Honours Project Report

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

Simeon Dobrudzhanski, 1406444, April 2019

I confirm that the work contained in this Honours project report has been composed solely by myself and has not been accepted in any previous application for a degree. All sources of information have been specifically BSc (Hons) – Project Guidelines School of Computing Science and Digital Media Page 15 September 2018 acknowledged and all verbatim extracts are distinguished by quotation marks.

Signed:  **Date: 29/04/2019**

1. Abstract

This report covers the design, implementation, testing and evaluation of a digital clone of the board game Scrabble. The project has been developed as a web application and has been completed through the research of the board game's history and rules and through the evaluation and appreciation of other existing implementations as well as various algorithms. The algorithms are related to creating an efficient data structure for the game's word dictionary, to developing an intelligent computer player and to providing a usable front-end design and solid back-end architecture for the solution. The information gathered from the research of these algorithms has been used to provide a working version of them and presenting them through the use of modern technologies.

Contents

2. Introduction

The primary objective of this project is to implement the Scrabble word-game, capable of being played by two or more human or intelligent computer players. An implementation of such a game will be attempted, based on which further customization will be attempted such as the size of the playing board and the arrangement of its tiles, list of dictionaries, total number of players, scoring per letter and other potential features. The game's scoring is dependent on the language of the dictionary being used. The report will mainly be covering the English set of rules and scoring unless otherwise stated. The main question to answer is can the game be re-implemented with new technologies given current research, implementations and techniques available and can it be improved using the available information?

Important Notice: By the time of writing this report, the project has already been finished or will see minimal changes in the future. The only section of this report that is written before the start of the development phase is the Literature Review and the Requirement Analysis. It will be left in this state to provide a better narrative for the reader and this way, the reader will have an easier time following the though process of behind the development

The report is split into a few main sections which are explained below:

Literature Review – The purpose of the review is to present to the reader relevant information that is important to understanding the thought process behind the development of the project. As an introduction of its own, the review shows findings about the game's origins, rules and potential legal issues that may arise from undertaking such a project and legal issues that have arisen from similar undertakings. The main topics of the review then dive into details about the main challenges and algorithms of implementing the game and the evaluation and discussion of techniques written in scientific papers by other scholars. Finally, similar projects such as this one are discussed with the goal of collecting new ideas about the development of this project and to become aware of the possible tools to be used for developing it.

Design & Methodology – This section will cover in detail the tools and algorithms used for the implementation of the project. The reader will be presented with an introduction for each piece of the project that plays a key role in the development phase. In addition, the environment created by the usage of the tools will be presented to the reader and will contain discussions about why a given approach to the solution was chosen. The requirements of the project will be reviewed. In summary, the goal of the section is to help the reader become familiar with the tools used, the reasoning behind all of the development choices and a narrated way of presenting the key pieces and context/environment. With a successful exposition of these elements, the reader should have minimal trouble understanding the following sections of the report.

Implementation – This section will describe the process of utilizing the key pieces described in the Design & Methodology section to achieve the final product. The development of extra features such as computer play will also be covered.

Testing & Results – With the final product finished, testing of each feature will be provided. Testing will cover individual features and a couple of full games.

Evaluation – In this section, the project will be evaluated if it has met or not met the original aims and requirements of the project. Unimplemented features will also be reviewed and future improvements will be discussed.

3. Literature Review

3.1 History

Scrabble is one of the oldest word-games available on the market, created by Alfred Mosher Butts from Poughkeepsie, New York in 1933 [23]. Word games such as crosswords puzzles and anagrams did not have a scoring system back then which, he believed, kept them back from being as popular as, for example, card games. He came up with Scrabble, which was then called LEXIKO and in later years, CRISS CROSS words.

Letter distribution was calculated by Butts by studying the front pages of "The New York Times", or so legend says, and that distribution has not changed since then. Butts' game was rejected by big game manufacturers until his idea was accepted by entrepreneur James Brunot and that was when the game received its final name Scrabble and trademarked in 1948.

The first couple of years were a struggle for the Brunots as the company made 2400 sets and lost $450 but popularity was slowly rising. It's rumoured that the president of MACY's learned of the game and bought it for himself, causing its popularity to explode. Since then, Scrabble has had a change of ownership but has never lost popularity. Below is an example of a Scrabble game in process.

Figure 1. Example of an official Scrabble game. AG is the first-word played and over the course of the game, other words have been attached to AG and other words to form additional words.

Approximately 150 million sets have been sold worldwide and there are around 4000 clubs around the world [26]. The figures show that the game is in great demand in its physical form. As for its digital form, many implementations exist that each bring new features and techniques for smart computer play. Commercial software for Scrabble is greatly sought after as well as, in a paper by Mark Richards and Eyal Amir [7], commercial software CrossWise generated US $3 million in sales.

The next section will be covering most of the rules for a casual game as tournament games have slight modifications which will be an important topic in future sections.

3.2 Rules

Taken from the Scrabble Hasbro web-page [24]. The game is played on a 15x15 board, some tiles of which contain special abbreviations such as a DL, TL, DW, TW (double-letter, triple-letter, double-word, triple word) which will be referred to as premium tiles.

- The game begins with a 100 tiles placed in a pouch. The pouch contains 27 types of tiles, 26 being each letter of the English alphabet and the 27th being a blank tile which may act as any letter.
- Each type of tile may be repeated a number of times, for example the letter E is to be found on 12 tiles while the letter Q is to be found on only 1 tile. Each tile has a number associated with it, it representing the score increase for that type if a player plays it in a legal move. Blanks tiles are worth 0 points.
- Each player is to randomly draw an equal number of tiles from the pouch, that number usually being 7. Any tiles that a player has are stored in the player's rack. The tiles left after a player plays a move is called a rack leave.
- The starting player is to make a valid word by placing the appropriate tiles in the starting position of the board, that being the star on the centre of the board. The word's validity is checked with the word dictionary before being played out (in tournament play, illegal moves may be called out by the opposing player for a score advantage).
- Words may only be formed in a horizontal or vertical manner. After a play, each tile of the winning word is counter towards a move's score which is also added to a player's total score. If a word is played on premium tiles, the score for that word or individual letter may be increased.
- Following the very first play of the game, each following word must be attached to a tile that is already on the board. The attached-to tile may be any part of the word (beginning, middle and end) as long as it is part of a valid word. A valid play also requires that the played tiles form valid words with any adjacent tiles they are connected to.
- For each played tile out of a player's rack, the player needs to draw the same count of tiles from the pouch. Once the pouch is empty, the players may only continue with the tiles that they have left. A player may also use their turn to redraw a number of tiles, that number being equal to the tiles that they wish to give up.
- Playing all seven tiles in a full rack in a single move results in a Bingo which is worth an additional 50 points for the player.
- The game ends when a player has played all of their tiles without any in the pouch left or when both players skip their turn (due to inability to play a word).

Before discussing research and implementation, it is important to consider what legal issues may arise and have risen before which is in the next section.

3.3 Legal Issues

According to American law, rules of a game are not copyrightable but the artistic expression of the game is [22]. In addition, protection against copying of game mechanics is difficult as there are no legal mechanisms to provide complete protection from cloning [18]. Copyright protection is the most

common defence and it protects the expressive features of the game i.e. visual elements. A popular story is the one of game Scrabulous documented below.

In the summer of 2008, the popular Facebook game Scrabulous, clone of Scrabble, was shut down by Hasbro and Mattel, owners of the Scrabble game [16, 17]. The game was among the top 10 most played games on Facebook and attracted over 500,000 users every day. It was shut down due to it supposedly violating the companies' copyrights on the game. Hasbro and Mattel partnered with companies EA and Real Networks to develop versions of the game for the social media platform. They have not been as successful at attracting as many users as Scrabulous, due to various factors - SCRABBLE® Worldwide could not be played within the U.S. and Canada while Scrabble Beta had technical faults that kept it from getting as popular. Since then the Agarwalla brothers, original developers of Scrabulous, have developed a different clone of the game named Lexulous which is supposedly safe from copyright violations as it is significantly different regarding rules, scoring, assets, and art styles etc. – anything that doesn't pick up from Scrabble's artistic expression.

An interesting comparison can be made to current very popular Scrabble-like game Words With Friends [20]. Speculations as to why the WWF creators have not been sued yet is due to their game being different enough although the core experience is the same. Another speculation is that WWF creators Zynga have partnered with Hasbro to sell the game but there is no concrete proof for this notion. Nonetheless, it is evident below that Scrabulous's board (on the left in Figure 2) resembles that of Hasbro's Scrabble a lot closer than the one that Words With Friends uses (on the right in Figure 2) in regards to the placement of premium tiles.

Figure 2. Scrabulous board (left) vs Words With Friends board (right)

In September 2014 Slate reported that Hasbro has been attempting to find and take down any word lists of the game which have not been licensed by them [15]. Digital word lists have been popular ever since Brian Sheppard's implementation of Maven (covered in section 8), the best AI player developed and bought by Hasbro, was presented to the world. As Maven works with a digital list as it is not a human player, various clones of Maven implemented by keen players of the game have

used similar lists. These lists have been man-made and distributed freely on the internet. Hasbro has argued that since the lists are a variation of their own official OSPD (Official Scrabble Players Dictionary) list, they have the right to claim copyright on them. Hasbro has stated that their actions are with the intention of democratizing the game and making sure as many people as possible have access to these word lists. Players have argued that these actions are not democratizing but restrictive and that the freely available word lists have been created as a "labour of love" and find it and that Hasbro has no right to try and monetize work which has never been theirs originally.

It is hard to judge when a game may have broken copyright laws. Words With Friends implements the same mechanics as Scrabble but plays them out differently and visually is slightly different which supposedly has helped it survive. This implementation will likely not be going commercial as it is a personal project. If there is a notion of commercialization, professional advice will be sought. The next section will give a summary of the main challenges in implementation.

3.4 Challenges

One of the main challenges for games is developing a capable AI opponent. In the history of the game there has been research from various scholars and, with that knowledge publicly available, an AI has been developed which is yet to be beaten in a world-class tournament setting called Maven [4]. It is worth looking into what techniques Maven uses for creating such a challenge to human players and to consider which ones can be imported into the current project.

Another challenge is to look into research which has covered the best algorithms for validating word plays and deducing moves. Incorrect mechanisms may lead to a state where an AI agent might miss many openings, play low-scoring moves or take too long to reach a decision. There are innumerable ways a Scrabble game may play out thus it is important for an agent to have enough freedom to explore different possibilities and make smart decisions and enough constrictions to allow fast decision making and gameplay. Maven uses a variation of existing well-researched algorithms which form the basis of most existing implementations and will be covered in the next sections.

3.5 Appel and Jacobson's Algorithm

A major breakthrough in designing an AI for Scrabble is documented in the paper published by Appel and Jacobson [3]. The scholars' objective was to develop an efficient backtracking algorithm with the goal to create a very fast program capable of playing the Scrabble game. One of the program's main features was the DAWG (Directed Acyclic Word Graph).

One of the challenges of the algorithm is to store the lexicon of the game in memory. For their project, Appel and Jacobson used a 94,240-word lexicon. One way they managed to implement it was in a 117,150-node trie (trie is a tree-like data structure in computer science and, as cited in the paper and due to the way it is used, can also be called a letter-tree) . Each letter has a starting node and separate branches are connected to those letters for each different continuation (figure below).

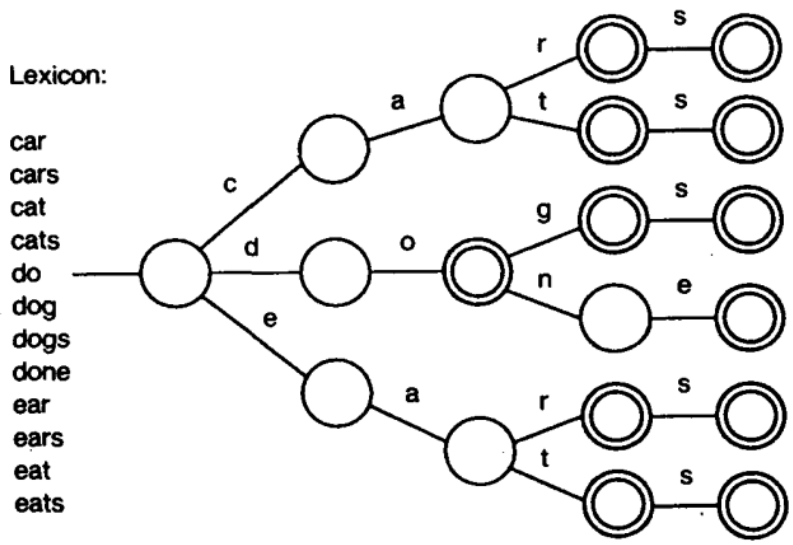


Figure 3. A lexicon and the corresponding trie. The circled nodes are terminal nodes i.e. they represent the end of a valid word.

They found that the trie was bulky as it occupied around 780 Kbytes. What they did was represent the trie as a directed acyclic word graph (DAWG) in which all repeating sub-tries have been merged. This reduced their number of nodes to 19,834 and the memory usage to 175 Kbytes. Below is an example.

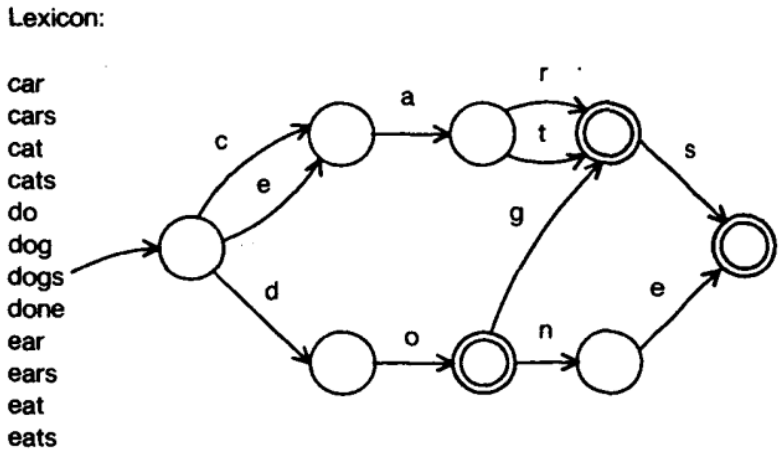


Figure 4. A DAWG. Nodes with edges leading to the same state in regards to valid word formations as different paths leading to that state have been merged.

The experiment was performed on a VAX 11-780 computer which, with a maximum memory of 64MBs [14], is much less compared to modern machines. With most machines now having gigabytes of memory, using the dictionary as a trie might not pose any challenge at all.

Another challenge was solving the problem itself – how is one to find all legal moves given the current tiles on the board? Their algorithm consists of a couple of techniques.

3.5.1 Techniques

One technique was limiting the problem to one dimension. Playing words vertically is like playing words horizontally but on a transposed board. This allows the algorithm to be designed for across plays an easily adapted for vertical plays.

Another technique is cross-checks. When making a horizontal play, each placed tile must form a valid word when connected with the already placed tiles that are vertically adjacent to it. But a key note is that in a horizontal play, only one tile can be added to a column. With this knowledge it is not too difficult of a task to compute all valid cross-checks which is important for optimizing the entire move generation from the AI agent and reducing the number of checks for valid words.

A third technique is the idea of anchors. Anchors are tiles which are adjacent to already placed tiles on the board. The rules of Scrabble dictate that each new word must be connected to an existing one, which means every word on a board (apart from the first) has used an anchor that has connected it to an existing word. Using this information, the algorithm starts looking for valid moves by locating all anchor points on the board, specifically those that are located on the left-most adjacent space of a placed tile, giving a starting point for move generation. A figure can be found below for both anchors and cross-checks.

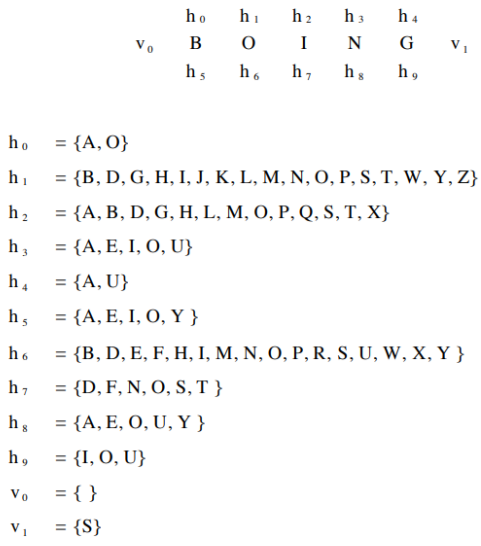


Figure 5. The letters h and v represent anchors for the current tiles and possible letter placements that would pass the cross-checks [2]

Once the anchors have been identified, the algorithm generates legal moves computing valid left parts (prefixes) anchored at the anchor point and computing all right parts (suffixes) for each valid prefix. Prefixes are built either from tiles from the rack or tiles already on the board but not both at the same time. After all moves for a given prefix are generated, the prefix is changed and will eventually grow in size. By looking up at the DAWG, valid words are computed by traversing the DAWG using tiles from the rack or already on the board and each successful traversal that stops at a terminal node is considered a legal word. Pseudocode from the paper is shown below:

```
LeftPart(PartialWord, node N in dawg, limit) =
        ExtendRight(PartialWord, N, AnchorSquare)
        if limit > 0 then
                for each edge E out of N
                        if the letter l labeling edge E is in our rack then
                                remove a tile labeled l from the rack
                                let N' be the node reached by following edge E
                                LeftPart (PatialWord + l, N', limit - 1)
                                put the tile l back into the rack

ExtendRight(PartialWord, node N in dawg, square) =
        if square is vacant then
                if N is a terminal node then
                        LegalMove(PartialWord)
                for each edge E out of N
                        if the letter l labeling edge E is in our rack and l is in the cross-check set of
                        square then
                                remove a tile l from the rack
                                let N' be the node reached by following edge E
                                let next-square be the square to the right of square
                                ExtendRight(PartialWord + l, N', next-square)
                                put the tile l back into the rack
        else
                let l be the letter occupying square
                if N has an edge labeled by l that leads to some node N' then
                        let next-square be the square to the right of square
                        ExtendRight(PartialWord + l, N', next-square)
```

Figure 6. Pseudocode taken from Appel and Jacobson's anchor extension algorithm. For each valid prefix, the word is extended to the right as much as possible given available rack tiles and all combinations are checked before trying again with a new prefix.

3.5.2 Analysis

Appel and Jacobson's algorithm is the most prevalent in Scrabble implementations from various other scholars and that also includes Maven [4], thanks to the algorithm's efficient techniques. As computers today are much more powerful than 30 years ago, different data structures for storing the lexicon may be more feasible and easier to manage but the transposing, cross-set checking and anchoring techniques are still very efficient for solving the Scrabble puzzle and will most likely be used for the final implementation.

It is important to note that this algorithm is not strategic in any way, it's only (but great) benefit is the speed with which it makes decision for the best words. It only performs a one-ply search and can only pick the highest scoring word in the current situation and it does not consider, for example, that using the selected word might create an opening for the opponent to use a premium tile and does no analysis of the probability of that opening being exploited as no simulations are done. Maven is

such a system that incorporates this algorithm and also uses other strategic elements that make it act more similarly to a human player.

A possible improvement Appel and Jacobson thought about was the implementation of a two-way DAWG. The way the algorithm works now is that it builds every possible prefix to the left of an anchor tile before looking if it is possible to build the suffix to go along with that prefix and hooked tile. The issue is that, due to the tiles already placed on a board and tiles on the rack, there will be times when it may be impossible for a given prefix to form a valid word. They suggest using a two-way DAWG. In this iteration, each node would represent the middle sections of words and would have out-edges for each letter that could be connected either to the beginning or the back of that node. By having that knowledge, a search for words could go both ways at the same time instead of extending a word rightwards with a given prefix (this is how it works now). A downside of a two-way DAWG would be the memory usage as it would need a node for each substring of each word. Steven Gordon offers a solution in the next section.

3.6 Steven Gordon's Algorithm

Gordon offers an implementation of a two-way DAWG called GADDAG as it works on a somewhat reverse principle of a directed acyclic graph. Gordon's algorithm presents every word in a given lexicon in a few representations and those are added to a graph (not yet minimized). The number of representations of a word is equal to the number of letters in that word. Each representation of the word is also split by a special character called a delimiter which splits the word into a prefix and suffix. But there is a peculiarity in the order of the letters, particularly in the prefixes. An example is given below, followed by an explanation.

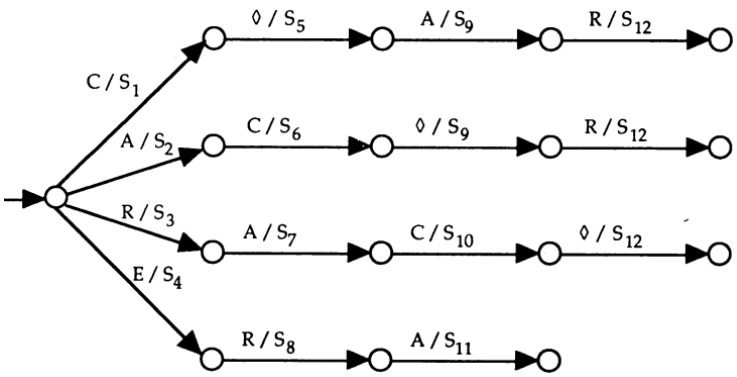


Figure 7. An example of the representations of the word CARE in Gordon's GADDAG. The rhomboid acts as a delimiter which separates the prefix and the suffix. The letters in the prefix are reversed for the reason given below.

Techniques

Gordon's algorithm employs the same technique from Appel and Jacobson's algorithm regarding anchor selection. Similarly to their algorithm, a word is built up by building up a prefix first on the left of the anchor and a suffix on the right of the anchor. In their algorithm, all generated prefixes (given current rack) of increasing size are placed in front of the hooked letter and checked if they form a word (size increases until the border of the board is hit or until already placed tiles are seen). In Gordon's algorithm, just like in Appel and Jacobson's algorithm, the suffix is built up in a direction opposite of the anchor (right) but the prefixes are built in a direction opposite of the anchor as well

(left). The letters in the prefix, before the delimiter, in the GADDAG are reversed because they are appended in a direction starting from the anchor and away from it to the left. An example is given below.

