An interactive clone of the board game Scrabble featuring smart computer play and support for multiple players and languages

Student: Simeon Dobrudzhanski & Supervisor: David Lonie

Introduction

Scrabble is one of the oldest word-games available on the market, created by Alfred Mosher Butts from Poughkeepsie, New York in 1933 and its original name was LEXIKO. Its name was changed to the current one in 1948 and since then, Scrabble has grown so much in popularity that it is now available in more than 60 languages. The aim of the game is to create words with a set of given letters on a board in a way that each word on the board is connected to a different word and is valid either in a horizontal or vertical play. Many digital implementations of the game have been made, some focused on interactive graphics, some on customizability and some on very intelligent computer play.

Project Aim

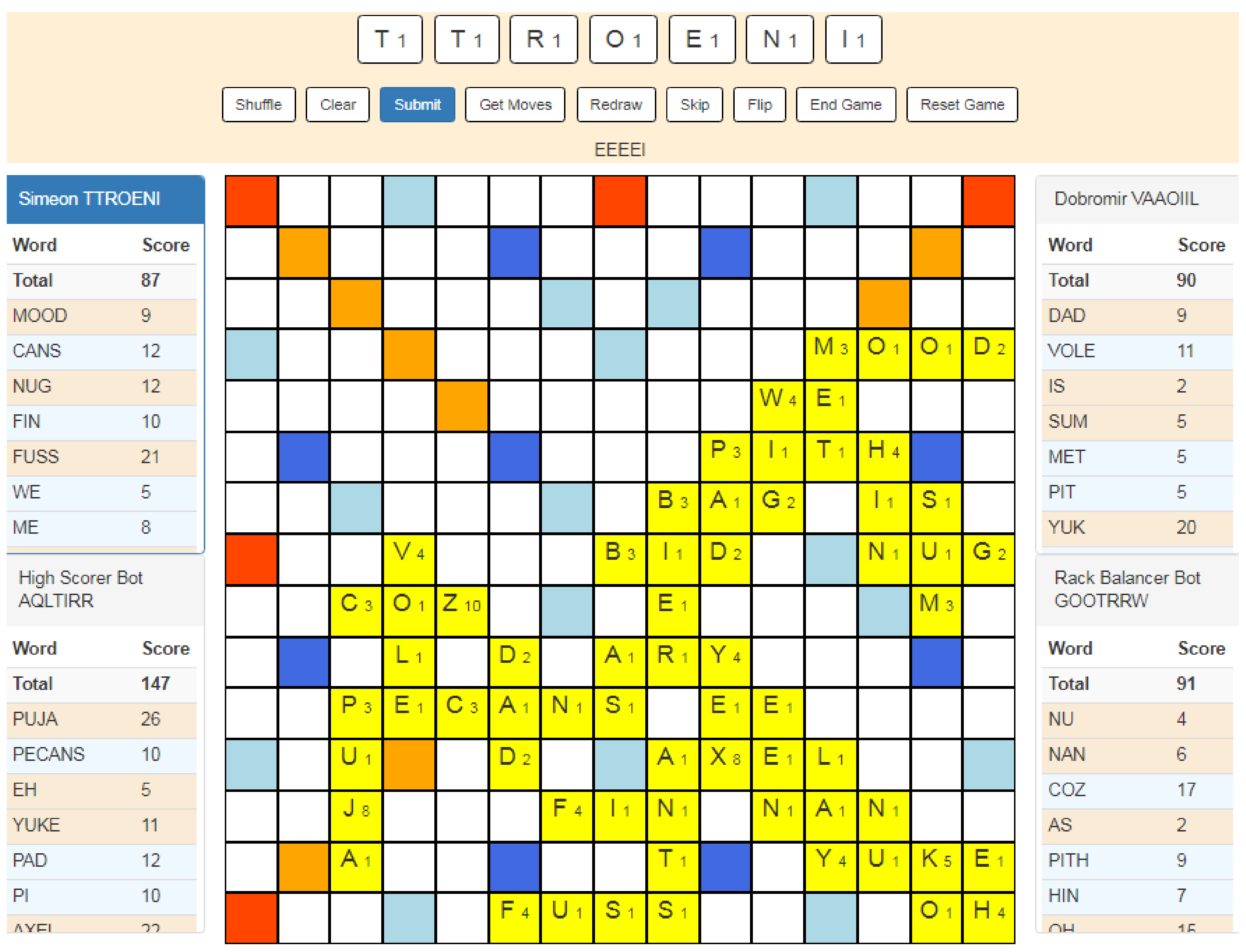
The aim is to provide a different implementation of the game with its own flair. Graphics need to be clean and responsive for different displays. Customizability needs to support multiple players and languages. And computer play should make valid moves and its strength should be adjustable.

