Lexiko - Scrabble Player

Pratik Mistry[180050077] Vrinda Jindal[180050120] Yash Gupta[180050121]

This is a racket implementation of the classic Scrabble Game.
Unlike games like chess, checkers, tic-tac-toe..etc. luck and outside knowledge play a role. Even a perfect Scrabble player may still lose due to unfortunate letters or an unlikely countermove.

Program Design

❖ Algorithm

This algorithm, inspired from the internet, incorporates significant modifications from our side. The brute-force approach (without any optimizations and heuristics) is described as follows:

- Firstly, the board is represented as a vector of vectors (each containing instances of 'tile' struct).
- Secondly, we take as input a 15*15 board with some arrangement of letters. This forms the input board.

Here's how it determines the best move in the space of the possible move:

1. First, it creates a list of provisional tiles (unoccupied tiles near an occupied tile). For example, If this is the first turn we only have one (7,7).
2. For each provisional tile, we run the program for horizontal and vertical directions.
3. For each direction, we create a list which stores lists of tiles to place. We do this by expanding along left and right (up and down) to get all combinations of time slots used. Growth automatically skips pre-played tiles and doesn't add placements which have tile slots out of bounds.
4. For each set in the list, we run through all permutations of possible tiles to place in each slot. For each of these, we run a board validation routine which tells us whether the move is correct and what score will result.

❖ Functions

- ValidateBoard - Given a board with unlocked (just played) and locked tiles (played at least 1 move ago) this function checks the correct placement of unlocked tiles (in a row or column only, whether a crossword forms...etc.) and checks whether each possible word belongs to the scrabble dictionary. This function has been reused many times in both the human and the computer play.
- validateWords - Validates words present on the board for correctness and calculates total board score.
- ai - The heart of the computer play. Given an input board, it goes through all possible moves and tries to find the highest scoring word.
- is-valid? - determines if a word is present in the official scrabble dictionary.
- get-score - scores the word taking into account TWS (triple letter score), DWS (Double Word Score), DLS(Double Letter Score), TLS(Triple Letter Score).

❖ HIGHER ORDER ABSTRACTIONS

- ValidateWords/ValidateBoard - the same set of functions have been differently used depending upon whether they are called upon after the computer's move or player's move.
- Build from a seed tile - in building tiles to play from, different directions to build (right, left, up, down) is executed.

❖ OBJECT ORIENTED DESIGN

- Structs: use of mutable and transparent structs to represent game structure.

1. Tile struct - has pts, letter, blank-status, locked-status, multiplier as its attributes.
2. Board struct - has a 2d 15*15 vector of tiles as its primary attribute.

❖ UTILITY FUNCTIONS AND MACROS

- **Copying the code structure of racket built-in functions for lists,** several functions for vectors have been implemented. vector-append*,vector-findf, vector-flatten, vector-remove-duplicates etc. were developed.
- Macros used : while, for, for/vector, for/list, etc.

❖ PACKAGES USED

- racket/gui: for all interfaces of the game.
- 2htdp/io : for processing dictionary.
- racket/vector: for dealing with vectors.

❖ DEBUGGING

Debugging: We created our own printing style of the board and functions to display tile-slots and tried-words conveniently.

