**Computer Science 231   
Assignment #2**

[John Aycock](mailto:aycock@ucalgary.ca)

Due: 4pm Friday, 3 March 2017

**Purpose**

To use while loops, conditional statements, Boolean logic, and lists; to perform basic input handling and error checking; to use functionality from a module; to work with program specifications.

**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.
* Sources of algorithms and code, if any, must be properly cited. Remember that plagiarism regulations apply to code too. You can put citations into comments in your Python code.
* Decompiling or disassembling the example implementation supplied is not permitted.
* If you have any questions about what you can and can't safely do, feel free to [email me](mailto:aycock@ucalgary.ca).
* Per course policy, late assignments are **not** accepted.

**Just Like Rogue… without all the fun parts**

The 1980 game [Rogue](https://en.wikipedia.org/w/index.php?title=Rogue_(video_game)&oldid=763234987) was an influential game in what is now called the “roguelike” genre – although it was not the first, interestingly. It featured a text-based map that a player would navigate through in their search for the Amulet of Yendor, because you just can't buy those things from the Shopping Channel.

Games can be repurposed for other tasks, and here you will be creating a similar interface for a map that could be used, for instance, in a “retro” information kiosk. Rooms in the map (really, rectangular regions) have associated messages that are displayed when the player moves into them, and those can be used for conveying information.

* Your program will display a text-based map, as described in the Map Specification Language section below, using the curses module.
* Walls are marked by '.' characters, and rooms are marked by space characters.
* The player's location is marked by a '@' character. The player can move around within rooms: w or i to move up; a or j to move left; s or k to move down; d or l to move right. The player cannot move beyond the edge of the map.
* The player cannot walk through walls.
* The player should be initially placed somewhere inside the first room defined in the map.
* Your program will be run as follows. A config.py file containing the map will be present in the directory your program is run from that your program can import.

% python3 as2.py

* Your program must run with multiple different maps. Different maps are supplied by changing the config module (i.e., config.py) to a different one.

A sample implementation is available for you to run on the CPSC machines, which will help you understand how your program should behave when it is complete. It's set up to use the “test-fancy” configuration. You can run it by typing ~aycock/231as2 from the CPSC terminal prompt (***not*** the Python interpreter prompt). It is ***not*** accessible via the web.

**Bonus (+1 mark)**

If you make a particularly creative map manually using the map specification language, send it to [me](mailto:aycock@ucalgary.ca) directly. To get this mark, it should be particularly interesting, clever, or artistic, and it should be obvious that you spent a nontrivial amount of time on it. If you want to write a separate Python program that produces maps using procedural content generation, I would also consider that for the bonus.

**Bonus (+1 mark)**

Add a command to the map description language that defines a location by a letter and a set of x,y coordinates:

location A 5 10

Modify your user interface such that, when the player presses the letter of a location (more than one may be defined), the player's symbol gradually “walks” over to that location. ***Note: walk. Not teleport.*** You will need to implement a pathfinding algorithm like A\*. As with the other bonus, if you do it, please send it to [me](mailto:aycock@ucalgary.ca) for the extra mark.

**Map Specification Language**

A map is located in the config module. There will be a string variable named CONFIG inside that module that contains the entire map description using the language described below. No other assumptions about the config module's contents may be made.

* CONFIG is a string with one or more lines that are delimited by a newline ('\n') character.
* Blank lines should be ignored.
* A line starting with a '#' symbol is a comment and should be ignored.
* All other lines contain map specification commands, one per line, and a line consists of words separated by spaces.
* There must be at least one room command.
* A room command takes the following form:

room ulx uly w h label

*ulx* is the upper-left *x* coordinate of the room; *uly* is the upper-left *y* coordinate of the room; *w* and *h* are the room's width and height, respectively. All units are in characters, with (0,0) in the upper left corner of the screen. The *label* is one or more words, and when the player is located on one of the room cells, the corresponding *label* should appear on the bottom row of the screen.

Rooms may overlap partially or completely; space (and labels) from rooms that are defined later in the map configuration take precedence over earlier ones.

* A char command takes the following form:

char ch x y

*ch* is a single (non-space) character; *x* and *y* are the coordinates to place that character at on the screen. If the player is located at those coordinates, the character there is not visible until the player moves, at which time the character is restored.

* Unknown commands should be ignored.
* You may assume all maps will fit in 22 rows and 80 columns, on an 80x24 screen.

Your TA will be assuming that you adhere to this specification, and will also be using test map modules that you have not seen before. For the files you have seen, they will be testing your program with fresh copies of them. **If you don't adhere to this specification or you read the map files in some different way, and your program cannot be run as specified, it will be treated as nonfunctional for the evaluation.**

Sample map files are below. To use one, download it to the directory where you are running your assignment and name the map file config.py.

