SpellMaster Report

Bitbusters

Yara Al Tassi – Hanan Zwaihed – Yasmin Halabi

2023 - 11 - 12

CMPS 270

https://github.com/yarasrepo/Bitbusters_CMPS270.git

Spellmaster is a game that can be played either with a bot, or with another person. It accepts a file containing words and tosses a coin to see which player will start. In alternating turns, the players each need to choose a word starting with the last letter of the previous word. The word must also NOT have been used before and MUST exist in the file.

Implementation:

Driver:

The program begins by calling “fileReading” which prompts the user for a file name and processes the words into a charmap. It then welcomes the player and gives them the option to choose between playing against a bot or another player. If the player chooses to play against the bot then they can choose between easy, medium, and hard modes of play.

It will then toss a coin that determines who will play first. This is implemented using a random number % 2 to, with 0 representing player 1, and 1 representing bot (or player 2). This is to ensure 100% randomness.

If the bot is chosen first, it will pick the first word according to “easy” mode, regardless of whether the player chose something different. This is to ensure that the game does not end before the player gets a chance to play against the bot. The game will continue playing according to the player’s chosen difficulty. (easy mode provides the extra perk of being able to see what words have been chosen so far).

When it is the player’s turn, they can choose a word that will then have to pass several conditions.

1. The word is in the list (of words from the file). This is done by calling a charMap function “isInCharMap”.

2. The word has not been used before. This is done by keeping track of every used word in a Set called usedWords, and checking if that set contains the chosen word.

3. The word starts with the same letter that the word before ends with. Done by keeping track of the last used word and updating every iteration.

These conditions would apply to both Player 1 and Player 2 if the player chose the 2 player option instead of playing with the bot. If the bot option is chosen, then these three conditions are not checked since is it is assumed that the bot can choose the word correctly according to its difficulty level.

As soon as one of these conditions fails, the game ends and the bot (or other player) wins. The program then prints the reason why the game ended.

Another condition that is checked is whether the word chosen's last letter has any available words starting with it. If it does not, then the player (or bot) who chose the word automatically wins.

We created two data structures: Set and Charmap to store the words in the file, and make them easily accessible

- **Set** is a structure that mimics the implementation of the built-in Set that can be found in other programming languages such as Java and C++. It is implemented using nodes and next pointers similar to a linked list, with the extra condition being that a word cannot be repeated within the set.

- **CharMap** contains key/ value pairs; the keys represent the characters from a-z and the values are sets of words starting with the key. By using these structures, we facilitated the quick search of any word in the file and ensured an efficient time complexity so it can handle files of all sizes. This structure works similarly to a hashtable which uses chaining for collision handling.

Bot class:

Our bot class named "spellmaster" consists of three functions that can select a word from the list according to three difficulties: easy, medium, and hard.

- The easy function finds the word that ends in a character that maps to the MOST available words. This gives the player the best chance of winning, since it gives him the most options to choose from.

- The hard function finds the word that ends in a character that maps to the LEAST available words. This gives the player the smallest chance of winning, since it gives him the least options to choose from.

- The medium function finds the word that ends in a character that maps to the MEDIAN number of words (as in the size of the set falls in the middle of the sorted list from least to most available words).

All three difficulties perform better than random.