Part 2 programming

This is the final deliverable for this project. What I'll expect to see here is your fully functional program. Note that you might want to consider re-factoring your code from Part 1 a bit. In particular, you'll need to rebuild the functionality from ExamineState into a broader function for exploring state after state throughout the whole board. And of course, you'll need a runBoard('board.txt') function that loads a board from disk, reports starting board and resulting scores, and otherwise controls the action. What I'll expect to see here are:

- Here is the tests file link.
- When run, your program will begin by showing the board being explored, so we all know what we're working on.
- To give an idea of efficiency, your program should report the time taken to run your code, and the total number of words checked. (from time import time). The time is merely of passing interest, since this will vary by machine (proc. speed, memory, etc). What is really telling is the number of words explored.
- To ease scoring, your output should group found words into groups for 1,2,...X-letters. To further ease correctness checking, your program should then also print the total number of words found and provide an alpha-sorted list of all words.
- OUTPUT: I'm providing a couple of sample boards to show you the desired output and give you some values to verify your program's correctness.
    - Here is a standard 4x4 Boggle board and here is my output file for it.
    - Just for fun, Here is a nice fat 10x10 board, along with the outputs for it. As you can see, it's not even computationally manageable without applying a little cleverness to cut it down to size! The combinatorics are killer here!
    - if you run it through interpreter.py then the output should look like

```
$ python interpreter.py program1_tests2.py

>>> from program1_funs import *

>>> runBoard("board.txt")

Y W A B
Y X I D
Q M D J
P L N A

And we're off!
Running with cleverness  ON
All done
```

```
Searched total of 944 moves in  0.003 seconds

Words found:
2 -letter words:  AI, ID, AN, AD, NA, BA, AB, BI, XI, AX, AW, MI, MY
3 -letter words:  BAD, IMP, AIM, DAW, BID, AND, ADD, DIB, WAX, DID, WAB, DAB, DAN,
MIX, MID, JIB, DIM, MIB, WAD, AID
4 -letter words:  JIMP, WADI, WAXY, WIMP
5 -letter words:  ADMIX, ADDAX

Found  39  words in total.
Alpha-sorted list words:
AB, AD, ADD, ADDAX, ADMIX, AI, AID, AIM, AN, AND, AW, AX, BA, BAD, BI, BID, DAB,
DAN, DAW, DIB, DID, DIM, ID, IMP, JIB, JIMP, MI, MIB, MID, MIX, MY, NA, WAB, WAD,
WADI, WAX, WAXY, WIMP, XI



>>>
```

To ease grading, your program's output should be essentially identical in both format and content to my sample outputs shown above. That means it reports same info in the same order, nice and clean and readable.

Analysis questions: thinking about what your program is showing you about the problem

As explained on our first day, this is NOT a "learning programming" class. This is the upper division, we can assume you all know very well how to program, so let's focus on the real meat. As real computer scientists, creating the code is incidental; what really interests us is using a program to explore the nature of a particular problem. Thus, the following write-up asks you to reflect on your experience in programming the solution, and use it to explore this problem more deeply. In keeping with our emphasis on thinking over programming, the write-up will be worth a substantial amount (up to 50%) of the total points, so be sure to leave time for it in your time planning! Your write-up should be professionally neat, with clearly labeled answers to each of the following:

- 1. A clear description of your algorithm, i.e., the approach/strategy that your code takes in solving the problem. Being able to clearly outline an algorithmic approach is a valuable communication skill in our business, and demonstrates the extent to which you truly understand what you're doing. Do NOT walk through your functions/code in low-level detail! You need to describe how your program solves the problem abstractly, as in the key features of the approach and the steps the program goes through in executing it.
- 2. Answer to the following "thinking" questions. Professionally present your answers to each as clear narrative interspersed with graphs, figures, equations and whatever else you need.

- a. Observe your own results: Run your solver on several 4x4, 3x3, and 2x2 boards. How many different words did your solver explore on each? How much time was taken on each. Now analyze: Come up with a curve showing your results. Then use this to predict the time/moves it would take to explore a 5x5 board.
- b. Analyze the problem generally: How many possible combinations of letters (i.e. actual words or not) can be constructed from an NxN board? Walk through your reasoning carefully, showing how your value comes together. Let's keep it simple just to get a decent upper bound without needing a PhD in combinatorics: Ignore detailed paths possible on the board and just assume that every letter on the board could be chained with every other letter on the board…how many words could be made that way? How does your analysis match up with your empirical findings in the last question?
- c. Use your solver to solve at least 10-20 different boards, then ponder the solution stats you got. Based clearly on your observations, consider the following: Suppose there is a Boggle competition where human players are given a sequence of boards to solve, and the time they have to do so decreases with each board. Now examine the outcomes from the boards that you've run your solver on. What strategy for finding words would a "smart" (or as we'll call it in this course, "rational") player employ to maximize points in a time-limited time? Don't just speculate, support your answer clearly with your empirical results!
- d. In the sample output provided above, there are two runs on the same board shown, with/without "cleverness" turned on…with drastic differences in time/resources used to find identical results. What the heck could that devious Dr. D be doing here to achieve this? Magic? Hint: put in some print statements to watch your program work…and then reflect on the implications of where effort is wasted. The difference between the two outputs in my source code is exactly one line of code…plus another easily-created resource.

Test boards

At this point, your Fred Flintstone solver should be all done and ready to run a few easy tests, producing output as described. To test the performance of your Boggle solver, run your solver on the following test boards (links will be posted near the due date):

- This 2x2 board
- This 3x3 board
- This 4x4 board

For each of your runs, paste the output to a text file and print it out. Please observe the following:

- Output from each run must fit on a single page; only one problem per page.
- Print out, then label each output page clearly: write something like "4x4 testing output" on it front and center in big red marker. Don't make me guess what output this is!
- As noted in the assignment statement, your output should be maximally similar in information and overall format to the sample output. That means each run must show: Total number of moves made (word possibilities explored), the time the run took, and total number of words found. Then it must show 1-, 2-, 3-…X-letter words found, followed by an alpha-sorted list of words.
- Put all your labeled output runs in the order shown above in the proper place within your output packet.

Deliverables

Submit a PDF with (1) Cover sheet: Name, course, assignment title, date, (2) your answers to the analysis questions, (3) output of running your solver on three test boards including prompt, input/output for each command, and a newline between commands, (4) your project1_funs.py source code with comments.

Rubric

70 points total

- 5 points professional presentation / proper English.
- 5 points for descriptions of functions and comments in code.
- 5 points for correctness of each board output (15 total)
- 20 points for quality/effort put into the code (you can get credit here even if your outputs are not correct).
- 5 points for each analysis question (25 total)

FAQ

- Why am I getting error X? Have you tried debugging with pdb.set_trace() and/or conditional breakpoints? For low level debugging issues like this please first ask your classmates and then if that does not help come to office hours.

- Do we have to use “cleverness” ? NO but if you do your program will run a lot faster.
- it mentions to output the “total number of words checked,” but the sample outputs mention only moves (moving from one tile to another I assume). Are words checked and moves the same thing? If not, can I get a clear definition of “words checked”? Yes, every time you move that is another candidate word to check.