Ashley Bertrand
Assignment 4

Description of mechanism you used to configure a cave configuration from a file and to save and retrieve games

I separated the saving and retrieval of the game status into two main areas. The first was implemented in a saveGame() method which gets called when 'Save game' is clicked. Current game settings are written to game.txt. The first line of the text file is the player's current location when 'Save game' is clicked. I then wanted to save the items that the player might be carrying; the second and third lines of the file are reserved for this purpose. Because a player can hold at most two items, it was safe to assume only two lines were needed to hold the player's items. If a player is not carrying anything when 'Save game' is clicked, both lines will be blank. If a player is carrying one item, only the second line in the file gets updated with the item's description, and the third line will be blank. Similarly, if a player is carrying two items, the first item's description is written to line 2, and the second item's description is written to line 3.

The remaining information that needs to be retained when a game state is saved is the status of the objects in each room. There are 12 rooms total representing rooms "outside", "r1", …, "r11". Lines 4 through 15 of game.txt are reserved for contents of each room, with line 4 representing "outside", line 5 representing "r1", and so on. I set up the program so that a room with no items would leave its corresponding line in the file blank, a room with one item would write the description of that item to its corresponding line in the file, and a room with more than one item would write each item comma delimited to its corresponding line in the file.

The second part of the save/retrieval process involves taking the information from the text file and loading the previously saved game. The user's game will be updated to the data of the most recently saved settings when 'Load game' is clicked. I wrote three separate methods to break up the components of this functionality. The first, loadGame() reads the text file and stores each line as a string into an array and then returns the array. This method is never called by the View and instead, gets called by loadPlayer(), loadCave(), and other helper methods to make sure that with each call, the most recently saved data is what is being worked with. loadPlayer() and loadCave() get called when 'Load game' is clicked.

The loadPlayer() method is responsible for setting the player's location according to the first line in the text file. It also sets a player's carrying objects. Before this can be done, it is important the any items that a player might be holding get wiped from the game, which is done by getRidOfItem(). If this doesn't happen, there is a possibility of multiples of items being introduced into the game. The player can then "carry" items according to lines 2 and 3 in game.txt. I wrote carry() for this process which has similar functionality as grab(), but it does not remove an item from a room. The item to be carried will not necessarily be in the room that the player is in, so a new method was required to accomplish this process.

Once a player's configurations are set up, loadCave() gets called to initialize objects for each room in the game. The method works by first clearing the contents of each room in the current game, by calling removeContents(). It then adds objects to rooms according to game.txt, supplying the information for the saved game, done by addContents(). Each of these methods run on a switch statement that uses that operates based on a given room. After all method calls have been made to restore the previous game, the player can resume the mission of finding the treasure from the level 0 saved game state. See more explanation in the last section of this report.

Description of the mechanism that you used to support "levels of difficulty"

To support levels of difficulty, I implemented the Abstract Factory design pattern. The motivation for using this pattern is that it allows for the creation of a system of multiple types. In this project, the multiple types are the different configurations that come with different levels of the game. The system that is constructed depends on which level, 0 or 1, the user selects. Both are built the same way, but specific parts differ between the two.

For my specific implementation, I made an abstract class, AdventureGameFactory. Its method to construct the configuration is createGame(). It takes an AdventureGameFactory object as a parameter, which, in the concrete classes, is used to instantiate the objects. The functionality for this method along is implemented in two concrete classes. The first is Level0Factory which sets up the game just as the Adventure class does in the original implementation. Level1Factory is the second class I wrote to give additional functionality to the game. When a user runs the game in Level 1, they are an “advanced wizard” exploring Hogwarts grounds where new enhancements come into play if they find the Marauder’s map. This is a magical document which under certain circumstances reveals the location of the key and the treasure. I wrote getHints() in Level1Factory which differentiates the two classes. More detail on this functionality is described below.

Description of the other enhancements that you made to the Adventure Game
I wanted to keep the Harry Potter theme running through my game and came up with the idea of adding the Marauder’s map. As any Harry Potter fan would know, the Marauder’s map is a map of Hogwarts grounds, which also reveals the locations of people in real-time. In my game, I decided to make it so a more advanced, mischievous, explorative wizard in level 1 has to opportunity to use the Marauder’s map to his advantage if they can find it. With the initialization of level 1, the Marauder’s map gets placed into room 9. A player who has not yet found the treasure will receive hints as to where it is hidden. I made another addition so that a player who is not carrying the key will also be shown its location. getHints() returns a string listing the available hints according to these restrictions. Revelations from the map are advantageous to a user who has learned how to maneuver about the cave. Constraints with the hints are listed in the following section.

What a user needs to know to play game/Special instructions for installing and running your program

- In loadGame() within the AdventureGameModelFacade class, the file path must be changed accordingly before the program can be run. This is the file path where the program looks to read in information from a previously saved game.

- Ensure you are running code from A4 (not a previous assignment by mistake). In the GUI you will see load, save, and level buttons in addition to a hints text field for A4. In Netbeans, for example, I run AdventureGameView by right clicking the class and selecting ‘Run File.’

- As one would expect, a game cannot be loaded until a game has been saved. In terms of the program, this means that if a game.txt file does not exist (because ‘Save’ has not been clicked in the GUI), the program does not have any information to load.

- If you run the system without selecting a level, the game defaults to level 0 functionality. If you are in the middle of a game and you select a level (or change the level) you begin a new game in that selected level. No previous game content is saved unless you click ‘Save” in the GUI.

- Hints are only provided when a player begins the program in level 1. In other words, when loading a saved level 1 game, player does not get to see hints. I wrote the program this way to prevent “cheats” in the system.