework.

Once everything is set up, the `main` program is simply as follows.

```
int main(int argc, char **argv) {
  Game game;
  game.run();
  return 0;
}
```

Put this `main` program in a file `main.cpp`.

Representing the Map

One benefit of encapsulation is that different components of the game are now independent of each other. For example, our game can now run on any valid map. A text representation of the map of SeePlusia is in the attached file `seeplusia.txt`. It has the following format.

- The first line contains, n, the number of locations in the map.

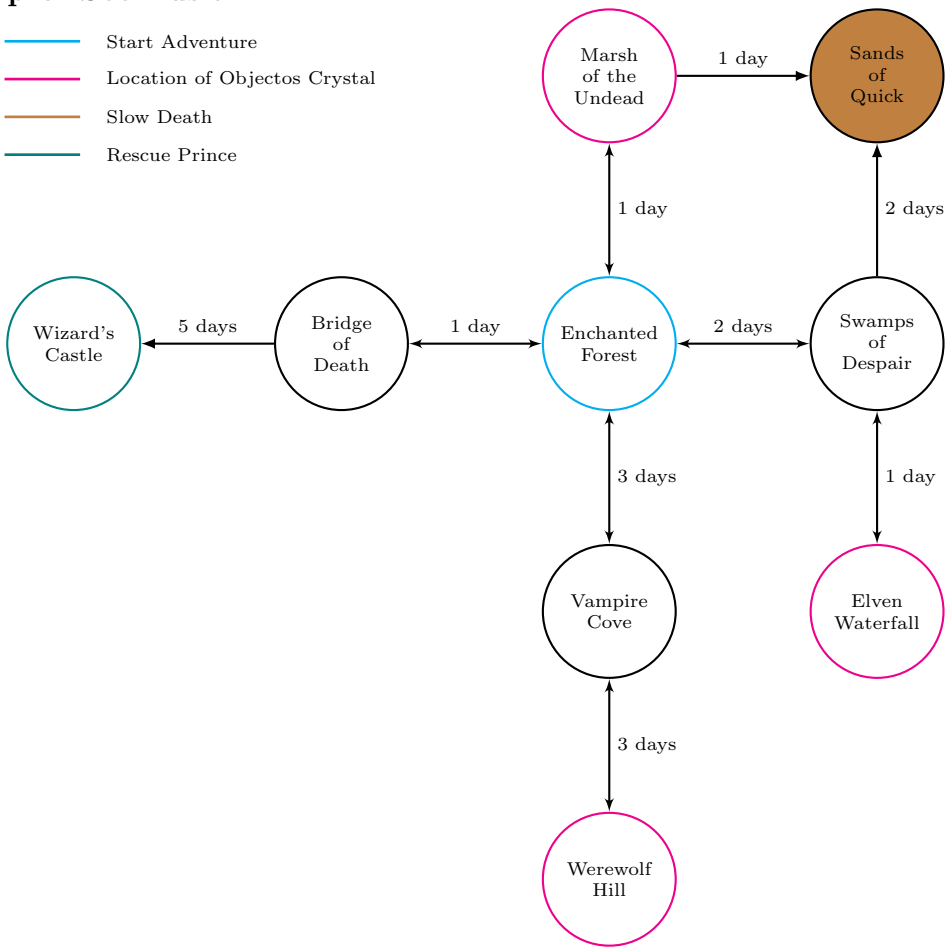
- Following the first line are 2 lines each for every location. The first of the 2 lines contains the id of the location and the second contains its name. The first location is the start location.
- Next are $10n$ lines—10 lines for each of the n locations. These 10 lines list the location's id, its neighboring locations and the number of days required to travel to each, and any special characteristic of the location. A blank value for a field indicates that the field does not apply to this location.

Implement the method, `Map::read_map()`, that takes a parameter indicating the name of a map file, i.e. the file specifies a map in the above format. The method, `Map::read_map()`, should read the map from the file and update the `Map` object accordingly. Test your method on the file, `seeplusia.txt`.

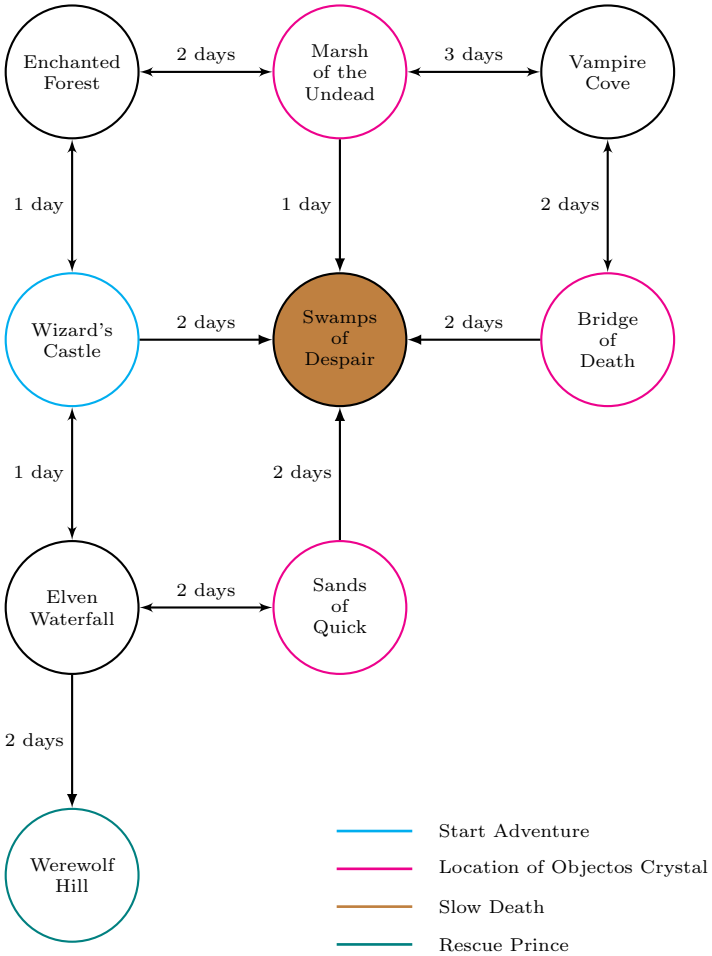
Level Maker

Write a map file in the format described above to represent the map on Page 3. Name the file `mymap.txt` and test your game by loading and playing this map.

Map of SeePlusia



Custom Map



Credits

The original map and assignment are courtesy of Naveed Ejaz.