Lab 2: Conditionals

Due: Tuesday, July 23rd, 10:00pm

“Aliens have invaded a spaceship and you are trapped. You have to go through a maze of rooms overcoming obstacles, solving puzzles, and defeating the aliens so that you can escape into an escape pod to the planet below.”

Using conditionals and taking in user input to modify your program during run time makes programming all that more interesting and powerful. Your job in this lab is to make a game, to be more like an adventure type of game, similar to the sample storyline in the quote above, with constant user interaction and amusing text outputs. Your game can be any kind of game you want in the same flavor. You are free to choose your own theme and the scenes to be used in your game. The game will involve an engine that runs a map full of rooms/scenes. Each room will print its own description when the player enters it and then tell the engine what scene/s to run next out of the map.

To make the game more interesting and entertaining, each player can have up to three lives. When a player dies, they are taken back to a checkpoint which your game should keep track of, unless they run out lives (not every scene has to be a checkpoint. While your program is running, the game should keep a scoreboard of the best players (you can make your own point system, or simply have the amount of moves needed to escape determine this).

The game should be made as interesting as possible, because who likes a repetitive game? Therefore, the last component will be to give the user the option to choose the difficulty of the game, between easy, normal, and difficult (maybe also adding different multiple levels to the game?). Therefore, feel free to add more scenes and more clever ways to go between rooms and solving puzzles. You should be using lists, functions, and modules as much as possible, and find as many new pieces of Python as you can to make the game work. Create the rooms, monsters, and traps that the player must go through on paper before you code. Once you have your map, try to code it up. If you find problems with the map, then adjust it and make the code match. One final word of advice: All programmers can become paralyzed by irrational fear starting a new large project. The best way to avoid this is to map out the program and code one section at a time, always testing and debugging along the way.

This is how the game should behave:

1. User enters their name, selects the level and difficulty, and begins in the first room.
2. The game sets up the map and asks the user what action to take.
3. Each action leads to different outcomes, which move the user around the map until they escape.
4. Certain rooms are checkpoints, which the user is notified when they have reached one.

5. If a player dies he is taken to the last checkpoint, unless they have no more lives left.
6. Once a user escapes, their score is added to the leaderboard and they can play again.

I have provided you with the basic skeleton for the project, you can check what sections of code you need to write by typing “grep todo game/*.py” in the terminal inside the game directory. Make sure to include comments and tests for your classes and functions. I’ve provided a module that you can use to type and run your tests. Read the comments I’ve made to see how each test function works.

Teams

For this lab, we will be working in teams. So, find your team, and get in a group. The teams are as follows:

Team 1	Team 2	Team 3	Team 4	Team 5
David Calahan	Sophia Pegues	Ismael Carreno	Nicole Avila	Jared Hernandez
Akwe McDaniels	Emily Simon	Alyssa Shteyn	Spencer Hoffman	Diego Garza
Chenjia Lin	Ismail Youssou	Nathaniel Martinez	Daniel Serrano	Charlie Mayville
Arleth Salinas	Branson Starr			

Read all the lab carefully and note how grading will work in the submit section.

Setup

Copy all the lab resources to your Lab2 directory, add, commit, and push.
Now, by the end of the Lab, you will have an executable file, which you should be able to run on it’s own. However, in order to be able to run tests and make files from your python scripts, you need to install a few packages, including *pip*, *nose*, and *pyinstaller.*

This is pretty simple once you get pip installed. I took the liberty of adding the pip installing script in your lab resources, so if you don’t already have pip installed, simply switch to the docs directory, and run the python script with this command in the terminal:

py -3 get-pip.py
Or

python -3 get-pip.py

This should installed pip automatically. Let me know if you have any issues. After pip is installed, you can simply run this command to install the rest of the packages:

pip install <PACKAGENAME>
E.g. pip install distribute

Or from the lab directory:

Pip install -r requirements.txt

Pip basically acts as a package installer for python, so go ahead and install all the packages I mentioned above **(*nose, pyinstaller, and virtualenv (optional)*).**

Now, you can work on your game. Change directories to the ***game*** directory. Here are all the python script modules you need to work on. *"grep todo game/*.py"* will list all the sections of code you need to write.

Testing

An important aspect of coding is testing. This is incredibly useful for avoiding messy code and annoying bugs. Therefore, in the ***tests*** directory, you are to edit the *game_tests.py* by importing the different modules you work on and creating tests for each part of your code. Do this often and constantly. To run tests, go back to the root directory of this project, and run this command:

nosetests

Make sure you are running the command from the root directory of the project or else it won't work.

Making an executable file

Once you have working copies of your scripts, switch to the game directory, and within it, I have added a batch file for Windows (make.bat) and a script for macs (make.sh). You can run each from the terminal, which should call on the pyinstaller package to make the executable and thus a standalone copy of your game. This should create two new directories, *dist* and *build*, one with the executable file and the other with all the components needed to make the executable file. You will find the game in the ***dist*** directory.

If for some reason the scripts to run the command don't work, simply type the command manually every time, the scripts simply automate the process:

```
pyinstaller --onefile game.py --name <filename>
```

Submit

To submit your work, simply commit and push the files to your lab2 directory in your git repository. **I will choose one team member to be graded randomly, and everyone else on the team will get the same grade.** Therefore, you are all in this together. Help each other out, but everyone will submit their own work. Good luck!