Word Games

1 Introduction

This assessment will test your ability to create several word games and analyse their complexity. The two word games are 1) a variant of Hangman and 2) Boggle.

2 Administration

2.1 Deadline

The due time/date for the assessment is **11:59am Friday the 9th of December**. Work submitted up to one day late will attract a penalty of 10%. Work submitted up to one week late will attack a penalty of 25%. Work submitted any later will be counted as a "no paper".

2.2 Submission Instructions

Please upload your assessment into the appropriate section on myAberdeen as a *single* zip file containing two subdirectories; one for the boggle section, and one for the hangman part. Each subdirectory should contain your code (ideally in a `src` subdirectory) for that section, which I must be able to compile, as well as a compiled version of your code (e.g. in a `bin` subdirectory). Your report should include instructions as to how I should run your code.

Your report might need to include some mathematics; if so, you can choose how to include it, writing it out *neatly*, scanning it in and including it as an image in your report is acceptable.

I would prefer the reports in PDF format.

2.3 Suggestions

This assessment will take you a fair amount of time. Please do not leave it to the last week or two, as you will probably not finish on time.

You might not be able to finish all of the assignment, do what you can. Note that the Boggle assignment gives you a choice if you find one approach too hard to implement.

2.4 Marking

The mark breakdown for each component is given in its own section. Note that bad commenting/poor documentation, or poor design of your code will reduce your mark, as will bad spelling, grammar or errors in your report.

2.5 Plagiarism

Please do not plagiarise. While you may discuss the assessment with others, get suggestions etc, please make sure your work is your own. *Please include citations and a bibliography section in your report as appropriate.*

3 Evil Hangman

Evil Hangman[1] is a variant of hangman in which the computer takes a novel approach to winning. More specifically, it cheats. Before introducing evil hangman, let's take a look at the rules of Hangman:

1. One player selects a secret word, and writes out a number of dashes equal to the word length.
2. The other player begins guessing letters. Whenever they guess a letter contained in the hidden word, the first player reveals all appearances of that letter in the word. Otherwise, the guess is wrong.
3. The game ends when either all the letters of the word have been revealed, or when the guesser has run out of guesses (the standard game allows 10 incorrect guesses).
4. If the word is fully revealed, the guesser is the winner, otherwise the player which selected the hidden word (the hangman) wins.

Clearly, it is critical that the hangman accurately represents the word they have chosen. This way, when the other players guess letters, they can reveal whether the letter is in the word. But what if the hangman doesn't do this? Then they're evil, and this leads to Evil Hangman!

As an example, consider a tense moment in the game: 9 guesses have already passed, and the word looks like this:

D O - B L E

There are only 2 words that match this pattern: "doable" and "double". If the hangman isn't evil, then the other player has a fifty-fifty chance of winning by guessing the correct letter. However, if the hangman is evil, then they will always lose. Guessing "A" would lead to the hangman stating that the word was in fact "double", and guessing "U" would lead to them claiming that it was "doable".

Let's illustrate this technique with an example. Suppose that you are playing Hangman and it's your turn to choose a word, which we'll assume is of length four. Rather than committing to a secret word, you instead compile a list of every four letter word in the English language. For simplicity, let's assume that English only has a few four letter words, all of which are reprinted here:

ALLY BETA COOL DEAL ELSE FLEW GOOD HOPE IBEX

Now, suppose that your opponent guesses the letter 'E.' You now need to tell your opponent which letters in the word you've "picked" are E's. Of course, you haven't picked a word, and so you have multiple options about where you reveal the E's. Here's the above word list, with E's highlighted in each word:

ALLY B**E**TA COOL D**E**AL **E**LS**E** FL**E**W GOOD HOP**E** IB**E**X

These words then fall into one of 5 word families:

1. - - - - which contains the words ALLY, COOL and GOOD
2. - E - - containing BETA and DEAL
3. - - E - containing FLEW and IBEX
4. E - - E containing ELSE
5. - - - E containing HOPE

[1] Based on an assessment by Keith Schwarz

Since the letters you reveal have to correspond to some word in your word list, you can choose to re-?? veal any one of the above Tive families. There are many ways to pick which family to reveal per-?? haps you want to steer your opponent toward a smaller family with more obscure words, or toward a larger family in the hopes of keeping your options open. For this assignment, in the interests of simplicity, we'll adopt the latter approach and always choose the largest of the remaining word fam-?? ilies. In this case, it means that you should pick the family - - - -. This reduces your word list down to

ALLY COOL GOOD

and since you didn't reveal any letters, you would tell your opponent that his guess was wrong. Let's see a few more examples of this strategy. Given this three word word list, if your opponent guesses the letter O, then you would break your word list down into two families:

1. - O O - containing COOL and GOOD
2. - - - - containing ALLY

The First of these families is larger than the second, and so you choose it, revealing two O's in the word and reducing your list down to COOL and GOOD.

But what happens if your opponent guesses a letter that doesn't appear anywhere in your word list? For example, what happens if your opponent now guesses 'T'? This isn't a problem. If you try splitting these words apart into word families, you'll Find that there's only one family: the family - - - -containing both COOL and GOOD. Since there is only one word family, it's trivially the largest, and by picking it you'd maintain the word list you already had.