* [One-room map](http://pages.cpsc.ucalgary.ca/~aycock/231/test-oneroom.py)
* [Map with overlapping rooms](http://pages.cpsc.ucalgary.ca/~aycock/231/test-overlap.py)
* [Map with more overlapping rooms and decoration](http://pages.cpsc.ucalgary.ca/~aycock/231/test-fancy.py)
* [Map with more overlapping rooms, decoration, and an edge hallway](http://pages.cpsc.ucalgary.ca/~aycock/231/test-edge.py)
* [Map with many rooms, locations for the pathfinding bonus, and a secret room](http://pages.cpsc.ucalgary.ca/~aycock/231/test-cpsc231.py)

**“Got any tips?”**

Minor spoilers

*You are free to ignore these.*

* Have a look at the supplied map files to get a better understanding of what the input looks like.
* Run the sample implementation and watch its behavior carefully, and compare that behavior to the map files.
* The main input loop will look very similar to the input loops you've seen many times now, where you read a line and process the one line, and do this repeatedly in an infinite loop.
* String splitting can be useful for breaking apart map descriptions.
* The curses module reference as well as a “howto” are on the python.org website. Google “Python 3 curses module”.
* You don't need that much out of the curses module. You can do this with initscr, addstr, addch, inch, move, getkey, deleteln, and refresh. You'll probably want noecho too. You can use different functions from curses but if you're straying *too* far from this list you may be heading the wrong direction.
* One of those curses functions in the list allows you to query which character is at a specific place on the screen.
* You may find that having a 2D data structure gives you a good place to keep track of what labels to display at any given time.
* Test your program on *all* the supplied map files and carefully verify that you're getting the correct results!

**“I have no idea how to do this!”**

Major spoilers

*You are free to ignore these. There are many ways to do this assignment.*

* All minor spoilers still apply.
* Write a small test program using the curses module to initialize it (initscr), add a string to a specific row/column location (addstr), and update the screen to make the string viewable (refresh).
* Extend your test program by reading a keypress from the keyboard using getkey. You've now used most of the critical curses functionality you need. Set it aside for a little while.
* Your program's structure can be something like this:
* initialize map
* initialize curses
* read map configuration from config.CONFIG

main loop getting user input from curses and responding

* You may find this code helpful for setting up a map-sized 2D array of strings:
* MAPROWS = 22
* MAPCOLS = 80
* MAPWALL = ''
* # initialize map
* map = []
* for i in range(MAPROWS):

map.append([MAPWALL] \* MAPCOLS)

Be sure to cite the source of this in a comment to make it obvious to the TA that it's not yours!

* You can split config.CONFIG into a list of lines (strings) if you split it on '\n'. Then you can iterate through the list to read the map configuration, very similar to other examples you've seen for reading input. Don't forget you can split each individual line too to get a list of the words in the line.
* You can use a Boolean flag variable to detect when you've read the information for the first room. (Remember that the player has to be located in that room initially.)
* You can write the code to go through the map configuration and test it before adding the main user input loop.
* The main loop's structure can be an infinite loop, very similar to the loop for the text adventure game. You're just getting your user input from curses and getkey as opposed to using input.

**General Hints**

* Make a directory to keep your files for this assignment in so you can keep them separate from things for other assignments and tutorials.
* Take notes as you go along, so you can remember what you've tried already and what did and didn't work.
* Break the task down into small pieces.
* Use an incremental approach, and test as often as possible!

**Assignment Submission**

To hand the assignment in, send *one* email to your TA containing:

* Your student ID.
* Your Python source code .py file. One only.

You are responsible for getting your assignment to your TA on time; keep in mind that email can take a few minutes to deliver! To be safe, you should CC yourself a copy of the email too.

**Evaluation**

You cannot be given a grade above zero if:

* You do not submit the required two items.
* There is no submission or the code doesn't run at all.
* You do not have a program in a .py file — you cannot turn in a solution that only uses the Python command line.
* Your code does not run on the CPSC machines using Python 3.
* Your code is not using the curses module.
* Your code is not using the map configuration from the config module.
* You have different .py files for different maps – you must have one .py file handling *all* maps.

Your assignment will be given a grade as follows. Marks are only given if a feature is fully implemented as specified.

* 1/13 — Player motion using both wasd and ijkl.
* 1/13 — Player motion honors room boundaries, i.e., no moving into walls.
* 1/13 — Rendering rooms in single-room map correctly.
* 1/13 — Rendering rooms in multi-room maps correctly with overlapping rooms.
* 1/13 — Rendering additional maps correctly adhering to the specification that are not supplied in advance.
* 1/13 — Printing room descriptions.
* 1/13 — Printing room descriptions when room overlaps exist.
* 1/13 — Rendering char commands correctly.
* 1/13 — Restoring char commands' characters after player moves over them.
* 1/13 — Player placed in first-defined room.
* 1/13 — Edge movement correctly handled with no wraparound or errors.
* 2/13 — Code quality.

For the code quality mark, here's a rough, not necessarily all-inclusive guide to properties that would get the different marks.

* 0/2 — Appears to be one dense blob of code; no documentation; documentation does not agree with code; variable name choices don't reflect their usage; heavily overdocumented; documentation has extensive spelling/grammar errors; design of code is unclear; unnecessary duplicated code.
* 1/2 — Somewhere in between.
* 2/2 — Documentation well written and used where needed; variable name choices reflect usage; documentation and code agree; whitespace used effectively in code to convey logical groupings of code and make code aesthetically pleasing to read; design clear; no unnecessary duplicated code; could use as an example of excellence in lecture/tutorial.

*John Aycock, 11 February 2017*