Arizona State University

XYZ123 – **Anonymous Course Title**

Assignment #99: Submit on Canvas

All assignments are to be completed and written individually, although general strategies can (and are encouraged) be discussed with anyone. Refer to the Syllabus for more information on Academic Integrity Policies.

No late assignments are accepted. Unlimited submissions are allowed, but only the last complete submission is graded. Submission details are located in the last section of this document, and will be similar on subsequent assignments.

Problem 1 (25 points)

Alphabear[1] is a mobile game in which the player is given a series of letters, each of which has a "timer" (see Figure 1[2]). The player then forms a word based on the available letters, and for the letters not chosen, the timer for those letters decreases by 1. If any letter's timer decreases to 0, then it will be unavailable from that point on. At the end of the game, either there are no possible words that can be formed, or all available letters have been used. Bonuses are awarded based on what bears the player has chosen (each has specific bonuses, such as one uses certain letters, or a percent score increase at the end of the game). The game awards a very large bonus for being able to clear the entire board without having any letters unavailable.

We will be writing a "partial solver" for Alphabear. We cannot create a full solver because when letters are chosen, others appear in a random way (i.e., the characters themselves are random), so we cannot necessarily predict what the best strategy is overall. Nevertheless, we will write a solver for *one turn* in the game; as you will see, it can be extended to an arbitrary number of turns using a loop. For this reason, we won't be doing calculations for bonuses at the end of the game. What we want to accomplish here is to find the "best" word at the current turn based on what is visible.

Suppose a visible letter on the board c, has a current timer value of t (we assume that the timer is always a positive integer). Then the *urgency* of c is $1/t$. In other words, c has higher urgency if the timer is lower. The Alphabear game loves words that are longer, so we want our model of what the best word is to factor this into consideration. Let $x_1x_2\cdots x_n$ be an English word, and has corresponding timer values $t_1, t_2, \cdots, t_n$ on the board. Then the urgency of this word is:

$$(1/t_1 + 1/t_2 + \cdots + 1/t_n) \times 1.1^n.$$

The urgency of an English word, with letters $x_1 \cdots x_n$, is the sum of the urgencies of $x_1, x_2, \cdots, x_n$. For example, if ah is the considered word, a has an timer of 1, and h has a timer of 2, then the urgency of ah is $(1 + 1/2) \times 1.1^2 \approx 1.815$.

Therefore, we want to find any English word of highest urgency. **We will assume that each character on the board appears at most once.** For this reason, you will only consider words from the given dictionary that have no duplicate letters.

Note that if we want to consider an English word, all of its letters must be usable by the board. For instance, in Figure 1, *hit* can be chosen (since all three letters are visible), whereas *hunt* is not (assuming n is not visible anywhere else).

[1] `http://spryfox.com/our-games/alphabear/`

[2] Yes, it is extremely adorable.

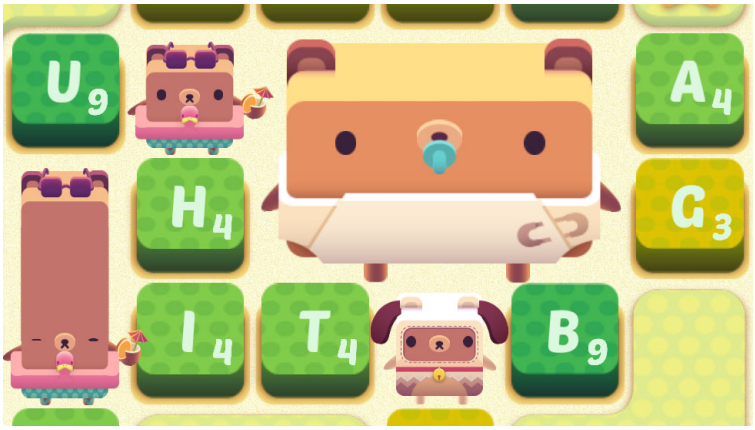

Figure 1: Alphabear gameplay, `https://patricktay.files.wordpress.com/2015/09/alphabear1.jpg`.

All English words will be provided next to this PDF on the course webpage. The file has one English word per line (i.e., separated by a newline character). You will have to research how to read a file line-by-line (or all at once) in `C++`, but a good resource is here:

`https://stackoverflow.com/a/7868998/3232207`.

Of course, since there are so many English words, it would be impractical to print *all* valid words. So, you will proceed as follows:

- Write code to read from the given file. **Make sure you don't rename the file, and that it is in the same folder as the `main.cpp` file you will make.**
- Read from the user (until the user enters -1 for the timer) a character, along with its corresponding timer. You can assume that the user will enter everything correctly here, only the 26 lowercase letters can be inputted as the character, and all of the timers are at least 1.

  To store the letters you read, you can use any technique you wish (including creating many variables for all possible letters), but it is recommended you look at `std::map`, which stores key-value pairs (hint: it ties the character and timer together).
- Read through the file word-by-word. (Hint: check that your program can do this first before tackling all the other parts).
- Check if the considered word has no duplicate letters; if so, continue to the next step. Otherwise, check the next word.
- For each word, if it is compatible with the letters the user entered, then calculate its urgency. You can loop over a `std::string` as follows:

  ```
  string s = ...;
  for (char ch : s) {
      // process character ch
  }
  ```

  Or if you want to use an index:

```
string s = ...;
for (int idx=0; idx < s.length(); idx++) {
    char ch = s[idx];
    // process character ch
}
```

- Maintain which word seen so far has the highest urgency (how would you do this?)
- Output to the user the single word which has the highest urgency. **If there is a tie, you will output the "earliest" word in dictionary order.**

As always, sprinkle your program with helpful comments throughout, as well as if you use any functions not discussed during class.

Submission

You will turn the `cpp` file and the `txt` file to Canvas at the appropriate submission link. The source file must be in the format `a99-studentid.cpp`. For example, if my ID was `anonymous`, I would submit the source file `a99-anonymous.cpp`.