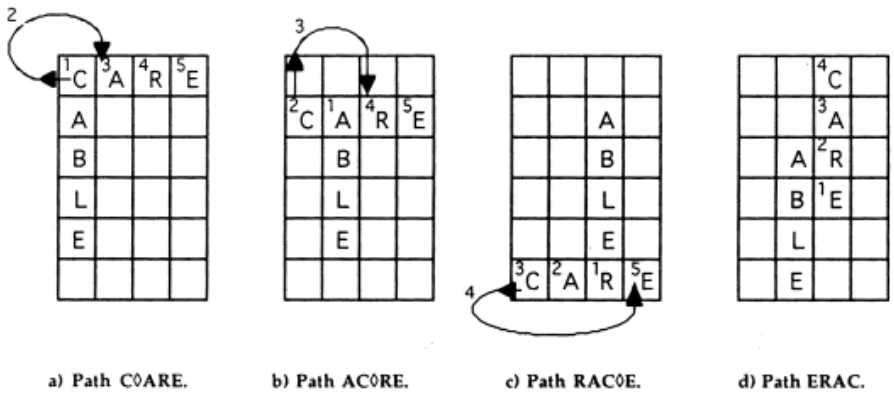


Figure 8. In Figure a), the "word builder" is set to move add letters to the left of a given point. C is added first in a given spot. Once a delimiter is hit, the builder switches word building to the right of the given point instead. In Figure b), A is added to a location and C is added to the left of A, forming CA. Once the delimiter is hit, R is added to the right of the point instead and E is added to the right of R, forming CARE. The same idea applies to the other figures.

Why is GADDAG faster at searching than DAWG? An example can be given with the horizontal word HARPY on the board [25]. If the player wanted to use P as the hook square and had a space of two letters above it, with DAWG the algorithm would have to go through all the words in the lexicon to find the words which have P as the third letter and many searches would be unrewarding. In the GADDAG, the machine will see all words that have a P somewhere in it and will be able to go through all the branches of P and form words, thanks to the use of the delimiter which allows every word containing P to be represented with a starting letter P and a combination of prefixes and suffixes. For example, a GADDAG might contain the word EXPLAIN as PXE<>LAIN and deduce that EXPLAIN is a valid word. In a DAWG, the machine would learn it is possible by visiting the branch starting with E and "stumbling" on a word that had its third letter as a P but with GADDAG this "stumbling" is greatly reduced. Detailed pseudocode for move generation is given below:

```
Gen(pos, word, rack, arc):        {pos = offset from anchor square}
        IF a letter, L, is already on this square THEN
                GoOn(pos, L, word, rack, NextArc(arc, L), arc)
        ELSE IF letters remain on the rack THEN
                FOR each letter on the rack, L, allowed on this square
                        GoOn(pos, L, word, rack - L, NextArc(arc, L), arc)
                IF the rack contains a BLANK THEN
                        FOR each letter the BLANK could be, L, allowed on this square
                                GoOn(pos, L, word, rack - BLANK, NextArc(arc, L), arc)

GoOn(pos, L, word, rack, NewArc, OldArc):
        IF pos <= 0 THEN          {moving left:}
                word <- L || word
                IF L on OldArc & no letter directly left THEN RecordPlay
                IF NewArc != 0 THEN
                        IF room to the left THEN Gen(pos-1, word, rack, NewArc)
                        NewArc <- NextArc (NewArc, <>{delimiter})       {shift direction:}
                        IF NewArc !=0, no letter directly left & room to the right THEN
                                Gen(1, word, rack, NewArc)
        ELSE IF pos > 0 THEN      {moving right:}
                word <- word || LeftPart
                IF L on OldArc & no letter directly right THEN RecordPlay
                IF NewArc != 0 & room to the right THEN
                        Gen(pos+1, word, rack, NewArc)
```

Figure 9. The GADDAG move generation algorithm. The algorithm will start off in an initial state from an anchor square and visit every possible prefix, recording moves as valid if the paths of those prefixes are ended in the GADDAG. When going along the branches, hitting a delimiter will force the algorithm to change direction and build away to the right from the square and again, record a valid play if there are no more letters along its path.

The GADDAG in the example above can be minimized. It is important to notice that a GADDAG represents several ways to make a word by splitting it into different prefixes and suffixes. Some representations will have a suffix of longer length than the suffix of other representations but, considering the final product is the same, this means that the longer suffixes are made up from the shorter suffixes of a different representation along with some more letters. For example in CAREEN, C<>AREEN and AC<>REEN have suffixes AREEN and REEN. When looking at those suffixes we can say that REEN is a part of both suffixes. This means that C<>A and AC<> lead to the same state from which REEN is a valid suffix, thus instead of having two separate states for C<>A and AC<>, the GADDAG can be minimized by merging the states that C<>A and AC<> lead to as they are the same. This can be built further by looking at RAC<>EEN which has the suffix EEN. With the same logic, RAC<> can be combined with the state that C<>AR and AC<>R lead to as all three states would be valid for using the suffix EEN. A figure displaying the construction is shown below.

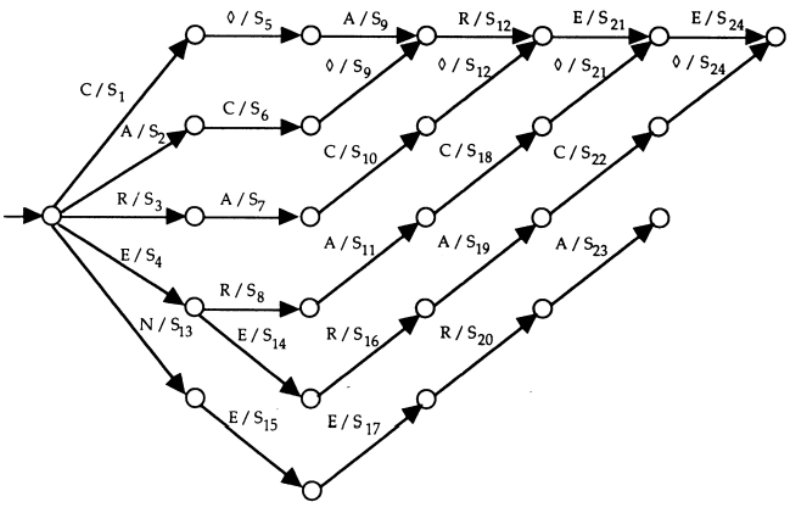


Figure 10. Sub-graph of a minimized GADDAG for the word CAREEN. States that have the same finishing suffix have been merged.

Gordon has also given differences between the size of the lexicons of DAWG and GADDAG as well as their performance results in the figures below.

	DAWG		GADDAG		Ratio (G/D)
	Unminimized	Minimized	Semi-minimized	Minimized	minimized)
States	55,503	17,856	250,924	89,031	4·99
Arcs	91,901	49,341	413,887	244,117	4·95
Letter sets	908	908	2575	2575	2·84
Expanded					
Bytes	5,775,944	1,860,656	27,110,092	9,625,648	5·17
Bits/char	91·6	29·5	429·8	152·6	
Compressed					
Bytes	593,248	272,420	2,669,544	1,342,892	4·93
Bits/char	9·4	4·3	42·3	21·3	

Figure 11. Size difference/ total number of states between DAWG and GADDAG. The states generated by GADDAG are around 5 times more.

	DAWG		GADDAG		Ratio
	overall	per move	overall	per move	(D/G)
CPU time					
Expanded	9:32:44	1·344s	3:38:59	0·518s	2·60
Compressed	8:11:47	1·154s	3:26:51	0·489s	2·36
Page faults					
Expanded	6063		32,305		0·19
Compressed	1011		3120		0·32
Arcs Traversed	668,214,539	26,134	265,070,715	10,451	2·50
Per sec (compressed)	22,646		21,372		
Anchors used	3,222,746	126·04	1,946,163	76·73	1·64
Number of moves	25,569		25,363		
Average score	389·58		388·75		

Figure 12. Performance time difference between DAWG and GADDAG (on VAX machine with 64M of memory). The speed of move generation using GADDAG is more than twice as fast.

3.6.1 Analysis

While the difference in memory usage may be insignificant in modern machines and speed differences may be radically reduced, the greatest benefit of Gordon's algorithm will be seen in move simulations. Simulations, as will be seen in following sections, involve making a move, generating a possible move back from an opponent then coming up with the best comeback to the opponent's move. This would be a two-ply search and a four-ply search might present the AI agent with moves that are better for winning the game but multiple-ply searches can be very costly memory and speed wise. Players in tournament games usually have a time limit of 25 minutes thus generating moves at twice the speed of DAWG can be invaluable and there has been no mention in any papers about having to use a particular set of hardware for tournaments, thus the problem of higher memory usage is "invisible" to players, the only requirement being that the AI agent will still work. Gordon's algorithm offers a strong alternative to Appel and Jacobson's and whether it is better or worse depends on the game preferences and available hardware resources (if speed is a requirement, the preferred option is GADDAG as it is faster and if machine resources are low, the answer is DAWG as it is less memory intensive).

These two algorithms form the basis for many Scrabble implementations that have been developed. The following sections will examine AI champion Maven and research and implementations from other scholars.

3.7 Maven

A paper on the fully-working Maven was produced by its creator Brian Sheppard in 2001 [4]. His first attempts at producing a Scrabble AI were in 1983 and, over a development period of a few years, his system received its current name and was further improved. Maven features an extensive set of evaluation functions and parameters and can perform simulations to average out scores from different scenarios. In general, Maven deduces a score for the game state by looking at its rack of tiles, the tiles that are already on the board and current score. Based on that information, Maven reaches the best solution which is largely dependent on the following heuristic functions.

3.7.1 Heuristics

The first heuristic is called the Vowel/Consonant Balance. An important part of playing effectively is to avoid getting stuck with an unplayable or hard-to-play rack. Such a rack is considered one if there are many more vowels than consonants or vice-versa. Having too many of either makes it hard to form a word and make a move where as having balance makes this task easier and will result in more consistent high-scoring plays, even though the current decision for a most might not bring the most points. The function has a threshold value that the current rack needs to pass. The rack will have a positive value if there is a good balance of letters and a negative one if one set outnumbers the other one, in which case Maven will prioritize playing out the dominant group to avoid getting stuck in future plays [11].

The second heuristic function is the U-With-Q-Unseen. Q can be very troublesome to play and being stuck with it could lead to unfavourable racks. If there is a U in the rack without a Q, Maven may prioritize keeping the U and playing it with a Q as it is a combination that is not too troublesome to play.

The third function is the First-Turn Openness. It is generally accepted that a player should not easily allow the other player to play a word using a premium tile. By using a long word as first, the chance for the opponent to make a play using a premium tile is greater. For that reason, whenever Maven is first, it will always open with a small word such as a two-letter word. This reduces the possibility of the opponent gaining an advantage using a premium tile.

Maven employs these functions in a slightly different fashion and employs different search strategies depending on the phase of the current game. The game is classified as one of three phases.

3.7.2 Game Phases

The first phase of the game is called the normal game phase. It lasts between the start of the game and up until there are 16 unseen tiles left (9 in the pouch and 7 in the opponent's rack). It is a basic one-ply search and uses an evaluation function that is quick and accurate enough for high-level play.

The second phase is the pre-endgame phase which lasts between having 16 unseen tiles and until the pouch is empty. Maven uses an oracle function to perform a two-ply search for a best move. Sheppard claims that a move usually uses up around 4.5 tiles thus performing searches deeper than two-plies is non-optimal as the rack only holds up to 7 tiles and the rack turnover is quick. Maven does a simulation playing out a few turns with random racks and picking a move with a high average point differential (an example can be found in the Implementations section). The idea behind such simulations comes from Sheppard's discussions with national champion Ron Tiekert in December 1987. For his own experiments, Ron had the choice of playing either AWA or AWARD as a starting move. AWARD was obviously a higher-scoring lead but after 50 games starting with each word, Ron deduced that AWA brought higher scores in the long run. This helped Sheppard develop the First-Turn Openness heuristic along with the realization of the importance of simulations.

The third phase is the endgame. During this phase the game becomes a game of perfect information i.e. it is possible to deduce the opponent's tiles in the rack as the pouch is empty and all the other tiles are either on the board or on Maven's rack. Maven uses a B*search algorithm to pick a best move. Alpha-beta has been tested by Sheppard but it was deemed too inefficient as endgame scenarios can be very turn-heavy (up to 14 turns) and simulations of that depth would be impractical.

3.7.3 Analysis

Maven is solid product of AI engineering. It is a system that has been iteratively developed over a long period with techniques that have been well studied and researched. It is also evident that Sheppard and his team have a solid grasp of professional gameplay and have been in touch with professional players with the goal of improving their system. The best game Maven has played, according to Sheppard, was against Adam Logan in their 12th game in the AAAI-98 exhibition match where Maven famously turned a 98 point deficit into a win in the endgame by playing MOUTHPART at 1A worth a 100 points.

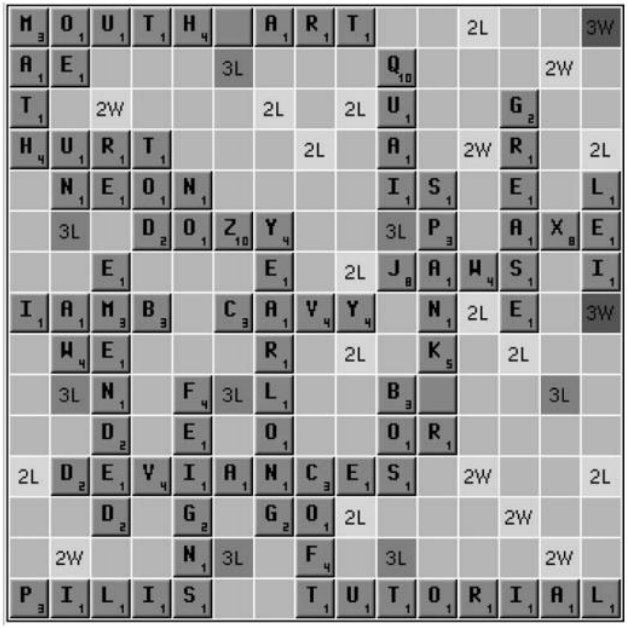


Figure 13. Maven versus Adam Logan (exhibition match, AAAI-98). Maven's last move is MOUTHPART at 1A.

Overall, Maven offers an ambitious solution for a few (tournament-level) use cases but there is useful information gained from it.

- Appel and Jacobson's algorithm is still applicable and proven to be good considering it is used for tournament play.
- Steven Gordon's GADDAG is also a good alternative as Sheppard has mentioned it is probably better than a DAWG but not as versatile.
- Scrabble is a game with imperfect information up until the endgame, predicting scenarios using searches such as A* and alpha-beta pruning is infeasible (but still doable) as the different scenarios of the board state and rack tiles of the players are too many and B* is a good solution.
- The used algorithm can be made smarter with some implementations of Maven's main heuristics such as Vowel/Consonant balance or First-Turn Openness.
- Simulations greater than two-plies will rarely return better results and the computational time needed for them is impractical.

There is also a lot of secrecy regarding Maven as there is no open-source code for it and the paper does not talk about its implementation in great detail. Fortunately, there are a few Maven-like implementations developed by other scholars and their research, while also secretive, reveals some clues on how to develop a smart Maven-like artificial player.

3.8 Additional research and implementations

There are various implementations and papers from other scholars and they all introduce new ideas either in their research or implementation.

D. Janakshi Dulanga's implementation

One implementation is from D. Janakshi Dulanga [1] who designed a Scrabble game for the Sinhala language. Key takeaways from this implementation are the addition of a language that's not officially recognized, it uses the GADDAG structure for word searches, alpha-beta pruning for look-ahead in the endgame, the board is stored in an array and it was developed using Java with an MVC pattern. Her final implementation can be seen below.

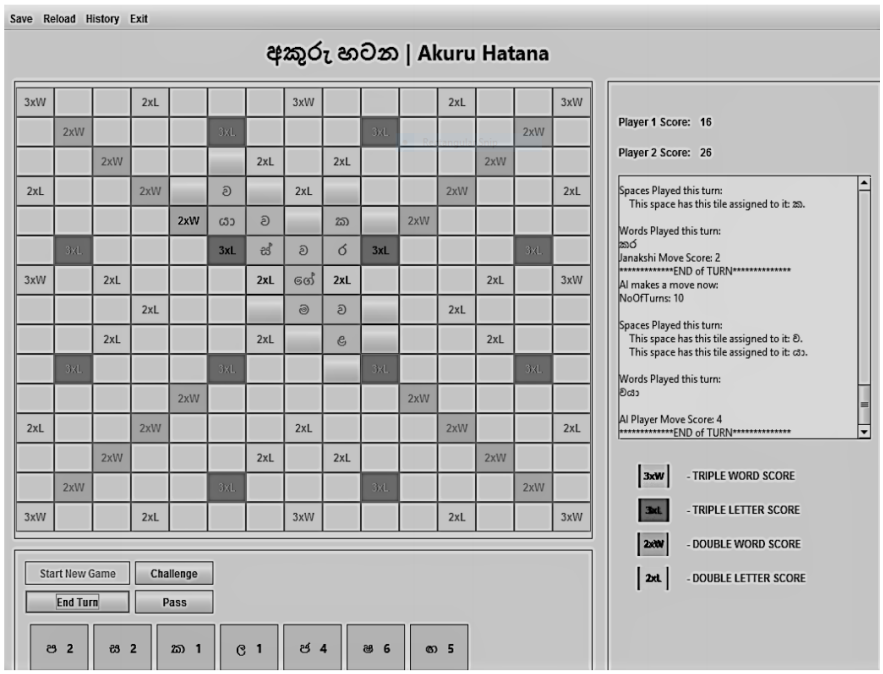


Figure 14. Dulanga's Java implementation of Scrabble. It features a Java GUI, the Sinhala letters and a log of performed moves to the right

3.8.1 Steven Gordon's findings on potential heuristics

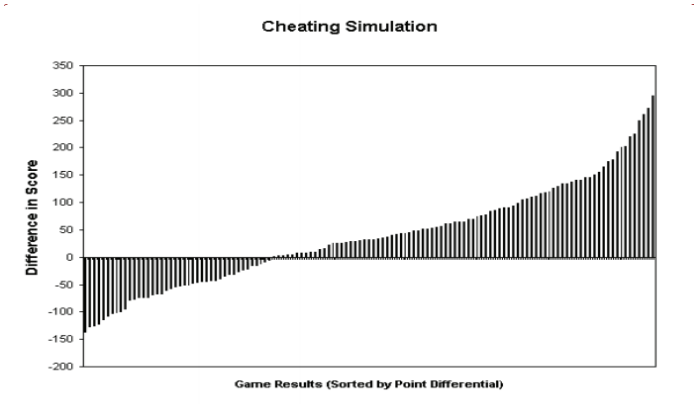
A different paper from Steven Gordon [8] talks about the differences between probabilistic search and weighted heuristics in the game. The paper reveals valuable information that was not available in Sheppard's paper such as exact data for calculating a heuristic based on the remaining letters on a rack, a formula for calculating the vowel-consonant ratio such as "VCMix = MIN(3V + 1, 3C) L = MIN(3V +1 - L, 2L - 3V) (where V = number vowels, C = number of consonants and L = number of letters)" and even talks about a heuristic which talks about duplicated tiles on a rack and their reduced values. He also mentions a fairly simple formula for picking a best move after simulations, that being "score of current move - the opponent's score of next move + the comeback score of current player's next move)". The tables below show two different heuristics and their grading in regards to individual tiles and duplicated tiles in the rack.

Letter	Heuristic1	Heuristic2		Letter	Heuristic1	Heuristic2	
A	+0.5	+1.0	−3.0	B	−3.5	−3.5	−3.0
C	−0.5	−0.5	−3.5	D	−1.0	0.0	−2.5
E	+4.0	+4.0	−2.5	F	−3.0	−2.0	−2.0
G	−3.5	−2.0	−2.5	H	+0.5	+0.5	−3.5
I	−1.5	−0.5	−4.0	J	−2.5	−3.0	
K	−1.5	−2.5		L	−1.5	−1.0	−2.0
M	−0.5	−1.0	−2.0	N	0.0	+0.5	−2.5
O	−2.5	−1.5	−3.5	P	−1.5	−1.5	−2.5
Q	−11.5	−11.5		R	+1.0	+1.5	−3.5
S	+7.5	+7.5	−4.0	T	−1.0	0.0	−2.5
U	−4.5	−3.0	−3.0	V	−6.5	−5.5	−3.5
W	−4.0	−4.0	−4.5	X	+3.5	+3.5	
Y	−2.5	−2.0	−4.5	Z	+3.0	+2.0	
BLANK	+24.5	+24.5	−15.0				

Figure 15. Heuristic1 shows evaluation of each letter in the rack. Heuristic2 shows a similar evaluation as well as a penalty for each repeated tile.

3.8.2 Mark Richards and Eyal Amir's findings on implementing smart simulations

Mark Richards and Eyal Amir [7] talk more about simulations and discuss how, as the number of tiles in the pouch decreases, the tiles in the opponent's rack become more predictable. This information can be used to get a rough guess for each simulation what tiles the opponent might have from a rack and that guess would be more accurate as the pouch becomes emptier, leading to more and more predictable potential moves from the opponent and thus a better evaluation of what the AI's current move should be as it will know what to expect. In the endgame, the AI will have perfect knowledge of expected moves as it will know all the tiles in the opponent's rack. Below are tested point differences and performance against a Maven-like implementation called Quackle (covered in next main section).



	Full Knowledge	Quackle
Wins	87	61
Mean Score	438	401
Biggest Win	295	136

Figure 16. The upper graph shows score differences from simulations where the agent knows the opponent's rack. The lower graph shows the scores of this agent against Quackle (an open-source implementation and an analysis tool comparable to Maven)

3.8.3 Frej Connolly and Diana Gren's findings on implementing different move generation priorities and strategies

Frej Connolly and Diana Gren [9] have also made an implementation with interesting features. Their implementation uses a trie (tree-like data structure) for the Swedish lexicon representation which has not been turned into a DAWG or GADDAG, showing an implementation is possible without those modified data structures. They have developed three different AI agents with different priorities for winning. The High Score Word player is equivalent to the one from Appel and Jacobson's where the word with the highest score is played without additional heuristics. The Bonus Square player will prioritise moves that use up any of the premium tiles as they may not bring the highest score but will prevent the opponent from using them somewhat. The Balance On Rack player will prioritise moves that leave a healthy balance of vowels and consonants on the rack which enables the agent to more easily play high-scoring moves in the future. The Bonus Square player seemed to win the most games. Below are graphs comparing the three agents against one another.

Vowel ratio	HSW wins	BOR wins	Vowel ratio	BS wins	BOR wins
0/8	9922	73	0/8	9925	72
1/8	9999	1	1/8	9987	12
2/8	9484	507	2/8	9670	325
3/8	9777	212	3/8	9829	165
4/8	9735	262	4/8	9818	172
5/8	9627	368	5/8	9731	261
6/8	9146	838	6/8	9370	620
7/8	9863	134	7/8	9859	141
8/8	8762	1206	8/8	9074	915

	Bonus Squares	High score words
Wins	6104	3857
Draws	39	39
Wins when started playing	3148	2040
Mean score	308	278
Median score	301	273
Highest score	724	606
Lowest score	85	61

Figure 17. Results between the three agents pitted against each other. The Balance On Rack agent did not fare well against either opponent and the Bonus Square player seems to win between 50 and 60 percent of its games against the High Score Word agent.

