Computer Science 1 (1021) -- Spring 2013
Lab & Homework 11
Text Games II

Topics covered: Object-oriented Programming

Demonstration due end of lab (show progress) April 3, 2013
*Team Leader submit code on Blackboard by midnight **Monday April 8***
One printout submitted per group.

This lab is a continuation of Lab/Hw 10. You'll be writing a text-based game, just like in Lab 10, but this time using objects and their programming features. Rather than individual submissions, you're asked to form groups of 2, 3, or 4 and build and submit a single game.

By end of lab on Wednesday (8pm) the team leader should email me (paul.talaga@uc.edu) the members of their group. Additionally each member should submit a peer evaluation to Blackboard (see attached document) by midnight April 8, assessing the other group members.

Below are the requirements for this project. These are in addition to the requirements from last week.

1. Use objects to store locations and their state. All 'state', other than the current location, should be stored in an object.
2. Use of at least one class/object using **inheritance**.
3. All object instance variables should be created, accessed, and changed ONLY by methods of that class. Even though Python does not enforce **encapsulation**, you're required to follow it. Thus, you'll need 'getter' and 'setter' methods.
4. Each method/function must contain the group member's name(s) who wrote or contributed to that code. All group members must write or assist with at least one method/function.
5. Have at least one method with the same name defined for different classes. Example: <location>.print() and <hand>.print() which prints the content of either the location or the hand.

Below are the old required features:

1. When starting the game, or when the user types 'Help' as a direction/action, the game should show a description of the game, with general actions allowed.

2. At least 5 areas. You can have rooms, locations, or whatever you call them, but there must be at least 5 areas a user can navigate between.

3. Description of your location. Whenever the user moves in your game it should print out the new location and a short description, similar to the text on P222-P225 of our book.

4. At least 6 directions/actions. The game must allow at least 6 actions in each location {North, South, East, West, Exit, Help}. You may use N,S,E,W, (or lower case) just be consistent and make sure Help describes this. Exit quits the game. Some directions may be invalid and will keep the user in the same spot. During the description of the location, the game MUST tell the user the possible actions allowed.

5. Special direction. In at least one area, allow a special direction such as 'Down', 'Up', 'Jump'. This brings you to a new area. Be sure to tell the user these possible directions.

6. Include at least one action in at least one room, such as 'GrabBomb' or 'DropBomb'. This can be chosen when the user enters a location (think of it as another direction). Doing an action does not move the user. This also means a user can have an item (or items). Make sure you inform the user as to what items they currently have.

7. Win/Lose. Include the ability for a player to win or lose the game. This could be finding some spot, or being in some location while possessing some item. Similarly allow the user to lose.

Grading:

1. 5% - File headers, function headers and comments. Be sure to insert comments (at least one) describing any complex operation into each function. Limit your line width to 80 characters.
2. 5% - File name as specified, paper copy, and runGame() function.
3. 5% - Peer evaluation submitted.
4. 4% - Group list emailed on-time.
5. 50% - Satisfaction of each (10%) of the new requirements.
6. 21% - Satisfaction of each (3%) of the old requirements.
7. 10% - Individual score for effort based on peer evaluation.

Extra Credit: 5% extra credit for each of the following extra functionality.

1. Infinite gameboard – Dynamically generate locations so the user can never explore all regions. Getting into a loop does not count.
2. Location map. After the user picks a new direction, draw (programmatically) a map of the world and where the user is within it.

Use show() to show the generated image to the user.