Jason Vessella

CSC 201

Program 5 Report

5/2/16

**Introduction**

The objective of program 5 is to make a very basic game. It needs to have semi-random generated levels, the character needs to be able to move, and there must be items that the character and monsters(if created) can interact with, such as gold, weapons, armor, etc. The specific classes must be made, and basic player movement commands must be implemented in the game.

**Methods**

For the game to work, many classes must be implemented. The first class that will be discussed is the Player class. This class contains the basic attributes of a player in a game. The beginning of the class has several variables set so that they may be manipulated later, some of which are the storage and location variables. The equip, remove, and unwield functions are pretty self explanatory, they allow the player to equip/remove items, with unwield specifically pertaining to weaponry. The eat function attempts to eat the indicated item, and provided that it is a food item, it will be eaten and removed from the inventory. The functions move\_to and get\_loc both deal with the player’s location, allowing it to move freely about the map. The inventory function prints out a list of what the player is holding at the time of calling the function.

The Treasure class is comprised of several subclasses and a few attributes of itself. Those actions are the location, pickup, drop, wear, remove, wield, unwield, eat, lookat, getType, and getWorth. The first gets the location of the treasure on the level itself. The next 7 functions pertain to actions executable by the player on treasure items, each returns a default message that is later changed by the functions of the subclasses. The next three functions return descriptions, such as gold value of an item, the item description itself, and the type of weapon that it is.

The Armor class initially contains the worn variable, which is initially set to false. The functions wear and remove manipulate the variable worn, to determine if the player equips a piece of armor or takes it off. The Food class simply contains a function to eat the assigned item, provided the item is of the food type. The gold class simply allows the user to look at how much gold they are currently holding. The Weapon class does the same as the armor class, contains functions that allow the player to wield or unwield an item, provided its item class is “weapon”.

The second file submit for the assignment is the main file that runs the level generator and incorporates the other classes. The first thing it does is import the other classes into it, then it randomly generates the level grid with set parameters, using the specific characters mentioned in the assignment itself. The program generates the level with the given pieces listed, and then removes any duplicates, but leaves ‘\*’ to symbolize items. The level is printed in a string, and the string is returned and prints in the shell. Next, the function “init” is defined, so that variables would not be used globally in the program. The next to functions, up\_level and down\_level were defined as test procedures to see if one could easily generate new levels or end the game (moving up when there are no floors above). The program then initializes the Player class, and the next function is used to find the player in the generated level. If the player is found, it prints a message stating so, and says it could not be found if the player does not appear for some reason. The next function, “update\_tile” updates the string every time an action is made, such as the player moving, while the function “get\_tile” returns the height and width location of a specific tile. Next is the while loop that keeps the game running, then the user commands are defined, and able to be printed to the screen by simply typing a ? mark within the command prompt while the game is running. If the commands “^” or “v” are entered, the program checks which tile your are currently standing on, and if it is not an up or down staircase respectively, it will give you an error message and not act. Finally, when a movement command is entered, it checks if you are moving into a wall or able to proceed to the next space before displacing the P on the screen, and prints the level after every movement command.

**Discussion**

This project as a whole was extremely difficult in my opinion, and is unfortunately far from bug free. Due to difficulty and time constraints, the game is lacking some features, such as actually being able to pick up items. I tried several times to implement a random item generator, as the piles of treasure in the game are all displayed by one symbol, but for lack of time, it was not completed and thus removed from the file to make it easier to read. The system of health, damage, and armor were never implemented either, although in the player class itself, they are all set to 0 by default. However, without the items in place, this could not really be changed without adjusting the base values. As for monsters, they certainly appear on the map as an M, though they cannot be interacted with, and might as well be empty cells. On the plus side, the rest of the movement works correctly, and the levels generate randomly.

As for bugs with the program, there are a few that I have found and been unable to fix. One of which is that when the character bumps into a wall and the “Cannot move there, that is a wall” message is displayed, a new wall appears in that location once the player moves again. This can lead to the player boxing themselves into an inescapable space, causing the user to either have to close the game or run the function to force a floor up. The other bug I have found is that the stairs and player seem to all generate near to each other, sometimes even side by side.