3.1 The Assessment

The assignment requires you to write a program which plays Hangman using this Evil Hangman algorithm. More specifically, you should:

1. Read a dictionary file (e.g. the one used in the boggle task below).
2. Prompt the user for a word length, until they enter a number such that there is at least one word that long in the dictionary.
3. Prompt the user for a number of guesses (which must be an integer greater than 0).
4. Prompt the user for whether they want to have a running total of the number of words remaining in the word list.
5. Play a game of Evil Hangman as follows:
   (a) Construct a list of all words from the dictionary whose length matches the input length.
   (b) Print out how many guesses the user has remaining along with any letters the player has guessed and the current blanked out version of the word. If the user wants to see how many possible words exist, print these out too.
   (c) Ask for a single letter guess (checking it is a legal letter, your input should be case-insensitive).
   (d) Follow the strategy described above to play the game.
   (e) If the player runs out of guesses, pick a word from the word list and display it as the word that was (allegedly) initially chosen.
   (f) If the player (somehow) wins, congratulate them.
   (g) Ask if they want to play again, and do so if necessary.

Your program should be efficient, there should be no noticeable lag between selecting a word and the computer responding.

The algorithm described above is not optimal. Can you think of a better approach? If so, implement it and describe it as part of your submission.

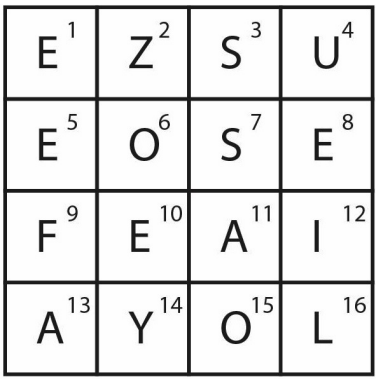

Figure 1: A sample Boggle board.

3.2 Marking

Successfully implementing the evil hangman program will yield you 30%, with marks deducted for poorly documented code, poor design, etc. Having no noticeable lag will give you another 5%, A short report (no more than 3 pages long) describing the basic algorithm in pseudo-code and the data structures you had to use will net you another 10% (poor spelling, insufficient details etc will cause this mark to be reduced). Up to an additional 5% will be given for a description and implementation of extensions to the basic algorithm.

4 Boggle

The game of Boggle[2] is played using a plastic grid of lettered dice, in which players attempt to find words in sequences of adjacent letters. In brief, your task is to write a *perfect super boggle* player, and analyse its performance.

4.1 The Rules of Boggle

The game begins by shaking a covered tray of 25 cubic dice, each with a different letter printed on each of its sides. The dice settle into a 5x5 tray so that only the top letter of each cube is visible. After they have settled into the grid, a timer is started, and all players begin the "main phase" of play.

Each player searches for words that can be constructed from the letters of sequentially adjacent cubes, where "adjacent" cubes are those that neighbour vertically, horizontally, or diagonally. Words must be at least 3 letters long, and include singular and plural (or other derived forms) separately. The same letter cube cannot be used more than once per word. Each player records all the words they find on a sheet of paper. When the timer finishes, the scoring phase commences.

In this phase, all players read out their words, and, if two or more players have the same word, it is removed from all players' lists. Points are then awarded for any words that remain, based on word length.

Word Length	Points	Word Length	Points
3	1	4	1
5	2	6	3
7	5	8+	11

As an example, the 4x4 boggle board shown in Figure 1 contains several words, including "lasso, sole and file". Lasso is made up of dice 16,11,7,3 and 6.

The boggle board contains dice with the following letters. Note that each dice can appear in any point on the boggle board, but can only appear once.

[2]see `http://en.wikipedia.org/wiki/Boggle` for details.

"AAAFRS"	"AAEEEE"	"AAFIRS"	"ADENNN"	"AEEEEM"
"AEEGMU"	"AEGMNN"	"AFIRSY"	"BJKQXZ"	"CCENST"
"CEIILT"	"CEILPT"	"CEIPST"	"DDHNOT"	"DHHLOR"
"DHLNOR"	"DHLNOR"	"EIIITT"	"EMOTTT"	"ENSSSU"
"FIPRSY"	"GORRVW"	"IPRRRY"	"NOOTUW"	"OOOTTU"

Note, The letter "Q" is actually the pair of letters "QU".

I have placed a dictionary available for download on myAberdeen.

4.2 Assessment

4.2.1 Task 1

Construct a program that randomly generates a legal 5x5 boggle board using the dice described above.

4.2.2 Task 2

The simplest (and perhaps dumbest) way of implementing a boggle player is to loop through all n words in the dictionary, and try identify if they are on the board. Write out this algorithm using pseudocode, and, using O notation, describe the time complexity of this approach.

4.2.3 Task 3a

Assume that your boggle program will be run multiple times on different boards. This means that you can put the word list into some sort of data structure which will make searching for the word more efficient, at the cost of initially preprocessing the word list.

Suggest a data structure that can be used to speed up the word search speed, and analyse how long it will take to build this data structure and how much memory it'll use (again, using O notation).

4.2.4 Task 3b

If you are unable to find a data structure suitable for 3a, please suggest a more efficient way of implementing a boggle player than found in Task 2. Describe your suggested approach and show it using pseudo-code, analysing it via O notation.

4.2.5 Task 4

Write a program that reads in a list of words from a file (each word will appear on its own line in this file), and then generates *all* legal boggle words for a boggle board. *Generating* these words should take less than a second if you've solved Task 3a efficiently.

Provide a plot of the running time of your program with increasing dictionary size (i.e. start by using a dictionary with say 100 words, then with 500, 1000 and so on). Can you estimate the average case running time of your algorithm versus dictionary size from this plot?

4.3 Marking

Each task will be marked out of the following:

Task 1	5%
Task 2	10%
Task 3a	10 %
Task 3b	5%
Task 4	25%

As for the Hangman task, poor code, lack of comments and documentation, bad spelling and the like will cause marks to be deducted from each of these components.

Note, you will only receive marks for one of Task 3a or 3b, so don't do both.