Technologies Used

The project was developed as an ASP.NET Core web application written in C#, following the MVC pattern and developed in Visual Studio 2017. EF Core was used to map objects to a database stored in a local MS SQL Server generated by IIS Express. The front-end was designed using standard HTML/CSS/JS with the help of jQuery and Bootstrap. Git was used for version control. NuGet packages include lazy-loading and specialized data structures.

Figures and Results

Below is an example of a game in progress with four players – two humans and two bots. High Scorer Bot plays the highest scoring word it has come up with in a limited time frame. Score calculation takes into account the usage of premium tiles as well as extra words that can be played with a given move, Rack Balancer Bot aims to always have a healthy balance of vowels and consonants left in its rack. This balance is calculated by assigning a score to each letter in its rack and a negative score for its duplicates in the rack. Afterwards an equation is performed (taken from previous research papers) and the result of the equation is used as a heuristic to pick the move that would result in the best rack. This bot is only available in the English version of the game.



Simeon TTROENI

Word	Score
Total	87
MOOD	9
CANS	12
NUG	12
FIN	10
FUSS	21
WE	5
ME	8

High Scorer Bot AQLTIRR

Word	Score
Total	147
PUJA	26
PECANS	10
EH	5
YUKE	11
PAD	12
PI	10
AXEL	22

Dobromir VAAOIIL

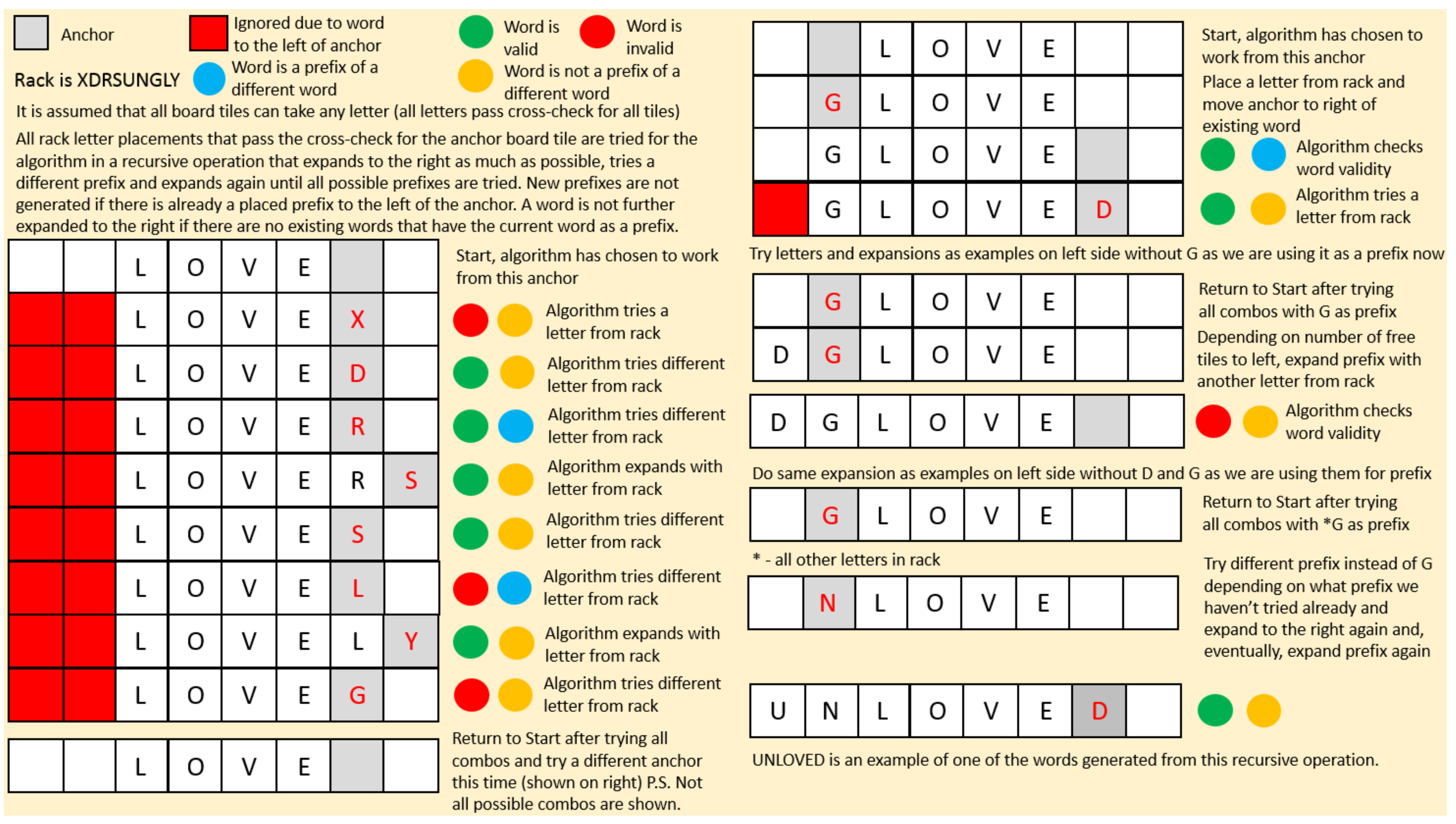
Word	Score
Total	90
DAD	9
VOLE	11
IS	2
SUM	5
MET	5
PIT	5
YUK	20