The following section is the comparison of some commercial and open-source software.

3.9 Features from commercial and open-source software

There are few a commercial games and open-source projects of Scrabble which provide interesting features.

3.9.1 Quackle

The most feature rich Scrabble playing experience seems to be provided by Quackle. Open-source and developed in C++ it offers a lot of the features which are goals of this implementation – the ability to play with multiple people and computer bots, computer bots can have various difficulties, different languages (and more can be added by changing the game files), customization of the board

in regards to size and premium tiles, customization of simulation, simulations on demand and more – it is a very good analysis tool. As it offers so many of the goal features in full working order, it would be of great benefit to match the results of this implementation to that of Quackle.

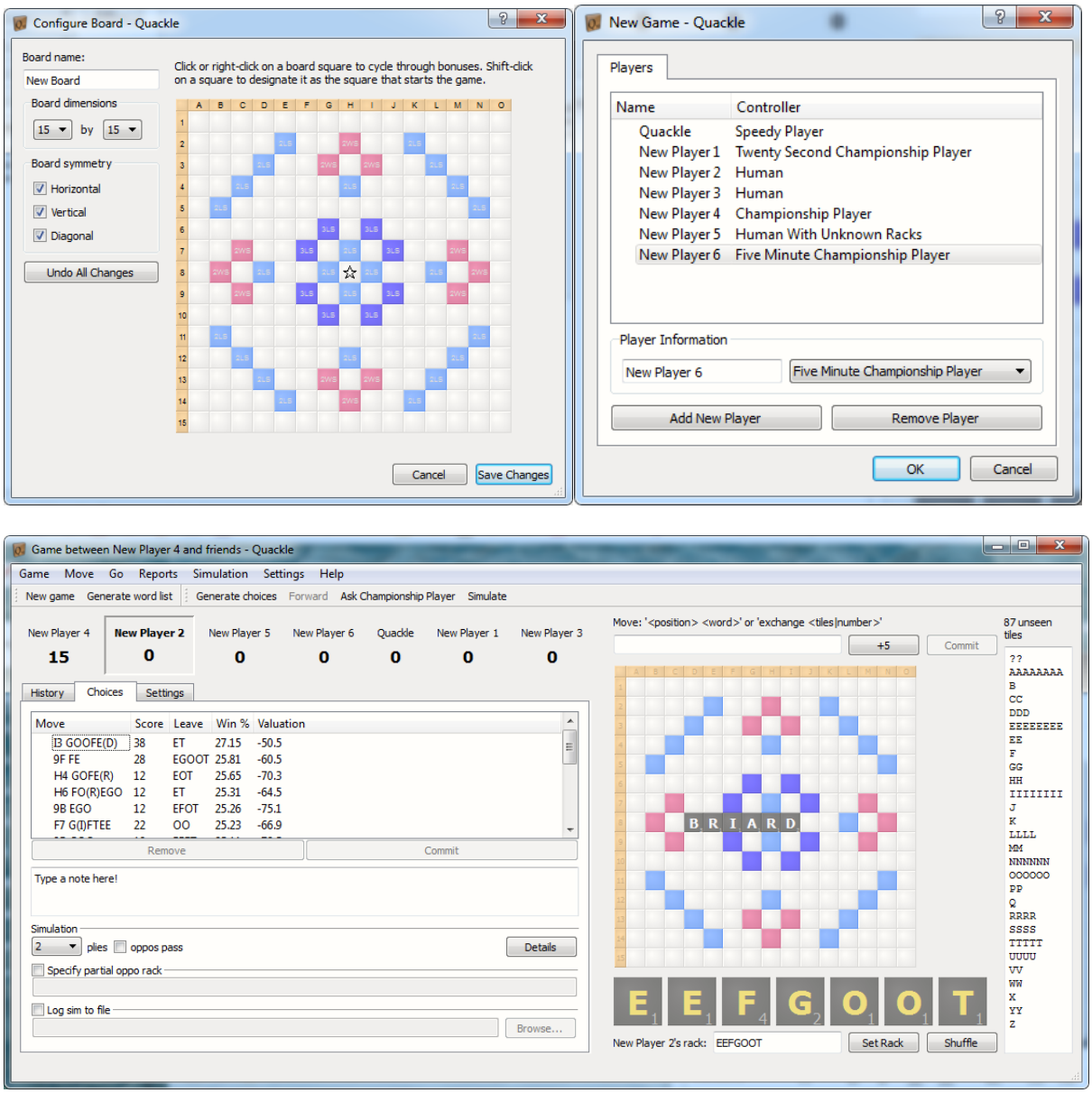


Figure 18. Demonstration of the Quackle GUI and the many options available to the user regarding gameplay and analysis.

3.9.2 Classic Words Solo

Another popular Scrabble game on mobile phones is called Classic Words Solo. It offers a very compact Scrabble game with some core features such as difficulty and a few languages. Key takeaways are how responsive it is in general and how well it works with drag-and-drop controls. The board can also be zoomed in or out. Images from the game are given below.

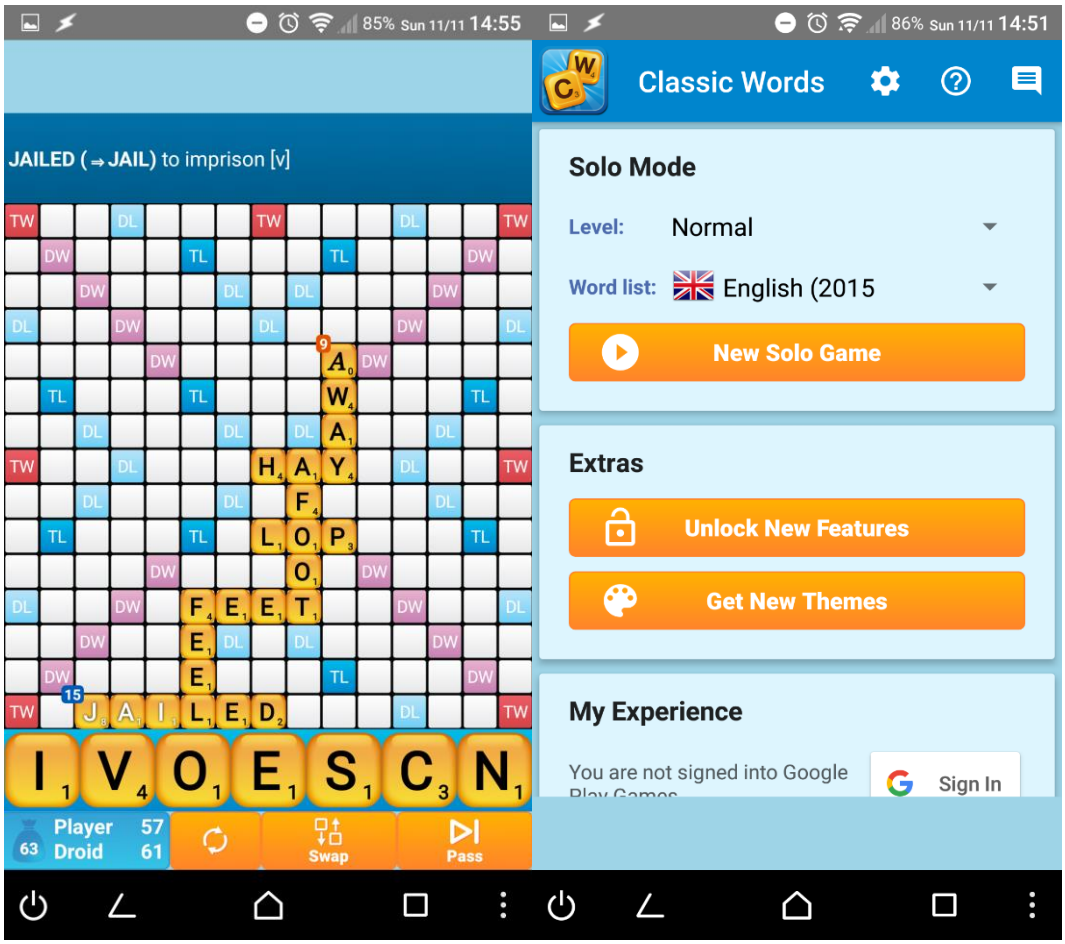


Figure 19. Images from Classic Words Solo. Interesting features include live messages of meanings of the last played words as well as 6 different difficulties and dictionaries in 6 different languages.

Implementing the best features and characteristics of the two games above would be beneficial as they are both of a high tier and provide an excellent user experience.

The following section is the conclusion.

3.10 Conclusion

There are some more documented implementations of the game [2, 11, 12, 13] which more or less present similar ideas but from this selection we have a solid grasp of how the development for these implementations goes. There is enough evidence of the effectiveness of the basic algorithms and they can be used as a starting point to developing a lexicon representation and a basic form of an AI player. Once word checking mechanisms are satisfactory, additional decision making can be added via heuristics and simulations. Java, C(Maven) and C++(Quackle) have been used in implementations thus the personal choice of C# should not prove to problematic for a game like this. As MVC has also been used as a pattern, an imitation using the given pattern should yield satisfactory results. The game's functionality will be developed first before any visual elements.

4. Design & Methodology

The first question that had to be answered when picking this project was “How will this be developed?” The “How” is referring to the main tool (or language) for development which would dictate how all other tools (such as packages and frameworks) would be used along with it.

4.1 Language and Framework Selection

From the review we have noticed that there are many applicable languages and technologies that can be used. This means that there no real limitations to developing an application of this kind. For example, the implementation from Janakshi Dulanga is written in Java which is a high-level programming language.

This report does not cover technicalities in the differences between languages. Instead, a similar language to the ones used in existing implementations (such as Java) was selected based on the technologies for application development available around its usage and personal motivations. C# and the features of the .NET Framework were selected for the solution.

4.2 C# and the .NET Framework

C# is a high-level compiled language and was developed in 2002 by a Microsoft team lead by Anders Hejlsberg [26]. It is an extension of the low-level language C, although they are vastly different languages. It is based on Microsoft’s .NET Framework. The .NET Framework provides tools and libraries to develop applications for Windows machines, phones or web applications [27].

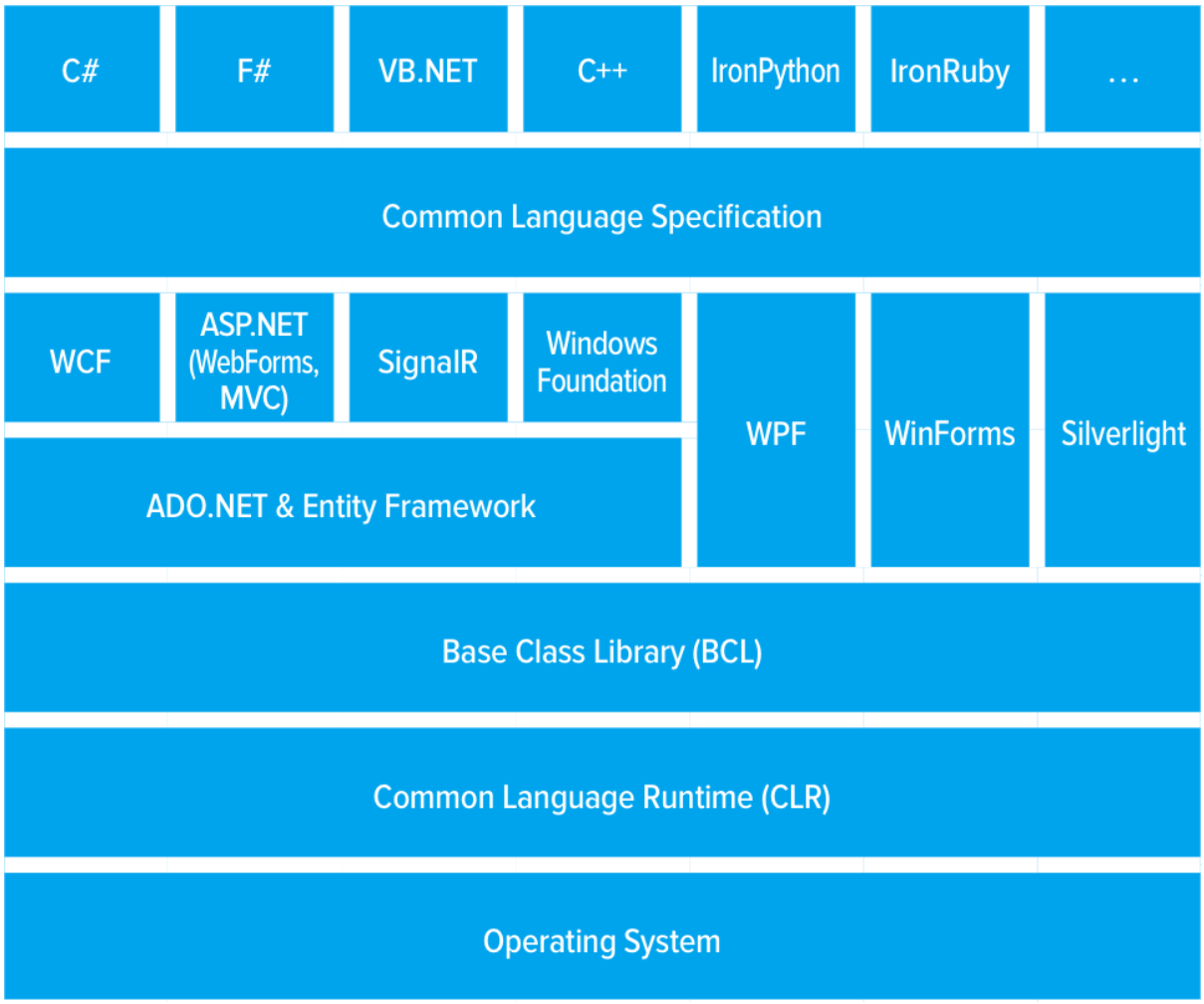


Figure 22. Some of the technologies provided by .NET Framework.

4.2.1 Why these technologies?

The combination of C# and the .NET Framework are very commonly used for the development of Form-based web applications. There is no written paper to prove this statement but this was learned from personal involvement and experience within the web development industry. Such an application is one where there is an object (that being the Form which has a collection of sub-objects) and it changes states depending on user input and some logic (commonly known as Business Logic i.e. a set of steps set up by the provider of the forms) performed with the input used as a parameter.

The above workflow for an application can be adapted for a Scrabble game. A game can be looked at as a form that has a series of steps:

- Player (user) is presented with a board, pouch and rack (all part of a form).
- Player plays a word from his rack (provides input) and announces the play (submits input).
- The players or a third party (server i.e. form provider) validate the input (checks if the word is valid and played in correct conditions).
- If valid, the play is saved and the game passes to a different player (form gets updated and its state is changed).
- Repeat previous steps.

These are the main steps of a game. There are other steps in between (such as redrawing or skipping a turn or getting an error from the form provider regarding a bad play) but they all work the same way – input is provided by a user and, depending on that input, the form may or may not change its state in a particular way.

An advantage to a web application is that it does not need to be installed of any machines other than the server (form provider). The players (users) that would like to play would only need a web browser and to follow an address to access the form. This makes the application more accessible to a broader audience due to the minimal requirements before usage. In addition, none of the implementations from the literature review are web-based, instead they are installed on a machine running the game thus only accessible by the developer of the implementations.

Another advantage is that web applications can be accessed both by desktop users and mobile device users. While development for a singular platform is possible (such as for Windows/iOS/Android), that would make the application exclusive for that platform. A web application is usable by all platforms as long as that platform has a web browser to access the application (which most platforms do have). This allows a greater focus on game logic without getting bogged down in technicalities relating to a platform (but in future work a platform-exclusive implementation might be an option).

From the discussion above the following points can be deduced:

1. Developing the project as a web application is possible by following the workflow of a typical Form web-based application.
2. Developing the project as a web application will provide access to the game for a broader audience and the capability of having multiple players is one of the project aims.

Locally installed applications might have some advantages over web-based applications but there is no official research that says that might be the case in regards to Scrabble implementations. Developing the project as a locally installed application would not be breaking any new grounds in regards to research and, possibly the biggest factor, the development of the project as a web application aligns very heavily with personal interests. For these reasons and from the discussions above, the project was built as a web application using the C# language and the .NET framework. It was built as an ASP.NET Core application.

4.3 ASP.NET and ASP.NET Core

ASP.NET is an open-source framework used for the development of server-side web applications [28]. It was released with the initial release of the .NET Framework. The last official update of ASP.NET was in October 2017 as Microsoft have put more effort into the development of ASP.NET's successor ASP.NET Core. ASP.NET Core is a rewrite of ASP.NET and is designed to be cross-platform and to offer better performance than its predecessor.

There are other differences between ASP.NET and ASP.NET Core but the general trend is that Microsoft are pushing for development with ASP.NET Core. As it is a newer technology it is expected to have some features missing compared to ASP.NET or not be as stable. Considering that this Scrabble implementation will follow the standard workflow of a form-based web application and

that the goal of this report is to establish what is possible or not given current tools and information, ASP.NET Core was used as the main framework of this project.

ASP.NET Core applications tend to follow a specific pattern for development and that pattern is called MVC (Model-View-Controller).

4.4 MVC

MVC is an architectural pattern that is commonly used for the development of applications that have a user interface [29]. The pattern's purpose is to separate the application into three major components for the purposes of maintaining separation of concern between the parts and to allow efficient code reuse and parallel development of each part. The three parts are called the model, the view and the controller.

4.4.1 Model

A model represents an object or a type of data structure of the application. A model will also contain the steps for managing the data, logic and rules of an application. In this implementation, the models are all the objects which are contained within the game code.

4.4.2 View

A view is a graphical representation of information. The information displayed can be customized as a chart or graph or whatever a business needs and the information may originate from a model. In this implementation, a single view is used to as a graphical interface of the game as it displays a board, players, scores, racks, tiles and more. The view is created as a Razor Page which combines HTML mark-up with C# code to provide and HTML page that contains information about a single instance of a game.

4.4.3 Controller

A controller accepts user input and uses it to either modify a model or to generate a view. In this implementation there is a single controller which manages all interactions between the user and the game.

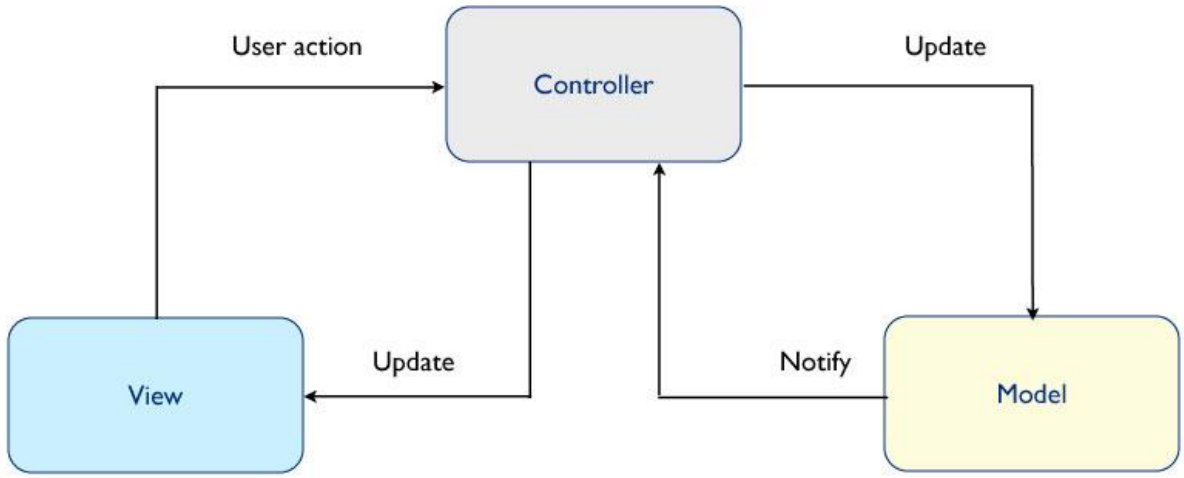


Figure 22. A demonstration of the interaction between the model, view and controller in an application.

4.5 Full Stack Development

Most web applications consist of three major components. The components are a client, a server and a database [30]. Using the example of a form-based application, the client is the user that provides input to the server who acts as a form provider. Once the form provider is happy with the user's data, the data gets stored in a "warehouse" of information called a database. Whenever a user asks the form provider to provide a form, the form provider looks for the requested form in the warehouse and brings it back to the user. Once the user has done his/her work, the updated form is sent back to the form provider and the provider stores the updated form in the warehouse for future use. A user can also create new forms or delete existing ones with the correct permissions.

All three components were implemented using a mix of technologies. All of the code written for the usage of these technologies was written in the Visual Studio 2017 IDE.

4.5.1 Client

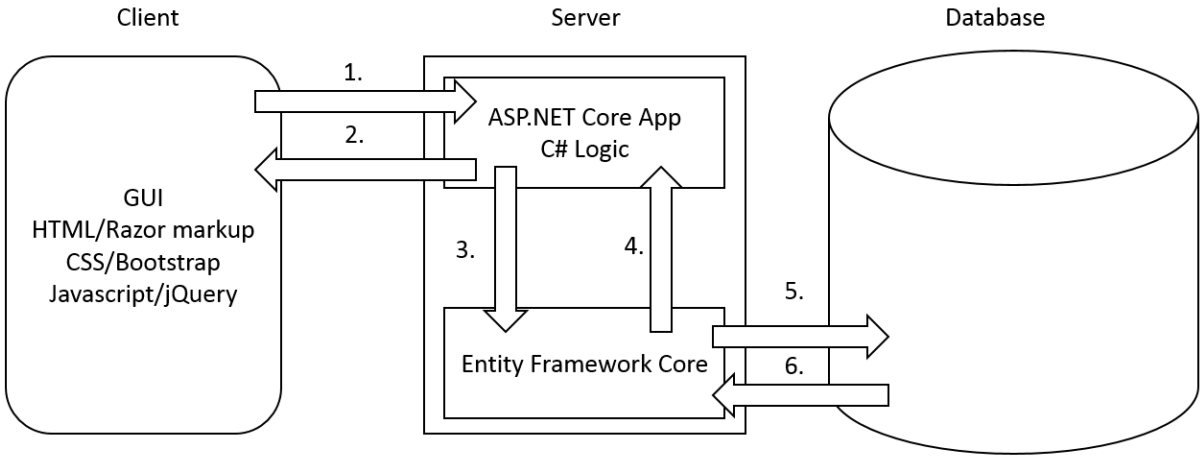
A client is a platform that uses a mix of technologies that a user would need to interact with an application. The technologies create the front-end of an application and are called client-side technologies. In this implementation, the technologies used are HTML (generated from Razor Pages), CSS (custom and Bootstrap) and JavaScript (combined with jQuery). HTML and CSS are responsible for the data and styling of a web page and JavaScript is responsible for any logic that happens on the page. Bootstrap offers extended styling options for a page and jQuery allows easier logical manipulation of a page's elements.