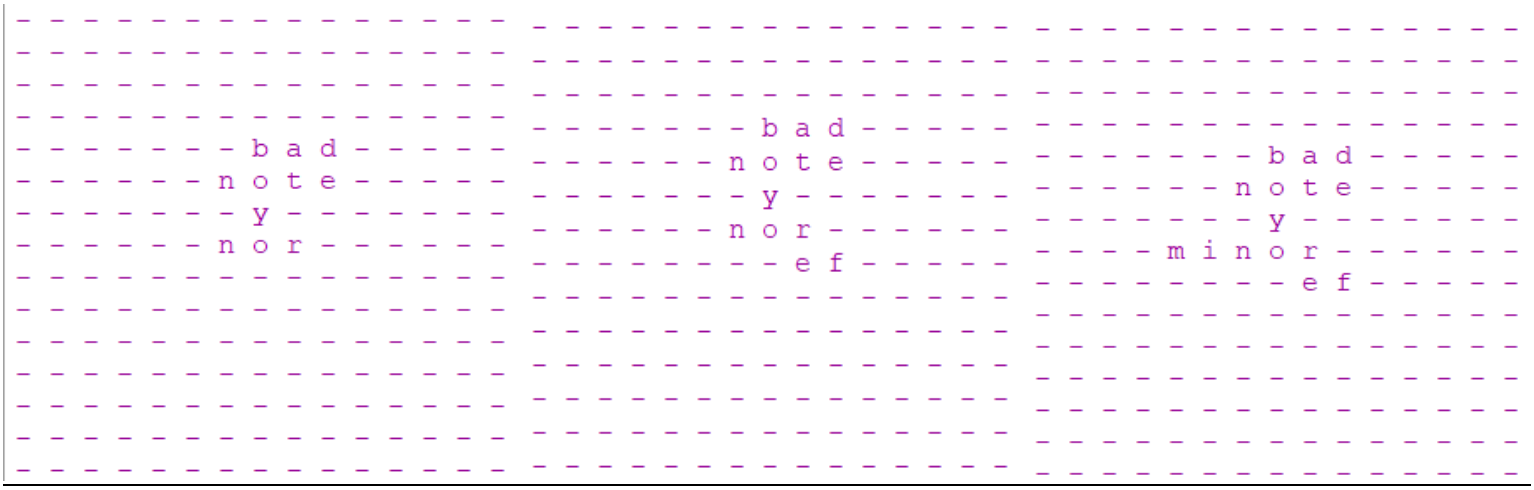
❖ GUI AND FEATURES

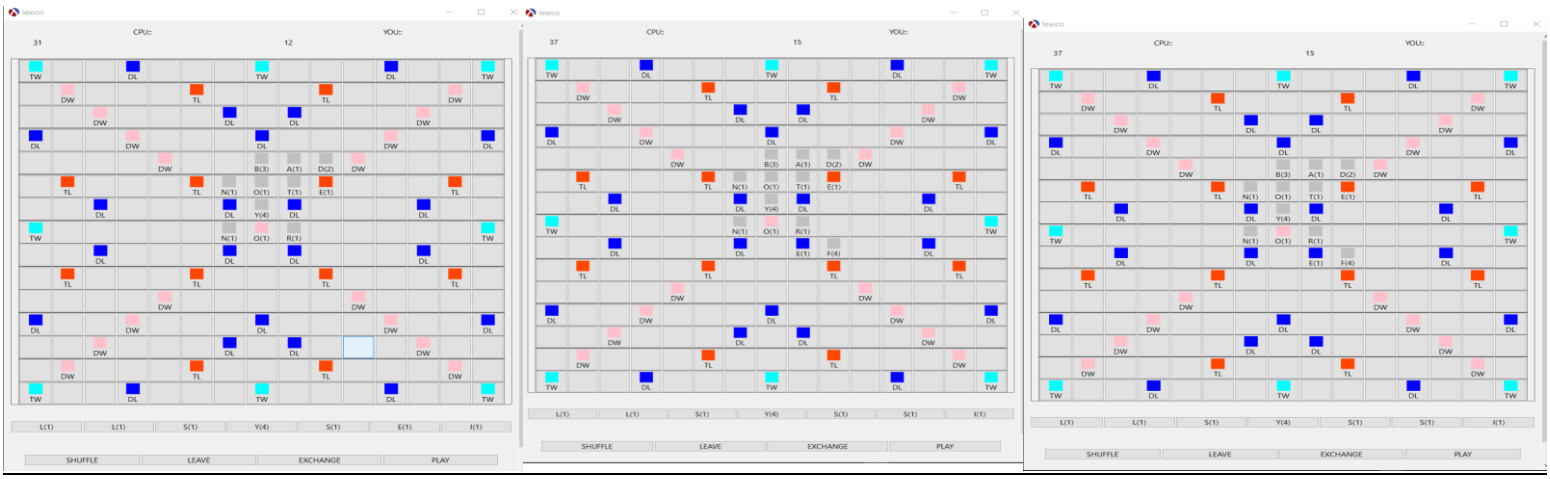
- Undo: Places the tiles back on the tray if player unintentionally places the tiles on the board.
- Exchange: On the cost of one turn, player can exchange his hand with the CPU if he is at a loss of words :)
- Shuffle: Shuffles the tray so our neurons can relax a bit and don't have to juggle between letters of the tray.
- Play: Plays the tiles the user played on the board and CPU plays in succession.
- Leave: Evaluates the winner on the basis of current score and ends the game.
- The board is an arrangement of 15X15 buttons, with a bitmap to represent the special tile scoring (TWS, DWS, DLS,TLS represented in the key) and a label. The tiles when placed on the board by the player are stored as unlocked tiles in the board-in-play which gives the state of the board.

Sample IO

- Input - Will be a board with some of the tiles filled and some empty. This board already satisfies the board validation routine.

❖ Output - Updated board with the highest scoring word played and scores of board updated.





Input-Output Samples (Command Line and GUI Versions)

Limitations and bugs

1. **Our program initially took an average of 1.5-2 minutes per move.** A game like Scrabble is not a game of perfect information because there's no way to predict your opponent's moves because you can't see his hand. So an algorithm like minimax cannot be used. For faster results, non-trivial data structures (dawg and trie) were tried but could not be implemented due to complexity and shortage of time. In our brute-force strategy where we enumerate all possible moves, **the main inefficiency was due to access of dictionary vector (length 1,78,717) hundreds of times** on a single call.
2. In the straightforward strategy, the greedy players place the longest word on each turn that earns the highest score. **Our program essentially employs this strategy.** A drawback of this method is that it can generate a lot of hot spots for the opponent. These hot spots on the game board let the competitors place a high score move in the next turn. The highest scoring move is not always the best one.
3. Blank Tiles have not been enabled. They were abandoned as they significantly increased the time taken to arrive at the best move.

Points of Interest

1.To go around the problem of inefficiency :

a) We took a dictionary sorted by the usage-frequency of a word.

b) We created a different dictionary using first 15000 words from this dictionary. We appended all possible 2 letter words to this dictionary. These are very high probable words which tend to save us from arriving at a deadlock due to a shorter dictionary.(2 letter words can be played almost anywhere.)

c) Now we use this dictionary for enumeration of all possible moves and the original dictionary only for validating the board.

d)This brought the runtime down to 15-20 seconds.

2. Pass by reference in Racket - We maintain a bag of tiles for undrawn tiles. To update it every time a function is called, box structures are used. (which are references rather than values).

3. Heuristics: An intelligent AI player must efficiently apply different strategies and heuristics at each turn depending upon the current board state. Due to a short dictionary, we can now conveniently search for the best word among the reduced sample space. Also use of 2 letter words is quite strategic as they generate very few hotspots for the next move.
4. Use of States: State of the program is given by player-tray, computer-tray and Board comprising of locked and unlocked tiles.