Rack Balancer Bot GOOTRRW

Word	Score
Total	91
NU	4
NAN	6
COZ	17
AS	2
PITH	9
HIN	7
OH	15

Bot Implementation

If the active player is a bot, the server uses a set of steps to get a collection of possible moves. The steps include determining the anchors (the empty tiles where a word can be connected), calculating the cross-checks for each tile (the set of letters that can be placed on a given board tile that would pass both horizontal and vertical checks) and then generating every possible right part (letters right of and including the anchor) of a word for every possible left part (letters preceding anchor).


Anchor
Ignored due to word to the left of anchor
Word is valid
Word is invalid
Word is a prefix of a different word
Word is not a prefix of a different word
L O V E
LOVEX
LOVED
LOVER
LOVERS
LOVES
LOVEL
LOVELY
LOVEG
GLOVE
GLOVED
DGLOVE
NLOVE
UNLOVED

Rack is XDRSUNGLY

It is assumed that all board tiles can take any letter (all letters pass cross-check for all tiles)

All rack letter placements that pass the cross-check for the anchor board tile are tried for the algorithm in a recursive operation that expands to the right as much as possible, tries a different prefix and expands again until all possible prefixes are tried. New prefixes are not generated if there is already a placed prefix to the left of the anchor. A word is not further expanded to the right if there are no existing words that have the current word as a prefix.

Start, algorithm has chosen to work from this anchor

Algorithm tries a letter from rack

Algorithm tries different letter from rack

Algorithm tries different letter from rack

Algorithm expands with letter from rack

Algorithm tries different letter from rack

Algorithm tries different letter from rack

Algorithm expands with letter from rack

Algorithm tries different letter from rack

Return to Start after trying all combos and try a different anchor this time (shown on right) P.S. Not all possible combos are shown.

Start, algorithm has chosen to work from this anchor

Place a letter from rack and move anchor to right of existing word

Algorithm checks word validity

Algorithm tries a letter from rack

Try letters and expansions as examples on left side without G as we are using it as a prefix now

Return to Start after trying all combos with G as prefix

Depending on number of free tiles to left, expand prefix with another letter from rack

Algorithm checks word validity

Do same expansion as examples on left side without D and G as we are using them for prefix