4.5.2 Server

A server is a platform that uses a mix of technologies that are responsible for performing logic on a form with information provided by the client. The logic is invisible to the client. The server is also responsible for handling client requests and only the server is capable of making changes to an application's database. The technologies create the back-end of an application and are called server-side technologies. In this implementation, a controller collects input provided by a client and calls various objects (instances of models or others) or helper methods within the application code (written in C#) to perform logic.

4.5.3 Database

A database is a platform that uses a mix of technologies that are responsible for storing user-specific and application-specific data for future use. Usually a database is stored on a separate server and contains tables and rows which represent objects or models of the application. ASP.NET Core comes with IIS Express which, upon running, generates a Microsoft SQL Server which acts as storage for the database and allows the server to interact with it. The technology that allows easier interaction between a server and database is called Entity Framework Core. EF Core is an object-relational mapper and is used to generate objects from database or to push objects to the database [31]. The generated item can also be used to populate a page for the user to look at. These operations happen through the use of a Context object that implements EF Core functionality.



1. Client sends user input to server running application
2. Server sends web page representing current or updated game to client
3. Application provides object to EF Core to update database
4. EF Core provides object from database to application
5. EF Core updates database with object provided from application
6. Database sends an entry as an object to EF Core as requested

Figure 22. Example of the workflow in a web application utilizing a client, server and database.

4.6 Requirements Analysis

The literature review went through various algorithms of playing the game as well as working implementations using those algorithms. But first, a basic working implementation of the game must be developed. Most requirements for such an implementation have been covered in the below table.

Requirements have been prioritized using the MOSCOW method. Requirements of priority MUST are expected for a basic implementation. Components of priority SHOULD are to follow after MUST priorities and components of priority COULD are to follow after SHOULD priorities. Lower level priorities are to be developed only after higher level priorities have been extensively tested.

The below table is a list of FUNCTIONAL requirements.

ID	Requirement (FUNCTIONAL)	Priority
1.	Ability to place tiles from rack on to board	MUST
2.	Ability to take back tiles from board to rack	MUST
3.	Ability to finish current play	MUST
4.	Ability to pass a turn	MUST
5.	Make sure cross-checks are valid	MUST
6.	Correctly locate anchor tiles	MUST

7.	Validate word against dictionary correctly	MUST
8	Correctly calculate score including premium tiles calculation	MUST
9.	Draw as many tiles from the bag as were used form the rack	MUST
10.	Dialog containing move history	MUST
11.	Tracking of scores	MUST
12.	Ability for AI to make any valid move	MUST
13.	Ability for AI to finish move	MUST
14.	Ability for AI to pass a turn	MUST
15.	Ability to redraw	SHOULD
16.	Ability to change dictionary language	SHOULD
17.	Ability to add players	SHOULD
18.	Ability to shuffle rack	SHOULD
19.	Ability for AI to make play highest scoring move	SHOULD
20.	Ability to reset current game	SHOULD
21.	Ability for AI to use heuristics for moves	COULD
22.	Ability for AI to use simulations	COULD
23.	Ability for AI to change strategy during different game phases	COULD
24.	Ability to save game	COULD
25.	Ability to remove players	COULD
26.	Ability to ask for help/best move	COULD
27.	Ability to see meaning of played words	COULD

Components of priority MUST are essential for basic gameplay. While having an AI opponent is a MUST for this project, the game must, at the very least, be playable my one more players and an end of the game with correct scoring, while playing according to the basic rules, must always be reachable. The game must be playable by the restrictions given word validity, anchor placing, cross checking and word connections and the same restrictions must apply to the AI opponent in addition to generating valid moves.

Most components of priority SHOULD have been referenced in the literature review as features that would make a basic implementation of the game stand out from other available implementations, that being multilingual support, a strong AI opponent and multiplayer support. Some of the components are features that are good for the game as well (shuffling, redraw) as well as for the developer (reset game).

Components of priority COULD involve developing the AI further to take it closer to championship-level strength (heuristics and simulations) as well as further extensions to the gameplay experience of the planned implementation (hints, saving game, removing players).

The functional requirements consider all topics covered in the literature review related to the implementation and features of the game. Non-functional requirements will also be covered. These requirements will cover how the game will be implemented rather than what it needs to be capable of and are subject to change depending on how development goes. As the choices regarding technologies have been made in the previous sections, these requirements will include them.

Below is a list of NON-FUNCTIONAL requirements:

ID	Requirement (NON-FUNCTIONAL)	Priority
1.	Game to be developed using C# and Visual Studio	MUST
2.	Game to be made to work using console input/output before developing interface	MUST
3.	To feature responsive GUI once tested	MUST
4.	To be developed with the MVC pattern in mind	SHOULD
5.	To use free services and APIs	SHOULD
6.	Be unique enough to avoid legal issues	SHOULD
7.	Use services such as AWS for database storage	COULD

Components of priority MUST establish a clear idea of how the technologies the game will be developed with (C# and VS, mostly due to previous research and personal preferences), the plan of development and the final product (responsive GUI).

Components of priority SHOULD propose ideas to personalize/differentiate the implementation of the game (MVC, use of APIs) as well as the notion of avoiding legal issues (this could be a MUST but commercializing of the implementation is not planned and thus left as SHOULD).

Components of priority COULD are additional extensions to the ideas of priority SHOULD (such as using AWS for database storage).

With the above functional and non-functional requirements listed, we can look into the application design.

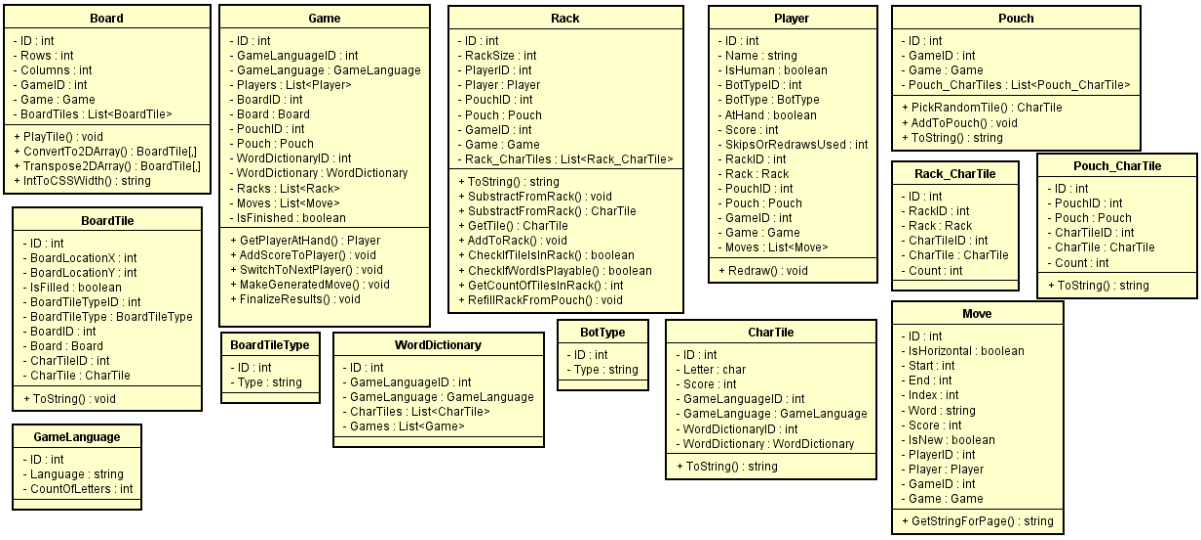
4.7 Application Design

All sections up until this one have served to introduce the reader to the technologies used in this project and how they work together to form a complete application. This section will show how these technologies and tools have been adapted to an application that can play games of Scrabble. It will show the very first ideas in regards to the structure and design of the application and their evolution in the final version of the project. This section will not go into detail about the logic of various components as that will be covered in the Implementation section. Its purpose is to show the environment within which all logic is happening.

4.7.1 Back-end, models and database design

Considering the structure and workflow of a full stack web application and the form examples, brainstorming was required in regards to what properties this Scrabble-like form would have. The form itself would be created as a model in the MVC application and each property would be created as a model as well. Every important property of the game would need to be presented as a model in the application which would allow the application to perform logic with the model. In addition, each instance of a model would be stored in the database. This would allow users to execute changes in the game and for other users to pick up the new version of the game, essentially creating a loop of getting the current game data, updating game with new data and passing it to another user to continue updating it. The advantage of a database also means that users are not required to leave their clients working – just visiting the correct address in a browser at any time would provide them with the latest version of the game.

Many hand-written drafts were made before and during the development phase. These are the models and their representation in the database in the final version.



Most models have links to other models that can be used to eventually retrieve the data related to every component of the game. Too many links may cause a security and maintenance issue but these were not priority goals of the project.

It is important to note that there is no perfectly correct version of a design. This design is not ideal and can be improved in regards to missing links or too many links but it was good enough to finish this project.

The following is a quick description of each model. More detailed documentation about the models, properties and methods can be found in the source code and the evolution of the models within the git commit history.

Model	Description
Game	Model of the highest level, it is essentially a complete form with many properties. Contains all relevant information about the game.
Board	Represents a board. A board has a collection of board tiles. While the collection is represented as a list, it is often converted to a 2D array for performing logic. Keeping it as a list allows EF Core to automatically create the necessary database entries that the list is involved in upon the initial migration (covered in Implementation section).
BoardTile	Represents a tile on a board. A BoardTile has a type (normal, double word etc.) and it might either be empty or have a letter (CharTile) placed on it.
BoardTileType	Represents a type that a BoardTile might be of.
CharTile	Represents a tile/square with a letter and a score.
GameLanguage	Represents a language and the language specific CountOfLetters (English has 26 letters in the alphabet, Bulgarian has 30 letters etc.).
Pouch	Belongs to a game and contains a collection Pouch_CharTile objects (which represent the collection of tiles this pouch has).
Rack	Belongs to a Player and contains a collection Rack_CharTile objects (which represent the collection of tiles this rack has).
Player	Represents a player. Can be human or bot (bot can be High_Scorer or Rack_Balancer). Belongs to a game, has a shared Pouch and a unique Rack. Has a collection of moves.
Move	Contains information about a made move such as its start and end location,

	word played, score etc. A play by a player can have more than one word played and MoveNumber is used to group all those words as a part of one move.
BotType	Represents a type of computer player and the strategy the player will use.
WordDictionary	Represents the connections between all CharTile-s and the language they belong to.

The models were the first pieces of the application that were developed. Once they were written, a database had to be created to accommodate any instances of the models which is covered in the next section.

4.7.2 Server-database communication

Communication between the server and database happens through a ScrabbleContext object which implements EF Core's functionality. The object allows the server to pull database entries as objects and to push objects as database entries. Normal SQL queries are not used. More details are available in the Implementation section of this report which reveals how the database is initially generated, how it is seeded, how it is manipulated and what the ScrableContext object is.

4.7.3 Client-server communication

In modern web applications, clients send HTTP requests to a server to make changes to a form and the server sends an HTTP response indicating if the change was successful or not [32]. This application works on the same principle. HTTP request and responses contain information (mostly in text) regarding an activity. For example, in a request a client might ask for a text file and in the response, a server might say that the text file is available or it is not available. In this application, the messages contain information about the game or changes to the game. If the user places tiles on the board and clicks Submit, the information sent will contain the letters the user has played and where the user has played them. A server will use that information and either tell the user that the move is wrong (due to an invalid English word for example) or send the user information about the new state of the game, which the client uses to update the page for the user. The requests and responses are sent and received through AJAX calls from the client which are generated by the jQuery library that the application is using on client-side. AJAX is covered in the following section.

4.7.4 Front-end design

As mentioned in the MVC section, this web application is a single-page application. This means that when a client asks for a change in the game and the server brings back the changed game, the page that the client is currently on gets updated with the new version of the game. This design is used to keep the number of pages for development to a minimum and to provide a smoother experience for the user using the client. This dynamic update is possible through the use of AJAX.

AJAX stands for Asynchronous JavaScript And XML and brings the following capabilities:

- Read data from a server after a page has been loaded
- Update a page without reloading the page
- Send data to a server in the background [33]

The way it is used in this application is to send information to the server and wait for a response without leaving the page. While waiting, all the page elements become locked behind a loading spinner while the client is waiting for a response from the server and once a response is received, the spinner disappears and the contents of the page are updated in an animated fashion – all without leaving the page, resulting in a better user experience.

The following figure is an example of an Ajax call using jQuery notation. This command resets the game and updates the web page. If an error happens within the server, the status message is changed instead to show the error.

```
$(document).on("click", "#resetGame", function () {
    $.ajax({
        url: '/Scrabble/ResetGame',
        async: true,
        type: "POST"
    }).done(function (view) {
        var viewBody = view.substring(
            view.lastIndexOf("<body>"),
            view.lastIndexOf("</body>")
        );
        animateHtmlUpdates($("body"), viewBody);
    }).fail(function (jqXHR) {
        updateStatusMessage(jqXHR.responseText, "danger");
        $('.button').prop('disabled', false);
    });
});
```

Figure 23. Example of Ajax call using jQuery notation.

4.8 Project Management

Git was used as the version control system for the project. Version Control is the management of changes to documents, computer programs, websites or other collection of information [34]. It is used to keep a history of the changes in the project and adds the ability to revert the project to a version from the past. This functionality is important for cases where specific feature implementations fail and a roll-back to a working version of the project is needed.

An Agile-Iterative approach was taken for the development of the project. As each feature was developed, it was extensively reviewed and tested before continuing development with a new feature. With this approach, each new feature resulted in a more and more complete version of the project and each version of the project was presentable.

4.9 End of Design and Methodology section

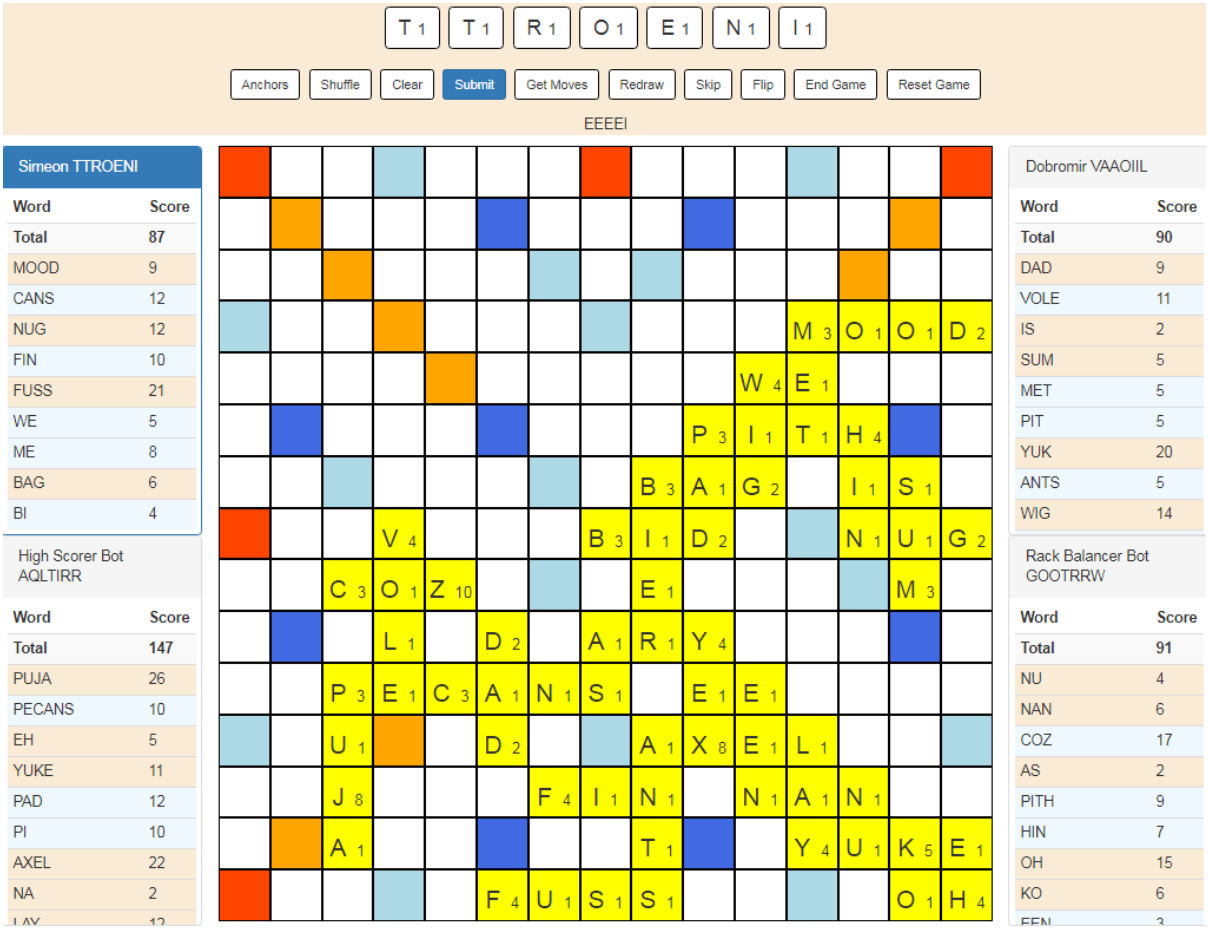
The entirety of the Design & Methodology Section has introduced all the technologies used, described them, and discussed the reasoning behind choosing them and how they work together. There are also introductions to modern day application patterns and designs and based on technologies available and current trends, a design for this application was created. This design was the result of initial brainstorming and eventually improved as development went on. It is not final and it will never be but it has reached a state where a complex application such as this has become possible.

The following section will cover how the game was implemented using said design and will go into more detail as to how each piece of the design was utilised to deliver a final product.

5. Implementation

5.1 Example of a game

This is what a game looks like in the final version of this application.



As the application is using the MVC pattern, the client-side, server-side and database logic were developed in pararell. But to develop them into anything required test data to work with. As the database is the source of data, the first thing that needed to be done was to seed the database with test data to work with. But as there was no database, it needed to be created first. The database generation and seeding processes are covered in the following sections.

5.2 Generating the database

Note: If EF Core is not available in the current ASP.NET Core application, it needs to be installed as a NuGet Package.

EF Core provides the tools necessary for automatically creating a database as long as models and their properties have been correctly provided in the application and creates tables based on that information. When creating the database and tables, it also notices when a foreign model has been used as a property of a given model and alters the table to fit it. For example, if a Game object has a list of Player objects, each Player entry in the Players table will have a property called “GameID” even if no GameID property is actually in the Player model. But by having the properties GameID and Game in a Player model, we can get the ID and instance of the Game object that the Player is participating in. How this foreign-instance-retrieval works is covered in a section called Lazy Loading.

To generate a database with a set of models to be represented as tables, the command “EntityFrameworkCore/Add-Migration Initial” is typed in the Package Manager Console. What this command does is it generates a migration file that contains all the commands to generate a database with the models as tables with properties and configures some of the table fields to contain a foreign key if it is mentioned in the models. The commands are in C# syntax rather than SQL syntax. The next command used is “EntityFrameworkCore/Update-Database” which creates a database with the models as tables based on the migration file. If available, the migration file is loaded with extra commands to insert data (seeding the database) with entries provided in the Context file. Seeding and the Context file are covered in the next section.

The following figure is a snippet of the migration file generated by EF Core. The snippet creates a table with properties, primary key and foreign keys.

```
migrationBuilder.CreateTable(
    name: "Games",
    columns: table => new
    {
        ID = table.Column<int>(nullable: false)
            .Annotation("SqlServer:ValueGenerationStrategy", SqlServerValueGenerationStrategy.IdentityColumn),
        GameLanguageID = table.Column<int>(nullable: false),
        BoardID = table.Column<int>(nullable: false),
        PouchID = table.Column<int>(nullable: false),
        WordDictionaryID = table.Column<int>(nullable: false),
        Log = table.Column<string>(nullable: true),
        IsFinished = table.Column<bool>(nullable: false)
    },
    constraints: table =>
    {
        table.PrimaryKey("PK_Games", x => x.ID);
        table.ForeignKey(
            name: "FK_Games_GameLanguages_GameLanguageID",
            column: x => x.GameLanguageID,
            principalTable: "GameLanguages",
            principalColumn: "ID",
            onDelete: ReferentialAction.NoAction);
        table.ForeignKey(
            name: "FK_Games_WordDictionaries_WordDictionaryID",
            column: x => x.WordDictionaryID,
            principalTable: "WordDictionaries",
            principalColumn: "ID",
            onDelete: ReferentialAction.NoAction);
    });
```

Figure 24. A snippet of the migration file generated by EF Core.

5.3 Seeding a database and the ScrabbleContext file

The ScrabbleContext file is a class that can provide an instance of an object which implements an interface originating from the EF Core framework called DbContext. A ScrabbleContext instance is used in any location within the application in which the server requires interaction with a database, mainly in the controller or some logical parts of the application. The Update-Database command uses an instance of a ScrabbleContext object to generate the database as well as to seed it by running the commands from the migration file. The information that is to be seeded is provided in the OnModelCreating() method of the ScrabbleContext class. The objects in OnModelCreating() are added to the migration file when running the Add-Migration command and eventually added to the database with the Update-Database command.

Many entries to be seeded rely on the existence of other seeded entries before getting seeded. This recursive requirement stops when seeding the CharTile entries. The CharTile table is never edited or changed and the information in is required to have been hard-coded by the developer. Most other entries in the other tables can be auto-generated when making a new game and when providing enough input about the type of game we want but the CharTiles table will always exist like it is. It is the root that all other tables and entries build upon.

P.S. The functionality to create games through a user interface has not been implemented. To create a new game, the proper seed information needs to be provided in the ScrabbleContext file or add/change data of the database through the use of Visual Studio's SQL Server Object Explorer.

P.S2 In the database, it can be seen that each CharTile has two versions – one with a score and one with a score of 0. The duplicates with a score of 0 are possible transformations of a blank tile (a blank tile can become any letter but always has a score of 0). Performing transformations of a blank tile and reflecting that transformation back into the database proved troublesome and duplicates with a score of 0 seemed to be the best solution at the time. If it was to be coded again, I would try to keep information about any transformations in a separate table.

The following figure is a snippet of the OnModelCreating() method in the ScrabbleContext class. The commands will add Pouch entries with some properties in the Pouches table when executed.

```
modelBuilder.Entity<Pouch>().HasData(new Pouch { ID = 1, GameID = 1 });
modelBuilder.Entity<Pouch>().HasData(new Pouch { ID = 2, GameID = 2 });
```

Figure 25. Snippet of the OnModelCreating() method in the ScrabbleContext class.

