Name: Brian Jon Stout
Date: 04MAR2017
Current Module: Object Oriented Programming
Project Name:Dungeon Dudes

Project Goals

The goal of the project is to create a linear RPG with basic attack mechanics. The character has to be able to move from room to room, attack enemies, check his own status, check the enemy's status, and check his inventory. The rooms need to have descriptions upon entering them. On entering a room the hero and the monsters should also roll for initiative. Initiative decides who attacks first.

Considerations:

- The program must use classes and OOP efficiently
- The dungeon must end with a boss
- The character must have menu options at all times

Initial Design:

The main file to run is dungeon_dudes.py. It contains almost all of the gameplay logic, and the basic battle loop.

The imports are sorted into two folders called the module folder and the entities folder. The module folder contains specific functions used in the program, and the entities folder contains individual class files used to generate objects in the game.

The module folder as of now only contains a dice module, which is used to roll 6 sided die, 10 sided die, and 20 sided die.

The entities folder contains classes for creatures, the hero, loot, and rooms.

The creature class contains the attributes and methods used for all living things in the game. It stores values for it's name, how much hp it has, and the max HP it can have. It also has the methods used to interact with other creatures, such as attacking them, or taking damage from them.

The hero class inherits logic from the creature class since it is a living thing. It contains additional information for storing items in an inventory.

The loot class contains the information to generate loot objects, which contain the name of the loot, it's value, and a description of the object.

The room class contains information used to generate room objects. Room objects have meta data on whether a room was successfully completed yet, and contains a list of all the monsters in a room that the hero must defeat.

Data Flow:

The program starts by asking you for a hero name, and uses that name to generate a hero object. Afterwards it generates a boss object. The game then runs the generate_rooms() function which decides randomly how many rooms are there in the dungeon from 7-12 and ensuring the last room contains the boss.

The generate_rooms function creates a list of room objects based off the Room class. When a room object is created it generates a list of creature objects, and sets values for the name of the room and it's description. The creature object is from the creature class which handles the methods for all living things in the game.

The program initializes a value keeping track of what the current room is, then starts the main gameplay loop.

A quick check is done at the start of the loop to see if the player has beat the dungeon by the current room equaling the amount of rooms total. since CurrentRoom starts a 0 and it means all the rooms are completed. The game prints out the room description, then assigns the currentRoom's monsterList (roomlist[currentRoom].monsterList) to the variable monsterList for easier reference.

Once all that's set up, the roll_for_initiative function is called which sets the initiative value for all the creature objects. The program sorts the monsterList based on the initiative of the monsters, then checks to see if the Hero's initiative is higher than the monster's initiative. If it is not then the monster gets to attack the Hero before he can make his first action.

After this the use_menu function is invoked which gives the player the option to check his inventory, check his status, check the enemy's status, move on to the next room (or attempt to,) and attack the enemy. Checking the statuses or inventory is self explanatory. Moving on to the next room attempts to set the room's movingRooms value to true, which the game logic uses to determine if it should move to the next room. If there aren't any monsters it's guaranteed, if there is the player has to roll a d20 for each monster in the room, and if it's below a 15 it fails. Attempting to move rooms takes a player's turn.

The player can also chose to attack the monster with the highest initiative. He rolls to see if his attack hits. The game checks to see if the mob is dead, and if it is, it pops it from the list and gives the player a random loot object (generated from the loot class) and adds it to his inventory. After this, there is a check to see if there's any more monsters, if there isn't, the room cleared attribute is set to True.

After the menu option, if there's any monsters alive, the monster with the highest initiative will attack the player. After the attack the game checks to see if the player's hp is still above 0, if it's not it'll tell the player he's dead and exit the game.

This is the basic gameplay loop and runs all the way to the boss. When the boss is killed the game exits the loop and prints a success message.

Communication Protocol:

Not applicable.

Potential Pitfalls:

Over complicated gane logic

Test Plan:

User Test:

Running through the game multiple times

Test Cases:

Ran into some index errors when trying to generate rooms for the boss. Eventually fixed the bug. Luck of the dice is incredibly important for this game.

Conclusion:

This project was a good opportunity to see how classes and OOP methodology can be useful. However, using classes and objects correctly is a new skill that requires practice, so doing it that way requires some extra thinking and time for beginners. I got bogged down initially trying to make the program work with more complicated classes. Using test driven development was also useful, but took too much time away from actually coding the project, so I had to revert back to old ways to get all the features implemented in correctly.

The program works correctly, albeit it's missing flourishes, but the main gameplay loop is clean, and the classes are set up to make a more complicated RPG game in the future relatively easy as of now. The code is fairly reusable. The weakest part was generating the dungeon rooms and the lack of combat complexity.