Computer Science 217
Assignment #3

John Aycock

Due: 4pm Friday, 26 March 2021 (Calgary time)

Purpose

To write code to specifications; to use loops, conditional statements, Boolean logic, lists, and dictionaries; to perform basic input handling and error checking; to compute the results of formulas.

Important Notes

- This is an individual assignment. What you submit must be your own work, and you **must** write the code yourself, although you may discuss the problem in general terms with other people. You should definitely **not** be showing other people your code, and generally speaking, it is not a good idea to talk about the assignment when you're sitting in front of the computer.
- Any references that you use for algorithms and code references, if any, must be properly cited. Remember that plagiarism regulations apply to code too. There's no required citation format but the citation must uniquely identify the source. You can put citations into comments in your Python code. Note that a reference would be used to answer a minor question like "how does this Python construct work?" If you are copying and pasting code into your assignment, this is **not** writing the code yourself, and completely defeats the purpose of what you're supposed to learn in this course. Using the 217 lecture/tutorial videos and slides as references can be safely implied from this context, I think, and you need not cite those unless you're drawing a nontrivial amount of code from them verbatim.
- If you have any questions about what you can and can't safely do, feel free to email me.
- Per course policy, late assignments are **not** accepted. Check the due date and time carefully.

Assignment

The text-based game "Star_Trek" has been around in many forms since the early 1970s. You will write a simplified version of it, as specified below.

A sample implementation is installed on the CPSC machines that will allow you to discover the finer points of how your game should behave. You can run it by typing `217trek`.

- The universe is a 12x9 grid with location (0,0) in the top left corner. The current state of objects in the universe is printed every turn. It should be displayed as shown below:

  ```
  . K . . . . . . E . . .
  . . . . * . * . . . . K
  * . . . . . . . . . . .
  . . . . . . . . . . . .
  . . . . . . * . . . . *
  * . . . . . . . * * . .
  . . . . . . . K . . . .
  . * . . . . . . . . . .
  . K . . * . . . . . . .
  ```

  This shows empty space (`.`), the Enterprise (`E`), Klingons (`K`), and stars (`*`). Only one object may be in a given location at any point in time.

- Any unknown commands should result in an error message.

- A `quit` command exits the game, which can be abbreviated as `q`.

- A `destruct` command counts down from 5, issuing appropriate messages every second, then exits the game.

- There is one Enterprise, whose initial location is chosen randomly (throughout this specification, "random" refers to a pseudo-random value generated using Python's `random` module). The Enterprise starts with 125 units of energy, and is destroyed if its energy is fully depleted.

- There are ten stars, whose initial locations are chosen randomly. A star cannot be destroyed, and the Enterprise cannot fly into stars.

- There are four Klingons, whose initial locations are chosen randomly. They each begin with 50 units of energy. At the beginning of each turn, before the universe is printed out, all Klingons within range of the Enterprise will try to shoot at it; a message should be printed for each Klingon in range, indicating its location and the result of the shooting. A Klingon is in range if the Euclidean distance between it and the Enterprise is less than 3. A Klingon has a 42% chance of hitting the Enterprise. When a Klingon's shot hits the Enterprise, the Enterprise loses a random amount of energy between 5 and 10 units, inclusive.

- Movement commands: `north`, `south`, `east`, and `west`, which may be abbreviated as `n`, `s`, `e`, and `w`. North is located at the top of the grid. This moves the Enterprise in the universe in the corresponding direction. The Enterprise is powered by Vulcan farts and so this movement consumes no energy. The Enterprise may not move beyond the bounds of the universe and does not "wrap around" from one side of the universe to the other.

- The modern Federation is a kinder, gentler Federation. Instead of using nasty phasers or photon torpedoes to shoot Klingons, the Enterprise executes a "boop" maneuver by attempting to fly into a space where a Klingon ship is located. Doing so inflicts a random amount of damage to the Klingon between 23 and 28 energy units, inclusive. If the Klingon runs out of energy, the Klingon is destroyed and the Enterprise takes its place in the universe map.

- If all four Klingons are destroyed, the game is won.

While the map must be displayed as shown, the text of messages does not have to match the sample implementation exactly, as long as the same information is given.

Spoilers

You may read these, or not, depending on how much help you want.

Minor spoilers (click to view)

- The main game loop will look very similar to the text adventure game, particularly the later versions of the text adventure game I've written in past videos.
- Start with creating and populating the universe map, and printing it out, before bothering with input and commands.
- Implement the universe map as a 2D list of lists containing strings, i.e., indexing into the map for a given (x, y) coordinate gives you the type of object that's there.
- After that, easy things to get working first: quit, destruct, unknown commands. Then, movement commands -- this will get a lot of the game framework in place so you can at least move around.
- Normally Python's `break` statement would be used to break out of the game loop and exit the game, but you may find Python's `sys.exit()` function useful too, if the game is won or lost when you are deep inside nested loops (remember `break` only breaks out of one level of loop).
- Recall that Python's `print()` function has some formatting options; see the notes on lists.
- Don't forget to check if anything's in a space already when you're creating the universe or moving.
- When placing objects in the universe, you can use an infinite loop, choose a random (x, y) coordinate, and break out of the loop only if the space is empty. (This is actually done by some games.)
- As a matter of good style, define variables near the top of your code for all the numeric values in the specification, so you know what they are when you use them in the code. If you were creating your own game, you'd also use these variables to tune the gameplay.
- The `pop()` operation for dictionaries can be used to remove a dictionary key and its associated value from the dictionary. For instance, if `D = { 'abc': 123 }`, then `D.pop('abc')` would make `D` an empty dictionary afterwards.

Major spoilers (click to view)

Assignment Submission

To hand the assignment in:

1. Place your name, student ID number, and tutorial number into comments at the top of your Python program.

2. On the CPSC Linux machines, assuming your Python program is named `as3.py`, run:

   ```
   submit as3.py
   ```

   You can resubmit your file as many times as you like prior to the deadline. Note that a new file submission overwrites the previous file submission, so be sure to only submit your Python program and not any other files!

   Afterwards, if you want to see what you submitted, run:

   ```
   seesub
   ```

Evaluation

You cannot be given a grade above zero if:

- You do not submit a Python program with the required information.
- It does not run at all on the CPSC Linux machines using `spy3`.

Marks are as follows. Marks are only given if a feature is fully implemented as specified.

- 1/16 -- comments used in code to identify program structure and nonobvious things
- 1/16 -- the universe is populated according to specifications
- 1/16 -- universe map is printed correctly each turn
- 1/16 -- error message given for unknown commands
- 1/16 -- quit command works
- 1/16 -- destruct command works
- 1/16 -- movement commands work for all four directions
- 1/16 -- Klingon firing works along with Enterprise damage/destruction
- 1/16 -- "boop" works along with Klingon damage/destruction
- 1/16 -- checking for winning and losing is done and works
- 1/16 -- mapping short (abbreviated) versions of commands into their longer forms is implemented using a dictionary
- 1/16 -- tracking Klingons and their energy levels is implemented using a dictionary
- 1/16 -- universe map is implemented using a 2D list
- 1/16 -- bounds checking for the Enterprise's movement works
- 1/16 -- the Enterprise cannot fly into stars
- 1/16 -- size of universe defined by variables whose values are used appropriately throughout code

---

John Aycock, 09 March 2021