5.4 Client-server-database workflow

When a client accesses the address of the application or performs an action within the provided page, an HTTP message is sent to the controller of an application. A controller may contain multiple methods and depending on the message received, the relevant method will be called. The method called will, in most cases, load a Game object from the database, make changes to it, push the changes to the database and send a reply to the client that sent the message. The reply may be an error message or an updated web page with the updated details of the Game object.

In this application, the Game object that is loaded is hard-coded into the controller methods as there is no front-end implementation of game selection.

Retrieving database entries as objects

By providing the type and ID (or any other information) of a database entry we'd like to have as an object to perform logic with to the ScrabbleContext object, EF Core finds the entry with the given ID in the given table and returns an object whose type is that of the table it was pulled from (pulling an entry from the Games table will return a Game object). The returned object's properties will be the same as the ones that they were in the database.

But there is a catch. For example, in a Game model we have a BoardID and Board model as properties. While the BoardID is a simple integer property in the Game entry, Board is a property that has its own entry in the Boards table and has its own models as properties. Because it is not a simple type such as an integer, string or Boolean etc., the Board property of the returned Game

object will be null. But by having the BoardID property, we know the entry of the Board we would like to load from the Boards table and we can pull it at the same time we pull a Game object rather than trying to pull a Board object from the Boards table separately. This is done through a technique called lazy-loading.

The following figure is an example of a Game object being pulled from the database. The object that is pulled matches the entry in the Games table that has an ID of 1.

```
Game game = _scrabbleContext.Games.Single(g => g.ID == 1);
```

Figure 26. A Game object being pulled from the database.

5.5 Lazy-loading

Lazy-loading allows the ScrabbleContext object to pull an object as well as the objects it contains in a single query [35]. It is installed as a NuGet package for this implementation. For example, when pulling a Game object from the Games table, the Game object will have a complete Board object as well. And this technique works at every level of a call. To expand the example, when a Board object is pulled through a query for a Game object, the Board object will also contain the full list of BoardTiles objects. This relationship is established by having the BoardID property available in each BoardTile entry which is why seeding the database for the first time with complete information is very important. In summary, as the Game object has (or can eventually get) the knowledge about every object used in the application, by pulling a Game object we essentially pull every object that is related to that game – board, board tiles, players, moves etc. It saves the effort of pulling every object separately and populating a non-lazy-loaded Game object with those objects.

The following figure examines the Game object from Figure 26 and it can be seen that a Board object and its collection of BoardTiles object have been lazy-loaded and part of the Game object.

Name	Value
this	{Scrabble.Controllers.ScrabbleController}
game	{Castle.Proxies.GameProxy}
Board	{Castle.Proxies.BoardProxy}
BoardTiles	Count = 225
[0]	{X0, Y0, _}
Board	{Castle.Proxies.BoardProxy}
BoardID	1
BoardLocationX	0
BoardLocationY	0

Figure 27. Examination of Game object.

5.6 Pushing changes to the database

As most of the objects that have undergone changes due to server logic have originally been pulled from the database, the objects are already connected to the correct entry in the correct table. Thus reflecting changes done to these objects to the database is as simple as calling the SaveChanges() method of the ScrabbleContext object. This method pushes all changes (or additions or deletions) of objects to the database. Not calling this method would result in no changes in the database by the end of the object manipulation by the server.

The following figure demonstrates how an object originally pulled from the database is modified, and how SaveChanges() is used to save all changes of the objects in the database.

```
Game game = _scrabbleContext.Games.Single(g => g.ID == 1);
var currentPlayer = game.GetPlayerAtHand();
currentPlayer.SkipsOrRedrawsUsed++;
game.SwitchToNextPlayer();
_scrabbleContext.SaveChanges();
```

Figure 28. Example of object pulling, modification and database update.

5.7 Web page structure

This is a single-page application and the content displayed on the page is a reflection of a Game object (loaded from the database) and the objects it is related to (players, racks, pouch, moves etc.). When looking at the HTML code of the page, it can be seen that many of the elements have their own ID and one or multiple classes. This information is provided by the Razor page which generate san HTML page with attributes that originate from a Game object. Having accurate descriptions of each page element is important as when a user submits their inputted data to a server, the message that is sent is a collection of the page elements that have been changed. When the server knows the attributes of the changed elements, it can find the objects with matching attributes within the game object by parsing the data that comes from the client. As the web page is a reflection of a Game object, by correctly making changes in the Game object and storing them in the database, the next time the web page is updated it will display the contents of the new version of the Game object.

The following figure is a snippet of the web page structure. The underlined sections mark the structure of the board tile with letter U and the board tile to the left of it. The attributes are used for styling, logic performing and data traversal from the client to the server.

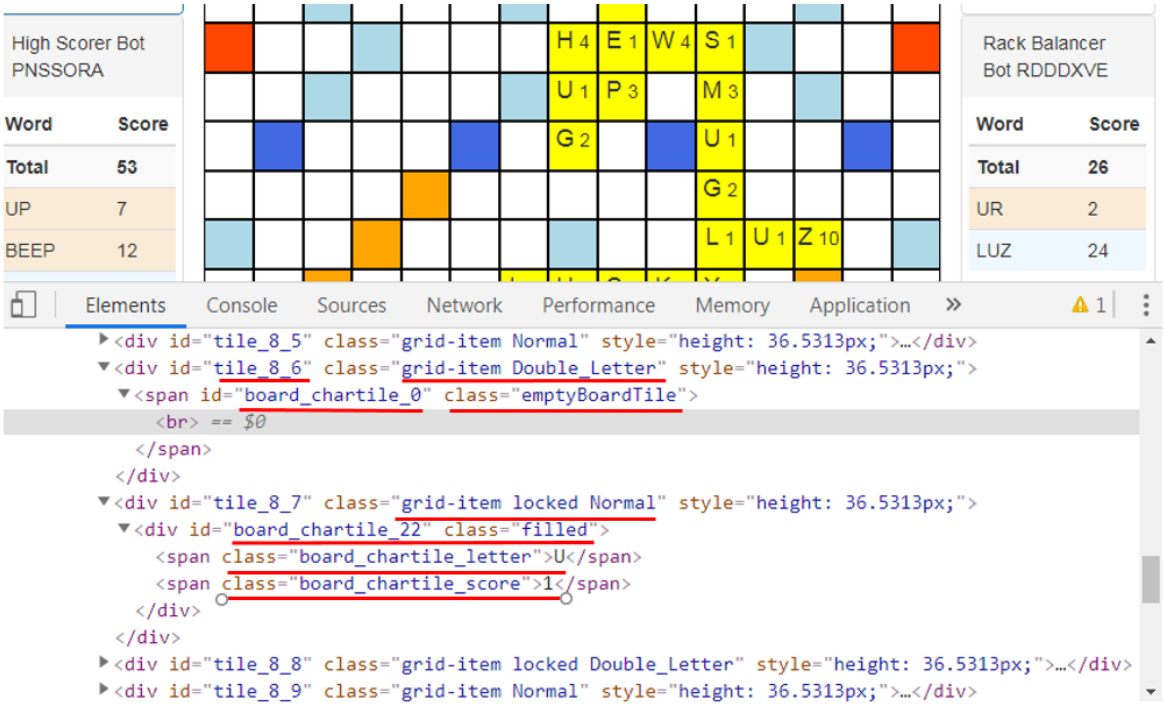


Figure 29. A snippet of the web page structure.

5.8 Client-side and server-side validation

In web application development, an important topic that needs to be discussed is validation of user input. There are cases when a user will provide incorrect data, such as when playing a word that is not valid or playing a word that is not connected to a different word on the board. These mistakes need to be validated and the validation can be done from two parties – either the client itself or the server – and there are advantages and disadvantages to using just one of the parties.

Usually, when a user uses a browser to access a web page, the browser loads a web page as well as any external files that are related to the styling or logic of that page. This application loads additional CSS files and a Javascript file. The CSS file is responsible for the styling of the page and the Javascript file is responsible for the logic. The Javascript file contains a collection of functions that are called whenever a user performs an action such as clicking a button. There are many functions contained in the JS file but the main one that performs the validation is the one that is called when pressing the "Submit" button. By clicking that button, the user is attempting to put new data into the database and that is why the server (form provider) needs to be sure the data is correct and not wrong or malicious.

5.8.1 Types of Validation

5.8.1.1 Client-side validation

Client-side validation is performed on the user's device. The JS file that is provided with the loading of the web page is provided by the server. In order for the client to send data to a server, the data needs to pass the checks that are validated through the functions of the JS file. If the checks are passed, the data is sent to the server and the server will perform the changes without double checking the data (unless server-side validation is performed as well). If the data is incorrect, no communication with the server is attempted and instead, the client receives an error stating that the data provided is incorrect.

The advantage to client-side validation is that it is much faster to validate as the checks are stored on the client's web browser. In the case of a mistake, the user would not need to wait for a response from the server to understand that his/her actions were incorrect which provides a smoother experience. The disadvantage is that the validation logic is stored on the client and the user, with enough computing expertise, can modify the validation or change the data after it has passed validation and before it has been sent to the server. In other words, the effect of the validations can be nullified and incorrect data may get sent to the server. If the server does not implement validation of its own, the changes in the Game object, and eventually the database, may eventually break the game. This also allows the user to send data that may expose a security flaw in the server.

The following figure demonstrates client-side validation. Attempting to play the letter R and H results in an error as they are, first of all, not connected to an existing word on the board and, in addition, are not connected to each other. This validation is done on the client-side and the server is not informed of the user's actions.

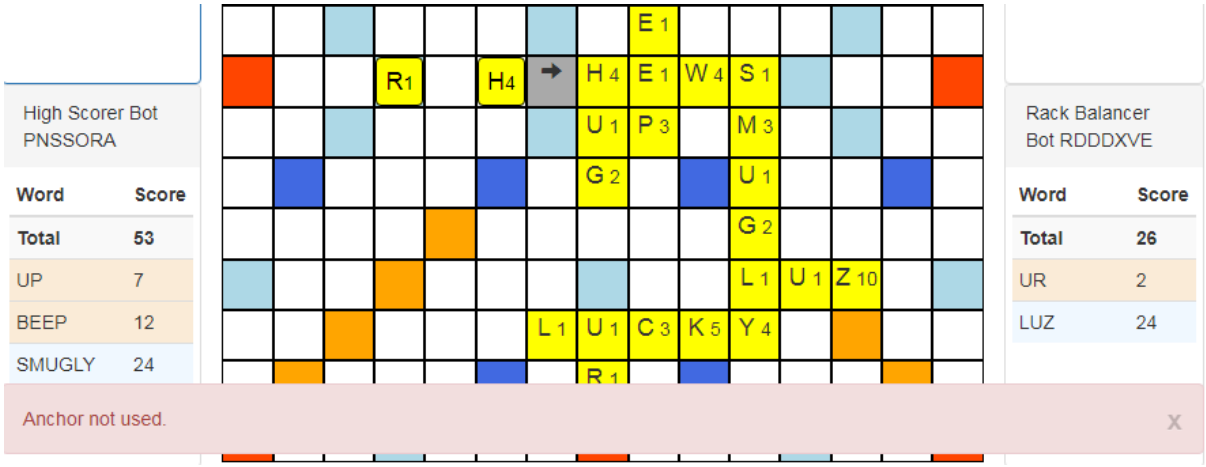


Figure 30. Example of client-side validation.

5.8.1.2 Server-side validation

Server-side validation is perofrmed on the machine that is the host of the game and has provided the game to the clients. When a client sends data to the server, the server will perform logic to confirm that the data received is valid (for example a correct move that is a valid word and follows the rules of the game). If it is valid, it will update the Game object and push the changes to the database, otherwise it will not change the object or database and will instead return an error to the client.

The advantage to server-side validation is that a user has no access to the server and the validation checks performed on it. If the user sends data that is incorrect, the user would not be able to bypass the validation of the server and it would be much less possible to damage the game. Another advantage is that the client would not be required to load a great amount of data that would be related to logic and validation. The disadvantage is that in all cases of move validity, a message would be sent to the server and a reply would be expected back. The process of communication with the server in this application happens through an Ajax call. As the page would get locked during an Ajax call, the user would be forced to wait for the call to complete before being able to interact with the page again. This would leave to a poorer user experience.

The following figure is an example of server-side validation. Attempting to play the word HHEWS results in an error as it is not a valid word. The validation is performed on the server as only the server has the dictionary that is consulted with to check word validity. Providing the client with the dictionary would result security and cheating dangers.

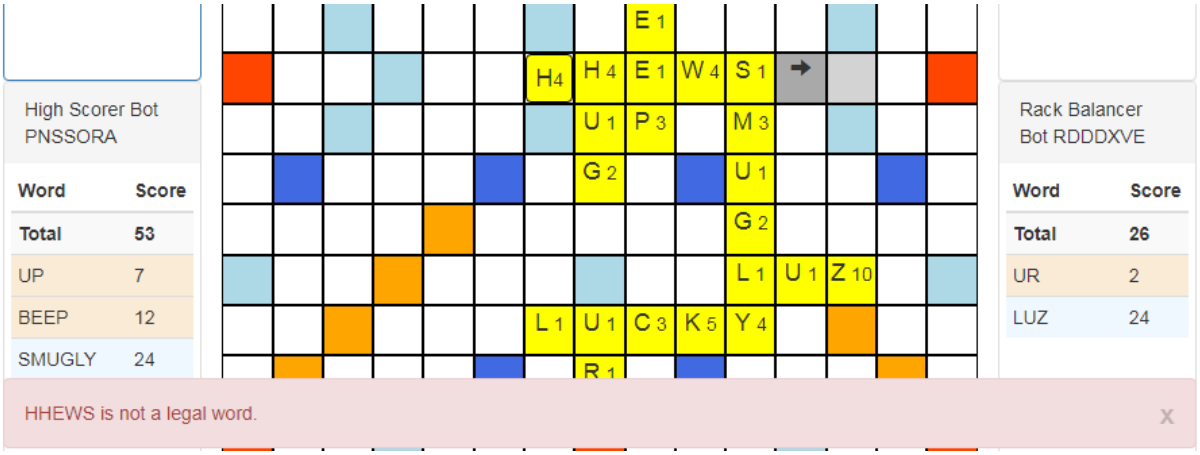


Figure 31. Example of server-side validation.

5.8.2 Solution to Validation Problem

This application performs check on both client-side and server-side. Some of the checks are repeated on both sides while some are exclusive to the client or server. Repeated checks include validating if a move is connected to a different word, if it is connected, if it is played from the start tile at the start of a game etc. Server-side validation repeats those checks (in the case that a malicious user has tampered with the data after it has passed the client-side checks) and checks if a played word is valid. Word validation is performed on the server as the dictionary is stored on the server and not provided to the client. Providing the dictionary to the client imposes a great risk in regards to security and cheating.

5.9 Helper class

The Helper file is a static class that contains many of the methods used within the application to validate moves, modify a board, generate a move etc. Its goal is to contain methods that are not model-specific and instead interact with multiple models as well as to contain more general methods that would not fit under any model. More detailed breakdown of the goal of each method is available in the source code documentation but this is a quick summary.

Method	Description
GetWordScores()	Uses some of the below methods to load a dawg (dictionary), get the input of a user (the words that the player made and the rack tiles that the player used), get the details of the used board tiles, check the validity of the words, get input's score and various attributes, reflect changes in the Game object (for example in its Board and BoardTiles objects, game racks and pouch etc..). Returns an error if it fails.
LoadDawg()	Returns a DAWG which is a special data-structure containing the dictionary of a language (covered in greater detail in DAWG-related sections).
GetPlayedWords()	Parses input from the client to get the words that the user played.
GetPlayedRackTiles()	Parses input from the client and returns the tiles the client played from their rack.
GetTileDetails()	Based on parsed input from client, gets a board tile location and the CharTile on it.
CheckWordValidity()	Checks if a word played is valid against the DAWG.
GetWordScore()	Gets the score of a played word based on the board tiles it was played on.

The following methods are more general or used for the move generating component of the game.

Method	Description
GetValidCrossChecksAndAnchors()	Gets anchors of a Game object and computes cross-check sets for each board tile (covered in greater detail in MoveGenerator sections).
GetValueFromAjaxData()	Parses input from the client and returns a specified piece of it.
GetMoveGenerator()	Returns a MoveGenerator object based on the state of a Game object. Responsible for generating possible moves on a board (covered in greater detail in MoveGenerator sections).
GetVerticalPlays()	Given a board tile with a letter on it, checks the vertical word that the tile is part of if any.
Shuffle()	Shuffles an ArrayList with objects.

The Helper class and the methods contained within it and the methods contained within the models cover most of the functionality needed to play a game with human players. Such a process involves a user playing a word, the client and server validating it, the server updating the Game object and the database and returning a page that contains the updated game and details relevant to the next player. An important part of this game flow, as well as in a game flow including computer players, is the structure containing the dictionary, which in this project is a Directed Acyclic Word Graph (DAWG).

The following figure shows a snippet of the Helper class. It contains most of the major methods that deal with parsing user input, validating words, calculating word scores and providing data for move generation.

```
13  namespace Scrabble.Helpers
14
15      public static class Helper
16      {
17          // public static void MakeEnglishDictionary() ...
33
34          /// <summary> Loads a DAWG (a dictionary of words as a compact trie).
39          public static Dawg<bool> LoadDawg(GameLanguage language)...
45
46          /// <summary> Using word, checks if word is valid in dictionary
53          public static bool CheckWordValidity(Dawg<bool> dawg, string word, bool alwaysExists = false)...
62
63          /// <summary> Gets the score of a played word on a board
68          public static int GetWordScore(List<BoardTile> wordBoardTiles)...
109
110         /// <summary> Gets all played words from POST request ajax data
115         public static string[] GetPlayedWords(List<KeyValuePair<string, StringValues>> data)...
131         // public static Dictionary<BoardTile, List<CharTile>> GetValidCrossChecksOneWay(BoardTile[,] boardArray, Wor
193
194         /// <summary> Goes through a board and gets all anchors and crosschecks for boar ...
201         public static void GetValidCrossChecksAndAnchors(BoardTile[,] boardArray, WordDictionary dictionary, Dictiona
322
323
324         // public static Dictionary<BoardTile, List<CharTile>> GetValidCrossChecksCombined(Dictionary<BoardTile, List
364
365         /// <summary> Gets inforamtion from data object from POST request ajax object
371         public static string GetValueFromAjaxData(List<KeyValuePair<string, StringValues>> data, string property)...
383
384         /// <summary> Gets the played rack tiles of the current move
389         public static string[] GetPlayedRackTiles(List<KeyValuePair<string, StringValues>> data)...
401
402         // public static void GetBoardArrayFromHtml(List<KeyValuePair<string, StringValues>> data) ...
421
422         /// <summary> Returns move generator
429         public static MoveGenerator GetMoveGenerator(Game game, List<Move> gameMoves, int timeLimit = 0)...
453
454         /// <summary> Gets details of a board tile submitted form the web page as a stri ...
459         public static string[] GetTileDetails(string tile)...
```

Figure 32. A snippet of the Helper class methods.

5.10 Directed Acyclic Word Graph (DAWG)

As discussed in the literature review, the DAWG is a compressed version of a trie (tree-like data structure) of a dictionary in which repeating states are merged together. The condition for repetition is that two states have the same continuation. A state (or node) may be terminal or non-terminal and contain a set of words. A terminal node contains only valid words, otherwise it is non-terminal.

The following figure is an example of a DAWG. The nodes (or states) with an outer circle are terminal nodes. CAR, CAT, EAT, EAR and DOG have been put in the same node as all five words have the continuation with letter S and are valid i.e. can be put in a terminal node. C and E have been put in the same node as they have the same continuation of with letter A.

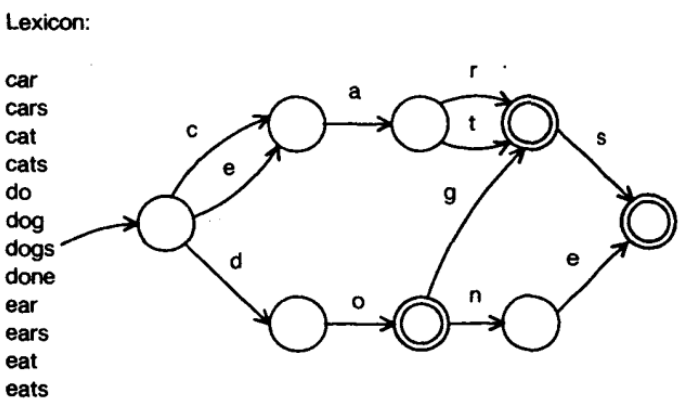


Figure 33. Example of a DAWG.

5.10.1 Traversal of DAWG example

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). This can be explained in the following process. If the DAWG is asked if the word CAT is in the dictionary, these are the steps that happen.

- Starting at a root node (empty word), the DAWG is checked if there is any path from the root node labelled by C. As there is no word yet, all characters in the alphabet are available as paths. The path labelled by C is followed to lead to the non-terminal node containing the word C. A non-terminal node does not contain any valid words, a terminal node contains only valid words.
- Starting at node C, the DAWG is checked if there is any path from this node that is labelled by the letter A. It is available (along with other paths). The path labelled by A is followed to lead to the non-terminal node with the word CA.
- Starting at node CA, the DAWG is checked if there is any path from this node that is labelled by the letter T. It is available (along with other paths). The path labelled by T is followed to lead to the terminal node with the word CAT.
- If the path of the input word has been fully traversed and the node with that word reached, the word will be marked as valid if the node it belongs to is terminal. Otherwise it will be returned as invalid.
- If the path of the input word has not been fully traversed, the word will be returned as invalid. This is due to the unavailability of paths labelled by any of the letters of the word to follow. In a fastest case scenario, a word may be marked as invalid when trying to follow a path from a node that is preceded by the root node (which has paths for all letters).

5.10.2 Comparison to GADDAG

The GADDAG is the structure developed by Steven Gordon and is described by him as a two-way DAWG. The analysis comparing DAWG and GADDAG in the literature review stated the advantages and disadvantages to each structure and that they are both used for Scrabble implementations. I personally had a preference for GADDAG as it is faster and the testing of the implementation has revealed that it might have been a better choice. But as they are both good structures and future results were unknown at the time, the DAWG structure was selected for the implementation as an implementation of it was available as a NuGet package called DawgSharp and, with the inclusion of the package, development time for the project was drastically reduced.

5.10.3 DawgSharp

The DawgSharp package is capable of building a DAWG out of a text file containing valid words [36]. A DAWG object generated by the package is very similar to a HashSet (as stated by the author and discovered from personal debugging). It is similar as it contains unique words and a word's validity can be checked by attempting to retrieve it by using the word as a key (for example dawg["CAT"] will return True while dawg["CATT"] will return False). According to the author's page, this implementation of a DAWG containing 2 million words consumes only 2Mbytes of RAM.