Return to Start after trying all combos with *G as prefix

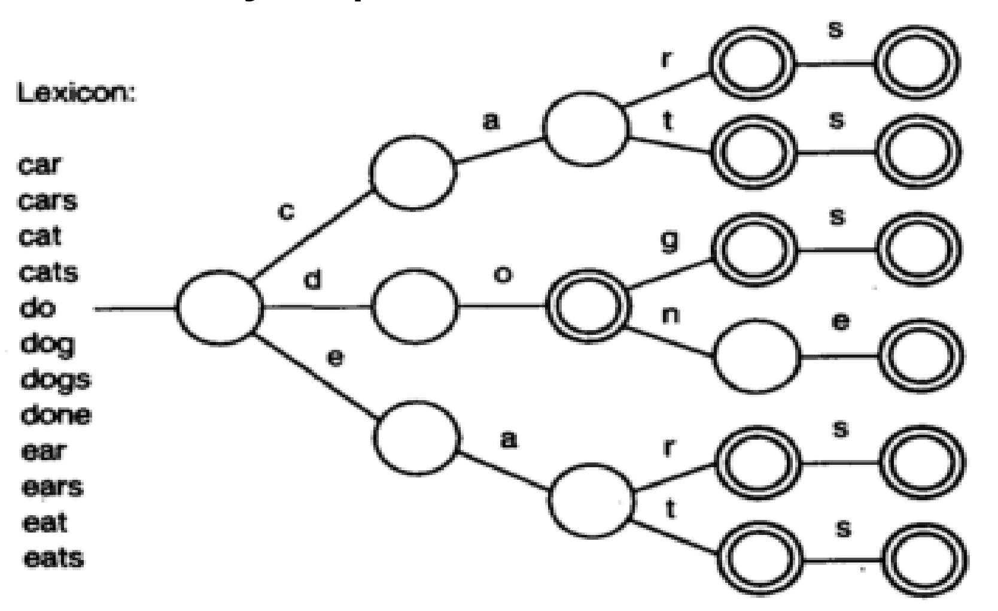
* - all other letters in rack

Try different prefix instead of G depending on what prefix we haven't tried already and expand to the right again and, eventually, expand prefix again

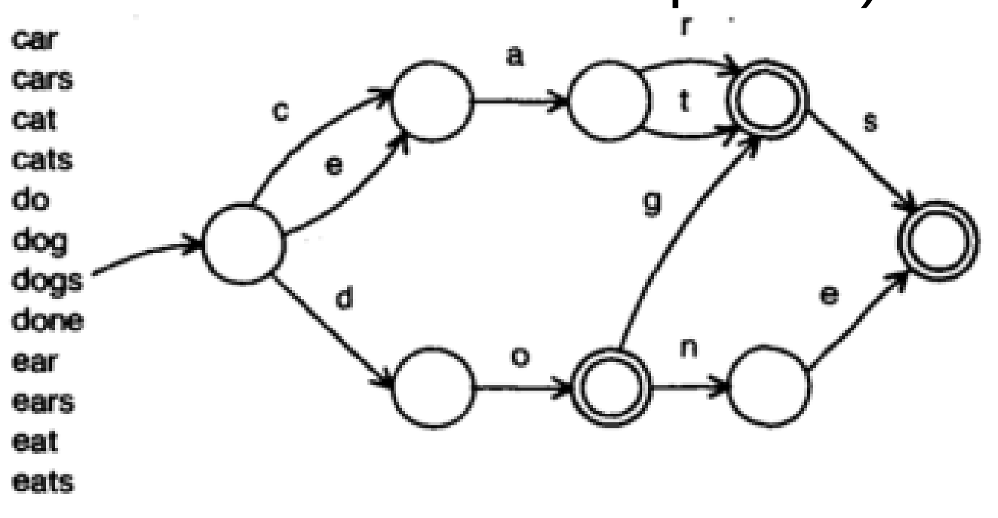
UNLOVED is an example of one of the words generated from this recursive operation.

Dictionary Implementation

Below is an example of a dictionary represented as a trie.



In this project, the dictionary is implemented as a DAWG (Directed Acyclic Word Graph). It is a compressed version of the trie (tree-like data structure) in which repeating states are merged together. The condition for repetition is that two states have the same continuation and must contain valid words. The DAWG is shown below. Asking the DAWG if a word is valid or is a valid prefix offers O(n) performance (n being the size of the looked up word).



Conclusion

The project has managed to achieve the goals from "Project Aim". Future work will focus on optimizing the speed and intelligence of computer move generation and the exploration of different frameworks and libraries for front-end and back-end design.

Acknowledgments

I would like to thank my supervisor David Lonie and Roger McDermott for their guidance in the project development and project report writing.

References

The World's Fastest Data Structures - *APPEL* , JACOBSON

A COMPARISON BETWEEN PROBABILISTIC SEARCH AND WEIGHTED HEURISTICS - Steven Gordon

ROBERT GORDON UNIVERSITY ABERDEEN
SCHOOL OF COMPUTING SCIENCE AND DIGITAL MEDIA

BSc (Hons) Computer Science