As the source code for the package is not available, it cannot be proven how closely the author's implementation of the DAWG follows the one from Appel and Jacobson upon which the performance complexity of O(n) was established. The reason for the suspicion is that Appel and Jacobson used a compressed trie while the author is using a HashSet (bucket-based structure).

Nevertheless, the author's implementation offers useful methods and their functionality is based on the principle that the structure being explored is a compressed trie such as the one from Appel and Jacobson. The most useful method is MatchPrefix() which takes a word and returns all words that contain the given word as a prefix from the dictionary. This method has been used to great effect in the move generation process which follows the idea of working with nodes and traversing available paths to different nodes. It is covered in more detail in the MoveGenerator section of the report.

The dictionaries for the game are provided within the Helper folder of the solution. The English dictionary is built from the englishWords.txt file.

The following figure is an example of a DAWG as provided by the DawgSharp package. The DAWG object contains all valid words and the validity of a given word can be retrieved by using the word as a key and to attempt to retrieve its value if it exists. P.S. AA, AAH and the following words are valid words from the dictionary.

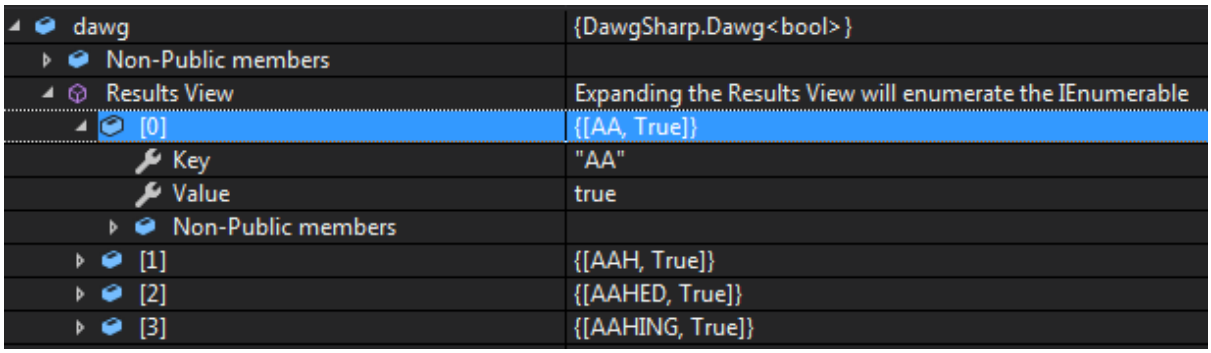


Figure 34. Example of a DAWG as provided by the DawgSharp package

5.11 MoveGenerator - Constructing

The MoveGenerator object is responsible for building a list of all possible moves a player can make given his/her rack and the state of the game and board.

When Appel and Jacobson wrote their paper "The World's Fastest Data Structures", they wrote about an efficient data-structure for storing a Scrabble dictionary, an algorithm to generate all possible moves on a board and a set of precomputed limiters for that algorithm to limit its breadth and depth (to avoid computing moves that would break the rules of the game) and to make it more efficient. As the DAWG structure from Appel and Jacobson's paper was used for the dictionary implementation, it seemed sensible to expand this project with the techniques and algorithm from the same paper.

To create a MoveGenerator object, the precomputed limiters need to be provided first. It is also important to understand how the algorithm from Appel and Jacobson's paper works. It will be covered briefly in this section and in more detail in later sections.

Without any limitation, the algorithm works by generating all possible words with all possible letters in the alphabet from all empty board tiles on the board by expanding each board tile with a combination of letters to the left and to the right. Without any limitation, this task would use a tremendous amount of time and provide fruitless results for most of it as the moves would be breaking the rules of the game. Its productiveness can be improved with the following limiters:

- Instead of using all letters in the alphabet, use all the letters that the current player has in his or her rack.
- Instead of using all empty board tiles for expansion, use all board tiles that are connected to a word already on the board (known as anchors).
- Instead of using all letters from a player's rack, use all the letters from the rack that would also not result in an invalid vertical move (respect the cross-check sets for the board tiles used in a move).
- As the algorithm generates horizontal moves, repeat the algorithm on a transposed (vertical-ized) version of the board to get all vertical moves for the original (untransposed) board. This means that board transposition must be done.

The limitation techniques are discussed below.

5.11.1 Algorithm Limitation Techniques

5.11.1.1 Board Transposition

A key technique from the paper. The algorithm only does horizontal expansions on a board and only comes up with horizontal moves and not vertical. To make the algorithm come up with vertical moves that are applicable to the original state of the board, the algorithm needs to work with a "vertical-ized" version of the original board. This is where transposition comes in.

Given a two-dimensional matrix, transposition is the act of providing a new version of the matrix where the rows are displayed as columns and the columns are displayed as rows [37]. As the board can be considered a matrix, transposing it would provide a board where all the horizontal words appear as vertical and all the vertical words appear as horizontal. It is essentially the same board but looked from a different angle. As the algorithm works in a horizontal fashion, having the algorithm build moves based on a transposed board would generate horizontal moves for the transposed board which would be equivalent to vertical moves for the untransposed (original) board. By running the algorithm against an untransposed board and a transposed board, the program can get both the horizontal and the vertical moves possible for the original (untransposed) board.

It is important to note that in Scrabble, a horizontal word is read from left to right and a vertical word is read from top to bottom. On a transposed board, the previously horizontal words become vertical and are the same words as on the untransposed board if read from top to bottom. The same applies for vertical words from the untransposed board – they become horizontal on the transposed board and are the same words as on the untransposed if read from left to right. The algorithm works based on these given facts about horizontality and verticality.

Transposition is a key technique for the implementation of the algorithms and each of the following checks need to be executed for both untransposed and transposed variants of the board to get a full list of valid moves.

The following figure shows the difference between a normal (untransposed) board and a transposed board where horizontality and verticality have been flipped.

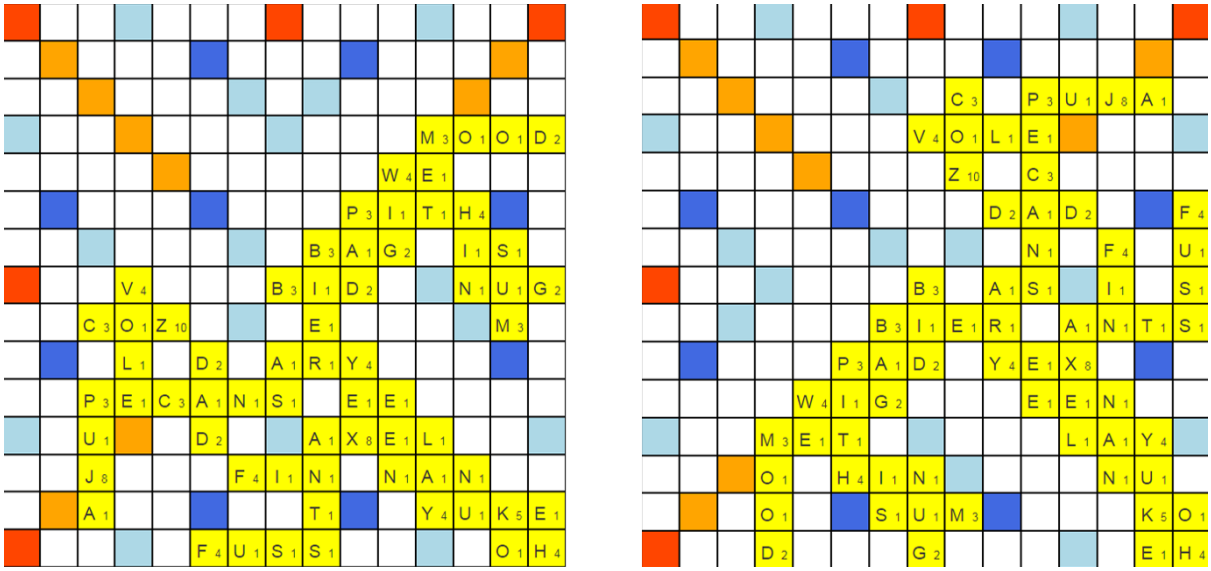


Figure 35. An example of a board in its normal untransposed variant (left) and its transposed variant (right). The horizontal words have become vertical on the right and the vertical words have become horizontal on the right.

5.11.1.2 Anchors

An anchor is an empty board tile that is adjacent to a busy board tile (has a letter). A condition for a move to be valid is that it must be connected to an already existing word or tile on the board. As the algorithm works by expanding an existing element in both directions, it needs to know which locations on the board it can use to expand upon. Generating a move of which the used board tiles are not connected to any other taken board tile would result in an invalid move in all cases and would be an unnecessary usage of time. By providing an anchor (or a list of anchors) to the algorithm, it will try to expand and generate words from said anchor. Not all the words that are generated from the anchor will be valid but none of them will be breaking the condition stated above in regards to connectivity which greatly reduces the scope of search of the algorithm.

The following figure displays all anchors on an example state of a board.

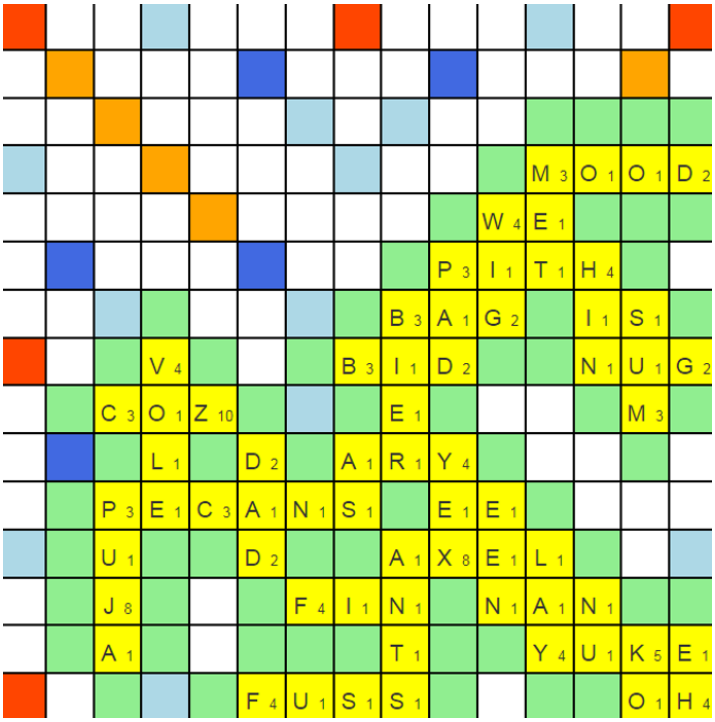


Figure 36. The tiles marked in green are the anchors in this board as they are adjacent to already placed tiles on the board either vertically or horizontally.

5.11.1.3 Cross-check sets

A cross-check set is a set of letters for a board tile that would be a valid placement for that board tile considering the collection of letters that are located above and below the said board tile. As the algorithm expands a word in a horizontal fashion, it does not consider if one of the letters of the generated word participates in the creation of an invalid vertical word at any point along its length. Cross-check sets add that consideration.

To do this, it is important to note that a generated horizontal word may include multiple generated letters on a single row but only one letter can be placed on any column of the board when playing that move. The rules of Scrabble state that a word can only either cover a single row and multiple columns (horizontal) or multiple rows and a single column (vertical). With this knowledge it is understandable that a horizontal word can add a maximum of one letter to the columns it covers and that a vertical move can add a maximum of one letter to the rows it covers.

Empty board tiles that are not vertically connected to any word do not have a limitation regarding cross-checks i.e. any letter can be placed on those tiles and it would not result in a vertically invalid move. Empty board tiles that are vertically connected to a word will contain a cross-check set. The set will be either empty or will contain a set of letters. If it is empty it means that whatever letter is placed on that board tile as a part of a horizontal move would result in an invalid vertical move which would make the whole move invalid. If the set contains letters it means that the letter placed on that board tile as a result of a horizontal move must be part of the cross-check set for that board tile in order for the whole move to not be marked as invalid.

With the cross-check sets for each board tile being known to the algorithm, the work load of the algorithm can be reduced. The algorithm is advised on what letters to avoid placing on board tiles when generating moves and thus it would not keep exploring fruitless opportunities for moves.

The following figure is an example of cross-check sets for a set of tiles. Hx tiles are used for horizontal moves and Vx tiles are used for vertical moves. For H0, the vertical cross-check set is {A, O}. If a horizontal move is played using tile H0 (in direction from H0 to H4), the only letters that can be put on H0 are A and O as AB and OB are the only valid vertical words that can result from a valid horizontal move. For V0, the horizontal cross-check set is {}. No vertical move can be made using V0 as no single letter can be a prefix of the word BOING thus0 any single letter placed on V0 would result in an invalid move.

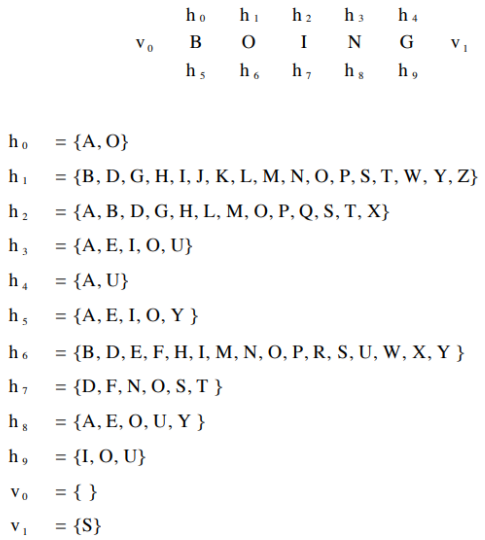


Figure 37. An example of cross-check sets for a set of tiles.

5.11.2 Conclusion

With the above limitations in mind, the original description of the algorithms is turned into the following one:

The algorithm works by generating all possible words with all letters in a player's rack from all anchor board tiles on the board by expanding each anchor with a combination of letters to the left and to the right while attempting expansions with a letter only if the letter respects the cross-check set of the board tile it has expanded upon.

It is important to note that an untransposed board and a transposed board will have different coordinates for anchor tiles and a board tile may have a different cross-check set for either an untransposed, transposed or both versions of the board. That is why when constructing the MoveGenerator it needs to be provided with both boards, anchor coordinates for both boards and cross-check sets for both boards. The constructor also includes the rack of the player generating the moves, a DAWG with the game dictionary and an empty list of valid moves which will be updated as more moves are discovered by the algorithm.

With the above limiters provide, they are used to create a MoveGenerator object which has a method for generating moves while respecting the given limitations. The next section will discuss how the algorithm works when it knows the state of the board, the dictionary to check for word validity and the limits imposed by the limiters.

P.S. The constructor also includes a time limit which stops the algorithm as soon as it is stoppable once the given time has passed.

5.12 MoveGenerator – Generating Moves

5.12.1 GetValidMoves()

The GetValidMoves() method of the MoveGenerator object employs a custom implementation of the methods from A&J's algorithm (LeftPart() and ExtendRight()). The only parameter it takes is a Boolean which tells the the MoveGenerator to generate moves either for an untransposed or a transposed board. The original version of A&J's algorithm is given a single board tile to expand upon while the GetValidMoves() method calls the algorithm multiple times with each time using a different anchor to expand upon. GetValidMoves() also employs checks to see if the time limit has been reached and shuffles the order of the anchors passed to A&J's algorithm. With a time limiter and anchor selection shuffling implemented, an element of randomness is added to the MoveGenerator and each call will result in a different list of valid moves and thus a more unpredictable play from a computer opponent.

A&J's algorithm works in a recursive fashion. Each expansion call either to the left or to the right calls another expansion call until the player's rack is left empty from expansions or until all possible letter combinations in a given direction have been tested or until a board border has been reached.

While expansions happen depending on a player's rack and the cross-check set for a tile that is to be expanded upon, an additional condition is a check to see if the letter selected for expansion is a valid continuation for the word created up until this point of the expansion. This is done by using the MatchPrefix() method of the DAWG which, with some customization (in MatchPrefix() section), returns all possible paths (letter continuations) of a given word. For example, if a board has the word CAT on the board and the letter in a player's rack K passes the cross-check set, MatchPrefix() will tell us if any words exist starting with CATK. If such a word does not exist, expansion with K will not be attempted and instead the next letter in the player's rack will be attempted for expansion with the same prefix checking. Eventually all letters in the rack will be tested.

There are assumed limitations in A&J's algorithm which are stated in their paper. One limitation is that an anchor is not expanded to the left if there is a word already placed to the left of the anchor. This is done to avoid the case of duplication of generated moves when the two anchors on the left and right of a horizontal word are used for expansion. If an anchor is on the left of a word, the existing word will be tested with various prefixes in front. If the anchor is to the right, no additional prefixes will be used for the existing word.

With this limitation in mind, for each anchor selected for expansion, GetValidMoves() checks how much a word can be expanded to the left. If there is a word to the left of the anchor, the limit is 0. Otherwise, up until an existing word or the border of the board is reached, the limit is incremented. After this process, if the limit is 0, expansion to the left is skipped and only expansion to the right is attempted (ExtendRight() is called). Otherwise, each letter of the player's rack is placed to the left of

the anchor (as long as it passes the cross-check set) as expansion to the left is attempted (LeftPart() is called). During each expansion part, valid possible moves are added to a list. Each move has details and a score based on various heuristics covered further in the report. Each expansion places tiles from the player's rack onto the board and puts them back in the rack once expansion is done.

When all anchors have had their expansions completed or when the time limit is reached, a list of possible moves is returned which the computer opponent will use to play a move.

LeftPart() and ExtendRight() are covered below. The next sections discuss fairly complicated usage of recursive methods. While reading, always refer to the following figure for a visual interpretation of the algorithm's mechanics. Zoomed-in figures are provided for each method.

The following figure provides a visual demonstration of the algorithm used for move validation. The left half of the figure demonstrates the ExtendRight() method and the right half demonstrates the LeftPart() method which itself calls ExtendRight() multiple times.

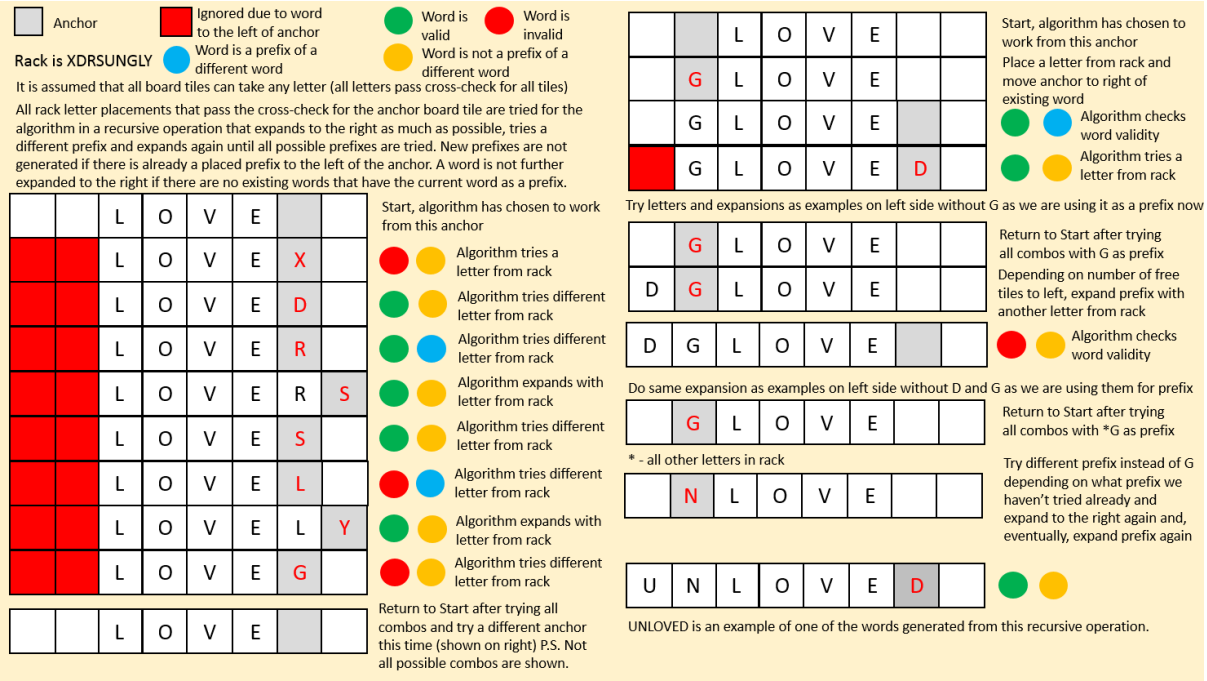


Figure 38. A visual demonstration of the algorithm used for move validation.

5.12.3 ExtendRight()

The first action ExtendRight() will do is to check if the word created up until its call is a valid move. If it is, a GeneratedMove object is added to the list of valid moves containing details about the letters used, board tiles, the rack the player is left with, the board before any move was attempted etc. Once the move validity check is done, expansion to the right is attempted. If there already is a tile placed on the right of the current word, the tile is used for the expansion instead. Recursive ExtendRight() calls follow which all test the remaining letters of the rack and build up on the list of valid moves. Eventually, the original ExtendRight() call will finish and sometime after it (after additional LeftPart() calls most likely) all valid moves for a given anchor will have been generated (or not all depending on time limit). If no time limit is available, all anchors will eventually have the valid moves for them available.

The graph below demonstrates the ExtendRight() method (the left half of the original figure).

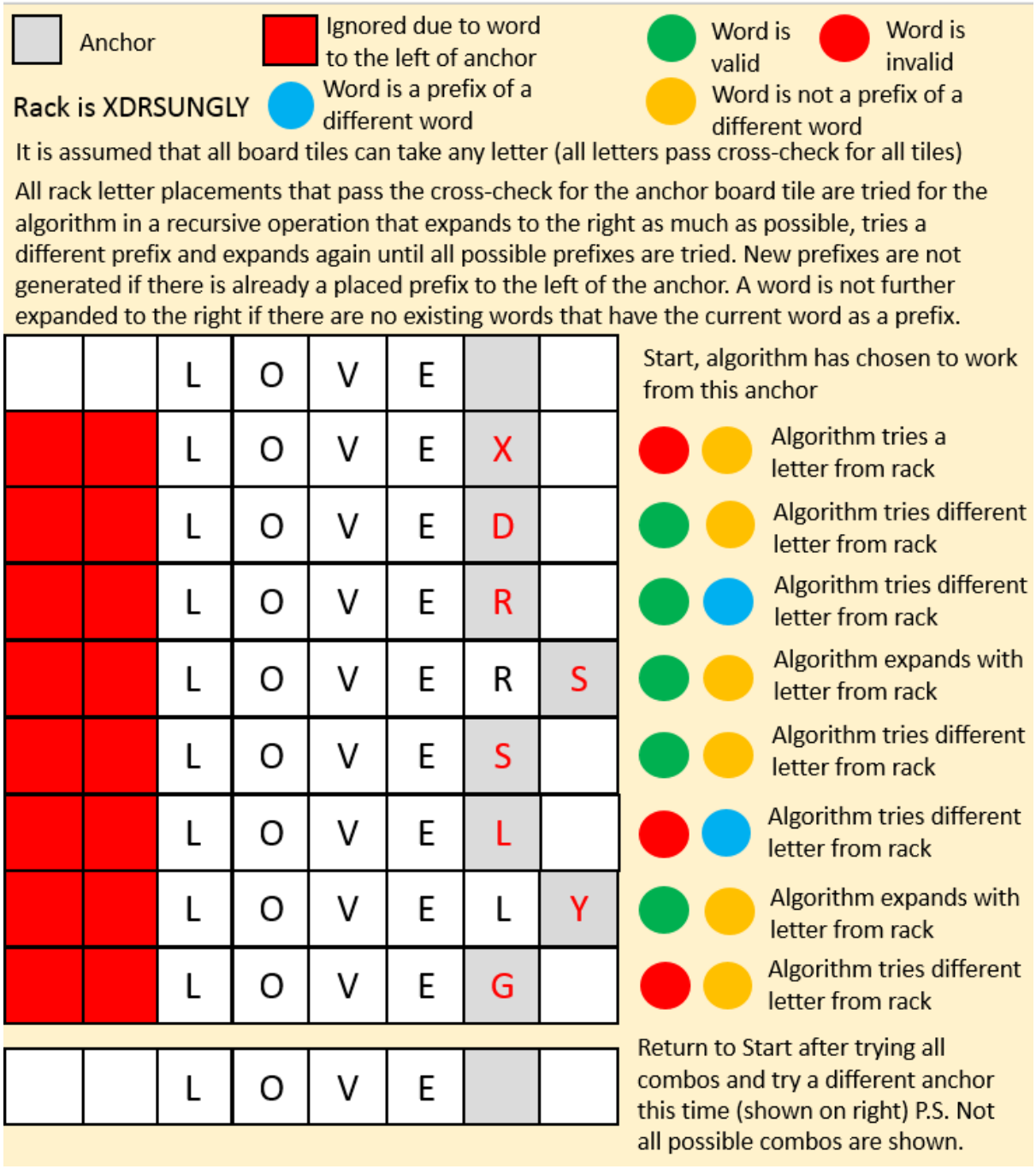


Figure 39. The figure above demonstrates the ExtendRight() method.

5.12.3 LeftPart()

LeftPart() is called when expansion to the left is possible and when a left part has been provided (either from an existing word to the left of the anchor or from a newly placed letter or letters from a previous LeftPart() call). Using the left part that is available, the method calls ExtendRight(). ExtendRight() attempts more ExtendRight() expansions and performs logic to create a collection of valid moves. Eventually, the original LeftPart() call gets control of the application. Using the CAT

example from above, an expansion to the left instead of the right will be performed if the conditions allow it (letter is in rack, it passes the cross-check and if new left part is a confirmed prefix of a different word). This new expansion means that LeftPart() gets called again in a recursive fashion but with different attributes regarding limit of expansion and word created up until this moment). This call again will result in recursive ExtendRight() calls. Eventually, the original LeftPart() call will be completed, a list of valid moves will have been populated and it will be returned and available for the system to use for play making.

The original LeftPart() call might reach its finish sooner given the time limit that was given. If a time limit is reached, LeftPart() will break out of the loop that tries expansions with each letter in the rack thus a limited result of moves will be returned.

The following figure demonstrates the LeftPart() method (the right half of the original figure).

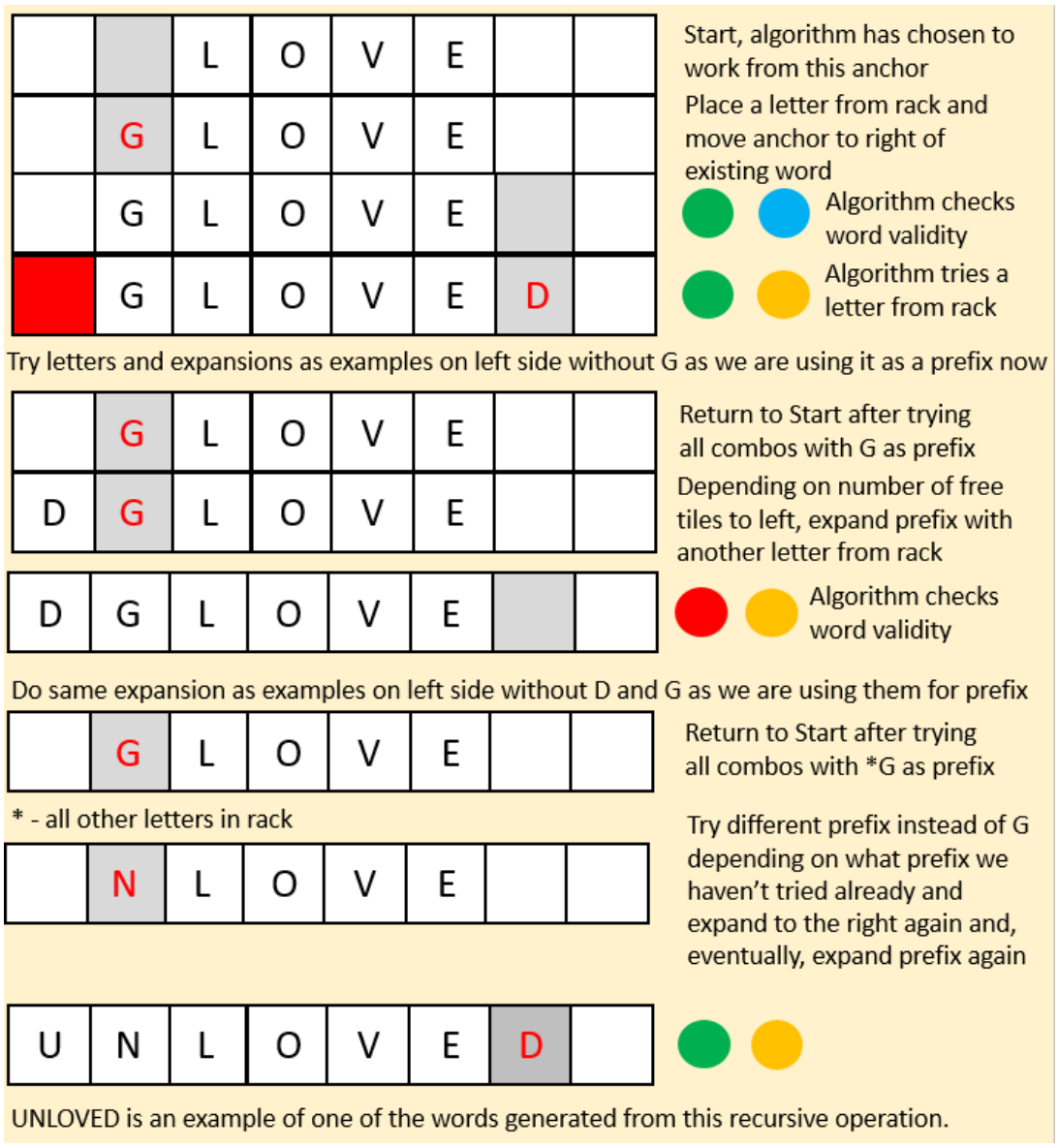


Figure 40. The figure above demonstrates the LeftPart() method which in itself calls ExtendRight() multiple times.

The next section discuss the object type GeneratedMove.

5.12.4 GeneratedMove

A GeneratedMove object is a valid move that can be played on a board. It contains information about the start and end of the play, about the rack tiles used and the board tiles they cover, the word played and other information. Important properties of it are Score, RackScore and ExtraWordsPlayed.

Score is the score that a valid move will earn the player. It is calculated by simulating a play of the move and calculating the score it bring which takes premium tile bonuses into mind. In addition, a valid horizontal move might result in one or more valid vertical moves or vice versa and they are marked as ExtraWordsPlayed. Each score of an extra word is added to the total score. In summary, Score is the sum of the points a played word has earned and the points of all other words formed as a result of the played word. High_Scorer bot picks the move with the highest Score from a list which is based on the above metrics.

RackScore uses a formula derived from Steven Gordon's paper A COMPARISON BETWEEN PROBABILISTIC SEARCH AND WEIGHTED HEURISTICS (listed in references). The score is based on the balance of vowels and consonants in a rack after a move has been made. This balance is calculated by assigning a score to each letter in the rack and a negative score for its duplicates in the rack. Afterwards an equation is performed (stated as VCMix in the paper) and the result of the equation assigned to the RackScore. Rack_Balancer bot picks the move with the lowest RackScore from a list which is based on the above metrics. A high RackScore means that the resulting rack would be one that the player would like to get rid of as it is not balanced. A low RackScore means the player would like to keep the rack as it is balanced.

The following figure shows a list of valid moves sorted by the highest score given a board state. Not each high-scoring move based on raw score necessarily results in a balanced rack.

P.S. This heuristic is only meant to be used in the English version of the game as no heuristic has been derived for different languages as of yet or information is not publicly available.

allValidMovesSorted	Count = 16
[0]	{NAPS, Horizontal, Extra words: 1 - URN;, , Start Index 7 to 10 anchored at 14, 11, Score: 27 points, Rack Score: 10,5}
[1]	{NAP, Horizontal, Extra words: 1 - URN;, , Start Index 7 to 9 anchored at 14, 10, Score: 24 points, Rack Score: 15}
[2]	{PAN, Vertical, Extra words: 1 - PLUCKY;, , Start Index 12 to 14 anchored at 5, 14, Score: 24 points, Rack Score: 15}
[3]	{PAR, Vertical, Extra words: 1 - PLUCKY;, , Start Index 12 to 14 anchored at 5, 14, Score: 24 points, Rack Score: 14}
[4]	{PAS, Vertical, Extra words: 1 - PLUCKY;, , Start Index 12 to 14 anchored at 5, 14, Score: 24 points, Rack Score: 12}
[5]	{POA, Vertical, Extra words: 1 - PLUCKY;, , Start Index 12 to 14 anchored at 5, 14, Score: 24 points, Rack Score: 17}
[6]	{POS, Vertical, Extra words: 1 - PLUCKY;, , Start Index 12 to 14 anchored at 5, 14, Score: 24 points, Rack Score: 14,5}
[7]	{PRO, Vertical, Extra words: 1 - PLUCKY;, , Start Index 12 to 14 anchored at 5, 14, Score: 24 points, Rack Score: 16,5}
[8]	{PA, Vertical, Extra words: 1 - PLUCKY;, , Start Index 12 to 13 anchored at 5, 14, Score: 23 points, Rack Score: 16,5}
[9]	{PO, Vertical, Extra words: 1 - PLUCKY;, , Start Index 12 to 13 anchored at 5, 14, Score: 23 points, Rack Score: 19}
[10]	{NAOS, Horizontal, Extra words: 1 - URN;, , Start Index 7 to 10 anchored at 14, 11, Score: 21 points, Rack Score: 10,5}
[11]	{NAS, Horizontal, Extra words: 1 - URN;, , Start Index 7 to 9 anchored at 14, 10, Score: 18 points, Rack Score: 10}
[12]	{NOR, Horizontal, Extra words: 1 - URN;, , Start Index 7 to 9 anchored at 14, 10, Score: 18 points, Rack Score: 14,5}
[13]	{NOS, Horizontal, Extra words: 1 - URN;, , Start Index 7 to 9 anchored at 14, 10, Score: 18 points, Rack Score: 12,5}

Figure 41. A list of valid moves sorted by the highest score given a board state.

5.12.5 DAWG MatchPrefix() customization

Using MatchPrefix() with a given word will return potentially thousands of available words that have the given word as a prefix. To filter the results down to the letters that can follow after the prefix, a

HashSet is implemented with a custom comparer called DawgEdgeEqualityComparer which states the conditions for uniqueness and duplication for elements (words) in the HashSet. The condition to be met for an element to be added to the HashSet is that the first letter of that element must not already exist as a first letter in any of the elements already in the HashSet. Each element that gets added has the prefix (given word) stripped away. The result is a collection of words that may follow the prefix and each word has a different starting letter. As there are no elements that start with the same starting letter, the result size cannot be greater than the count of letters in the alphabet.

If we look at a given word as a node in the DAWG, the first letter of each element represents a path to a different node labelled by that letter. By having this information, letters from the rack that are not valid continuations of any word are not attempted for expansion, reducing workload.

The following figure shows a list of 13 words with different starting letters words that have the prefix D. This list tells the algorithm that there are 13 possible letters to place after D that will result in a valid move eventually. Placing the other 13 letters from the alphabet that are not in this list would never lead to a valid move and thus are never attempted for expansion. All words (DA, DE, DHAK, DI, DJEBEL etc.) are in the dictionary used by the project.

labelsOfDawgEdges	Count = 13
[0]	"A"
[1]	"E"
[2]	"HAK"
[3]	"I"
[4]	"JEBEL"
[5]	"O"
[6]	"RAB"
[7]	"SO"
[8]	"UAD"
[9]	"VANDVA"
[10]	"WAAL"
[11]	"YABLE"
[12]	"ZEREN"

Figure 42. A list of 13 with different starting letters words that have the prefix D.

5.12.6 Computer play

By the end of these methods, the system will have a list of moves that can be played given the current rack and game state. If a computer player is at turn, these methods will be executed to get the list and the computer will pick a move based on the computer player type and the attributes listed the GeneratedMove section. Once it makes a move, the game passes on to the next player. If no valid move can be made, the computer will change all the tiles in the rack with new ones from the pouch. If the pouch is empty, the computer will skip its turn. Both redrawing and skipping give control to the next player.

5.12.7 What about blank tiles in the rack?

If a blank tile is in the rack and the algorithm picks it up to expand with, the tile will take the form of all letters in the alphabet and each form will be used for an attempt in expansion, increasing the number of moves and time taken for move generation greatly. Due to the massive increase in resource usage, blank tiles have been disabled for the demo and report. To enable them, change line

650 in the ScrabbleContext file. Change Count = 0 to Count = 2 and rebuild the migration file. Or you can access the database by using Visual Studio's SQL Server Object Explorer and adding an entry in the Pouch_CharTiles table to add the blank tiles to the pouch of a game (a blank tile has a CharTileID of 1).

5.13 What happens next?

If the game has been passed to a human player, a user will have (unlocked) access to the web page and will be able to play a word using the web page given the rack that is available to the current player object. If a game is passed to a computer player, the page will become locked, the server will use a MoveGenerator object to get a list of moves, a move will be played on the board, the database will be updated and the game will pass on to the next player who again might be a human or computer.

The game can be terminated or finished under the following conditions: either if one of the players is left with an empty rack after attempting to refill it from the pouch or if any of the human players click the "End Game" button on the web page. Rewards and penalizations to players' scores will be performed based on the players' remaining tiles in the rack.

A game can be reset at any time by pressing the "Reset Game" button on the web page. This resets the players' scores, rack, game pouch and board and lets you start the game anew.

Figure 43. An example of a finished game as the pouch is empty and Dobromir has played all the tiles in his rack.

5.14 Non-English Scrabble

To play a game in non-English languages, the following steps are required:

- Add the new language in the GameLanguages table.
- Add a new entry in WordDictionaries table for the new language.
- Populate the CharTiles table with the letters from the new language. A blank tile (*) for that language should also be added along with duplicate tiles of the letters with a score of 0. All CharTiles should use the WordDictionaryID of the new language.
- Add a new entry in the Games table that uses the WordDctionary of the new language.
- Add entries for the other objects of the game (board, board tiles, players, racks, pouch). Make sure these entries are related to the new game.
- Add Pouch_CharTile entries. Make sure they are related to the new pouch and that the correct CharTileIDs and counts are used. Rack_CharTiles are automatically populated at the start of a game.
- Load the new game from the browser. At the moment, this requires all calls to a specific GameID in the controller to be changed to call the ID of the new game.

A dawg object from the DawgSharp library works with Unicode characters and using it as a dictionary for a foreign language is not a problem. Find a dictionary of the language as a text file, convert it to a bin file (check DawgSharp's GitHub page for details) and load the bin file using the dawg object – just as how the process was done using the English dictionary.

The following is a game in progress using the Bulgarian language and dictionary.

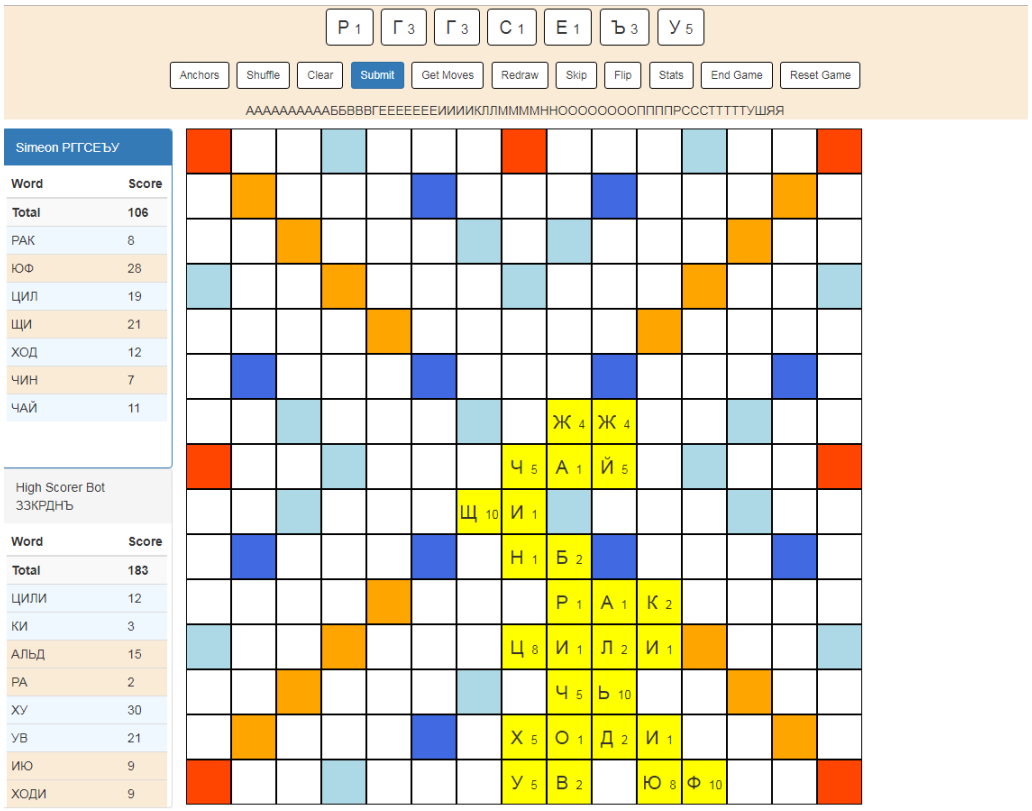


Figure 44. An example of a Bulgarian game. All words in the player logs exist in the Bulgarian dictionary (bulgarianWords.txt file in Helpers folder).

There are some things to note in the above game. One is that Rack Balancer Bot is not used in the above game as there is no heuristic available from existing research that can assign a score to the vowel/consonant balance of a Bulgarian rack. In addition, while the words played in the game are all correct according to the dictionary, the dictionary itself is not an official Scrabble dictionary and contains many nonsensical words. Finding a better one proved to be a struggle. Nonetheless, the application still works correctly given the data and dictionary that it is working with. The steps above can be replicated to add additional languages as long as a dictionary file is available for it.

5.15 End of Implementation Section

By this stage of the report, the development of computer playing, a normal game between a human and a computer and its implementation in different languages have been covered. The following sections will focus on testing which will aim to test all features of the game.

6. Testing

This section will test the implementation and functionality of all components of the game. Examples of finished games will be provided and features of the game will be tested individually.

The implementation of techniques and algorithms has already been extensively debugged and tested during the implementation stage of the report. Thus the content here will provide confirmation that the aforementioned elements work correctly and will test elements not thoroughly covered in the implementation section of the report such as the functionalities of the buttons of the front-end.

The following were the results of three different games in English.

Figure 45. Game 1 results. Each word in each player log (LIN, OI, LI etc.) has been confirmed as available in the dictionary.

Figure 46. Game 2 results. Each word in each player log (TIC, GOV, DIET etc.) has been confirmed as available in the dictionary.

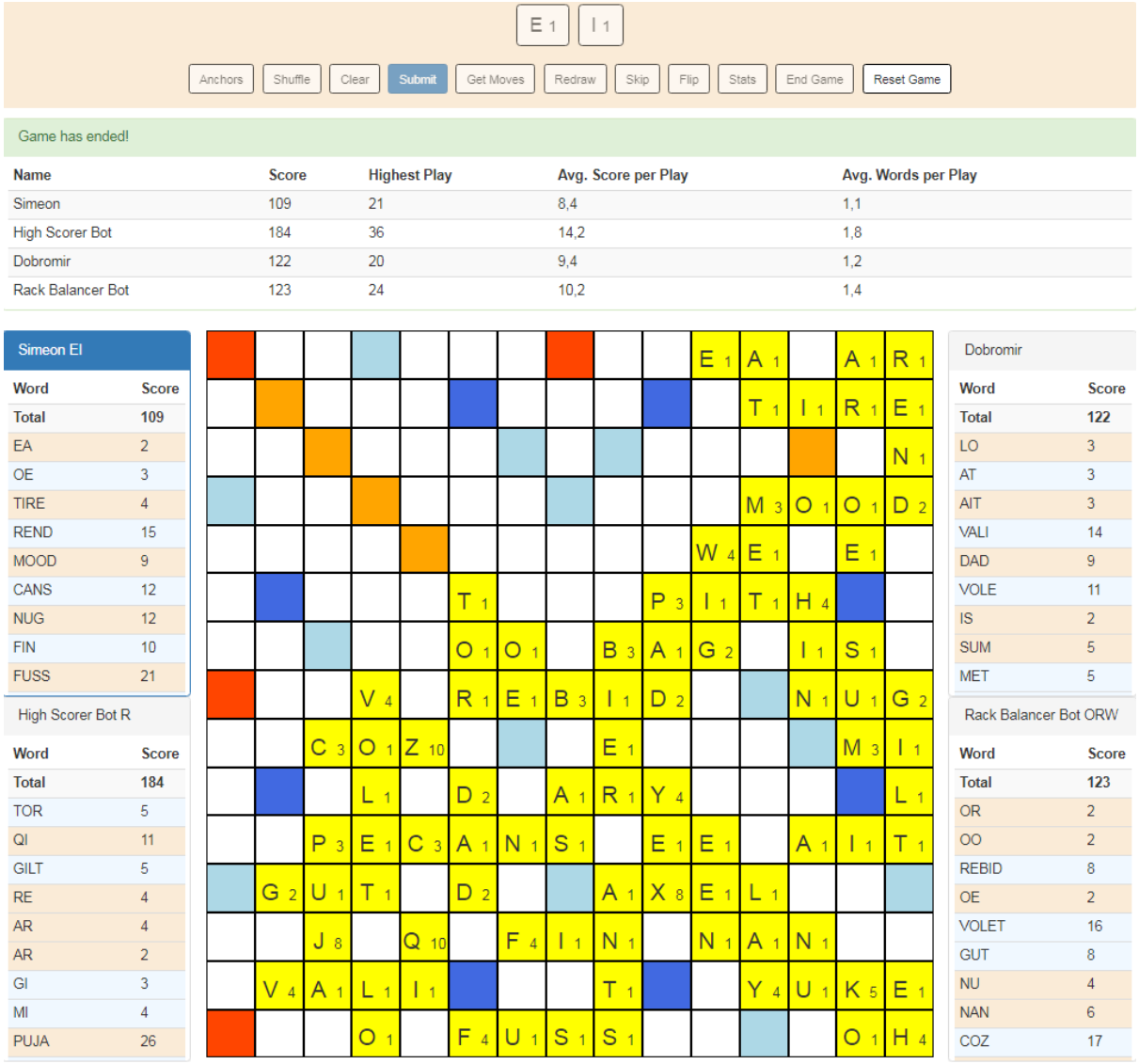


Figure 47. Game 3 results. Each word in each player log (EA, OE, TIRE etc.) has been confirmed as available in the dictionary.

A complete game in Bulgarian is also demonstrated.

Figure 48. Game 4 result (Bulgarian). Each word in each player log (ОМИЛ, МАК, СЕРАТ etc.) has been confirmed as available in the dictionary.

The following features have been successfully tested.

6.1 Client-side

- As a human player and at the start of the game, playing a word that is not using the start board tile results in an “Invalid starting move” message from the client.

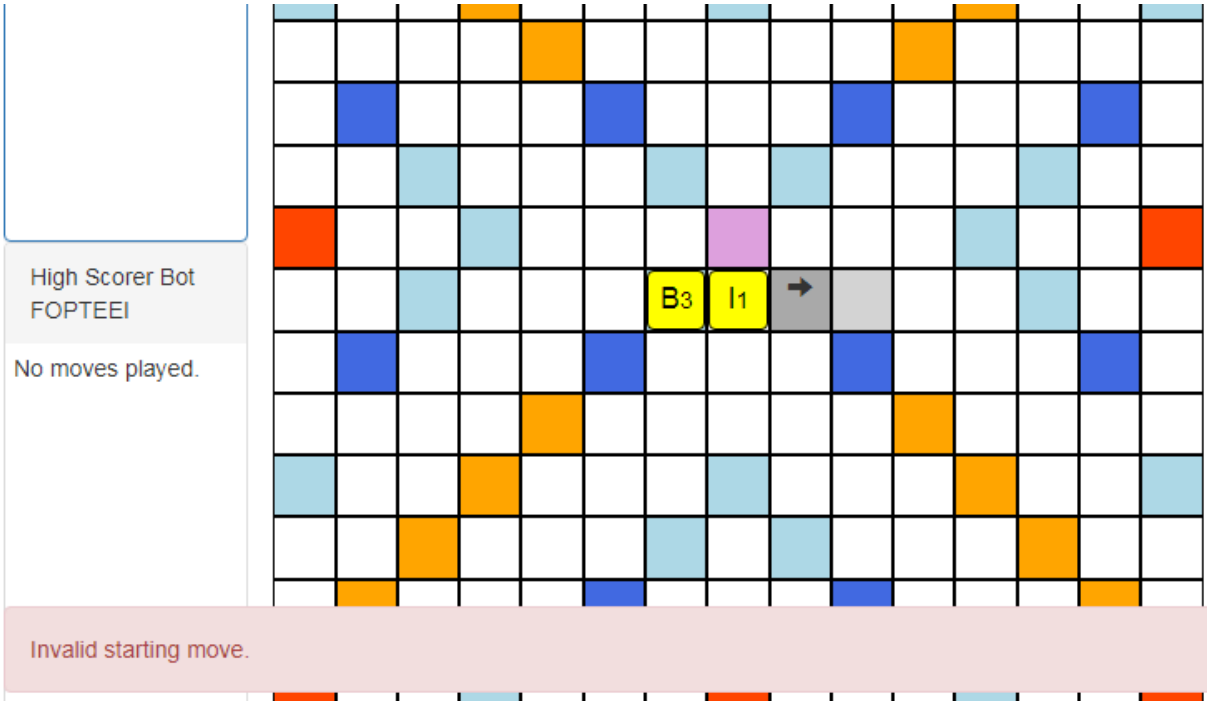


Figure 49. Testing client-side validation of start tile usage (start tile is pink).

- As a human player, placing letters that are not connected results in a “Play not connected” error from the client.

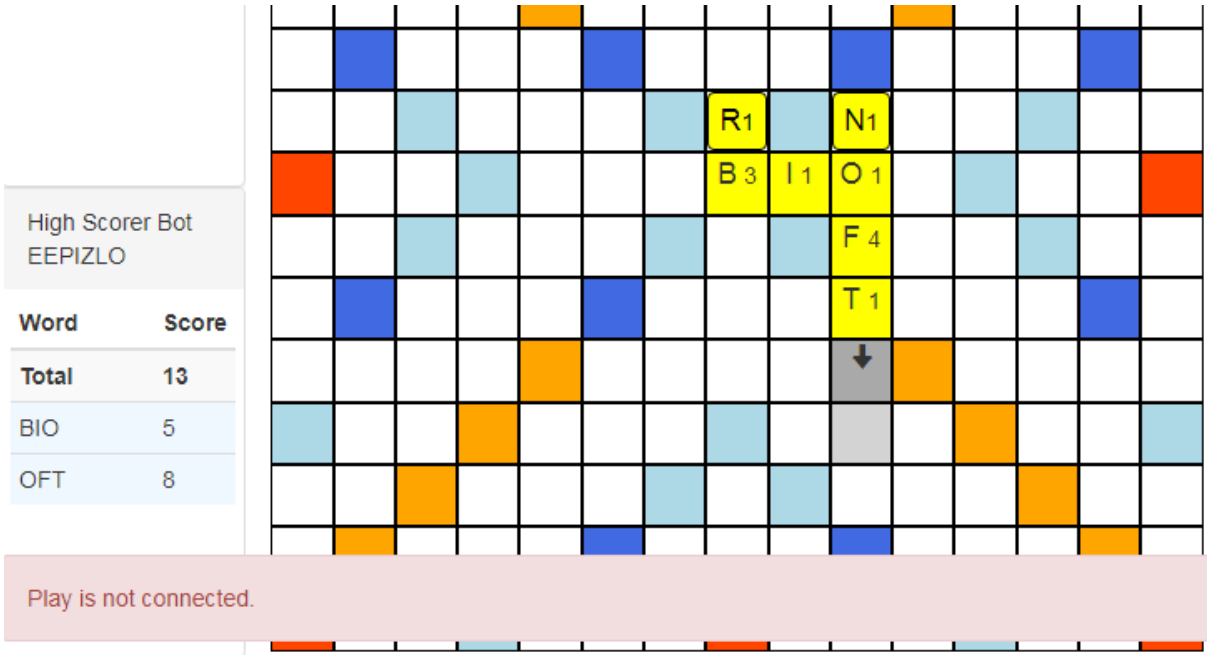


Figure 50. Testing client-side validation of play connection (R and N have been placed).

- As a human player, placing letters that are both horizontally and vertically connected to each other results in a “Please use only one row or column” error from the client.

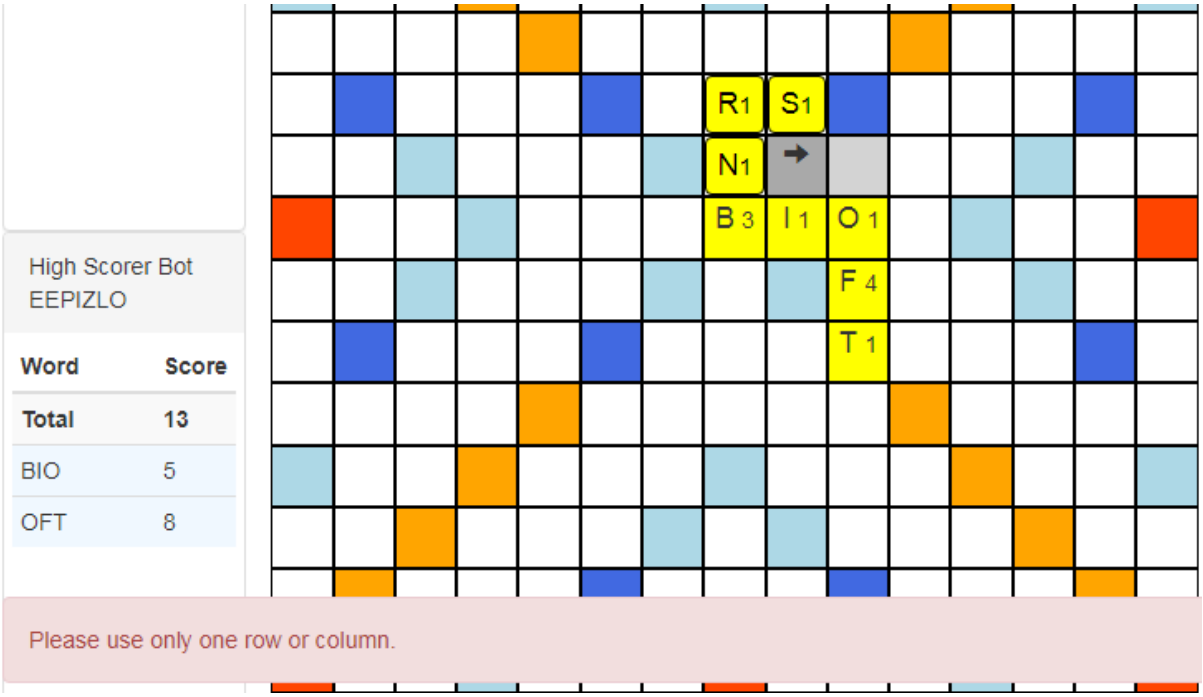


Figure 51. Testing client-side validation of same row/column play (R, N and S have been placed).

- As a human player, making a move that is not connected to an existing word on the board results in an “Anchor not used” error from the client.

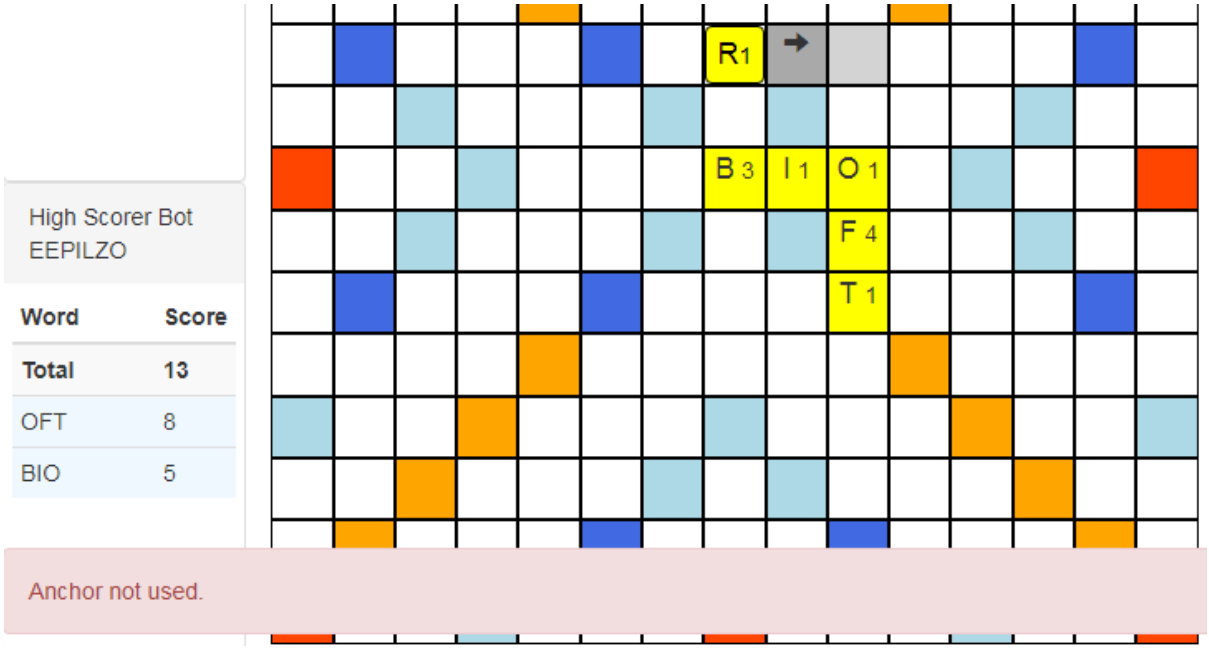


Figure 52. Testing client-side validation of anchor usage (R has been placed).

- Attempting to place a letter without having a tile selected results in a “Please select a tile” error.

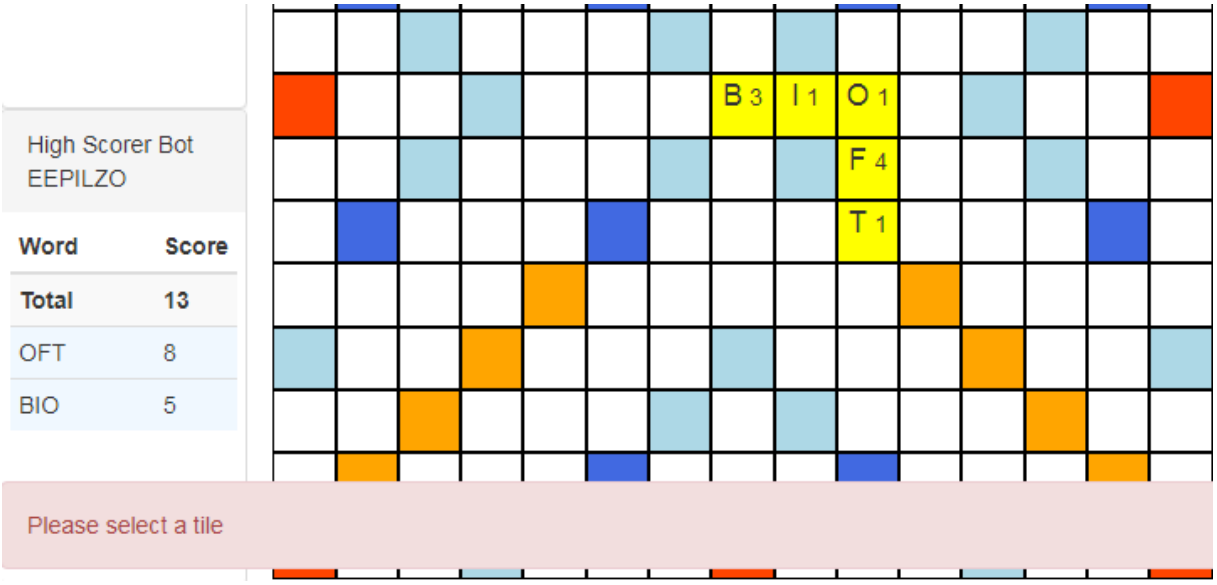


Figure 53. Testing client-side validation of existence of active tile.

- Clicking on a board tile makes it active and paints it in grey with an arrow on top.
- Pressing Enter perform a click on the Submit button.
- Pressing Backspace performs a click on the Clear button.
- Pressing Space changes the direction of the arrow on the active tile.
- Typing letters available in the rack places the letter on the active tile and also marks the letter in the tile as used.
- The active tile is changed depending on the arrow location when a letter is placed.
- When the active tile is changed from placing letters, the next active tile is the closest one that is not taken. If not, the active tile does not change.
- The board, rack, controls and player logs are properly resized and reordered when changing the size of the window.

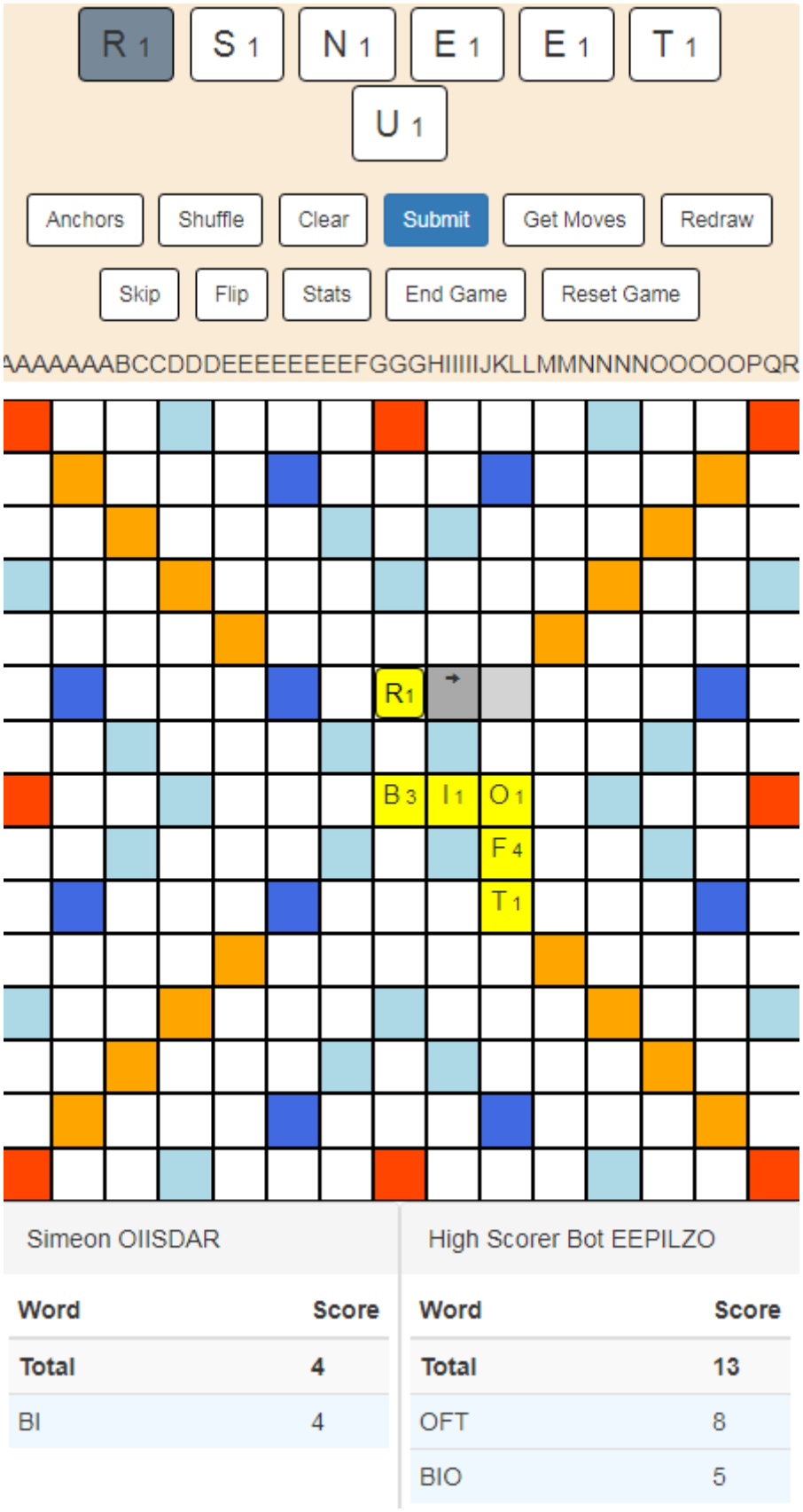


Figure 54. Testing page responsiveness.

- The height of the player logs matches the height of the board.
- Hovering over a word from the player log marks it in green on the board.

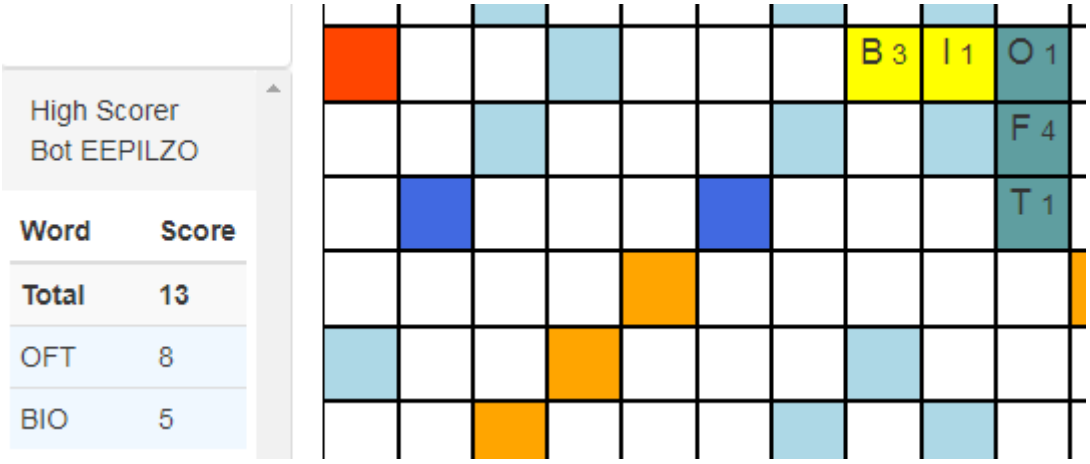


Figure 55. Testing word marking (OFT from the player log was hovered over).

- The active tile can be changed either by clicking a different tile or by using the arrow keys on the keyboard.
- Board tiles that already have a letter cannot be marked as active.
- Clicking on Anchors marks all the anchors on the board as green.

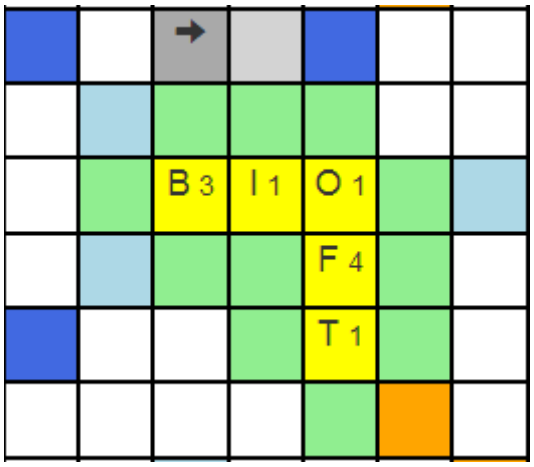


Figure 56. Testing anchor marking.

- Clicking on Shuffle shuffles the rack of the player.

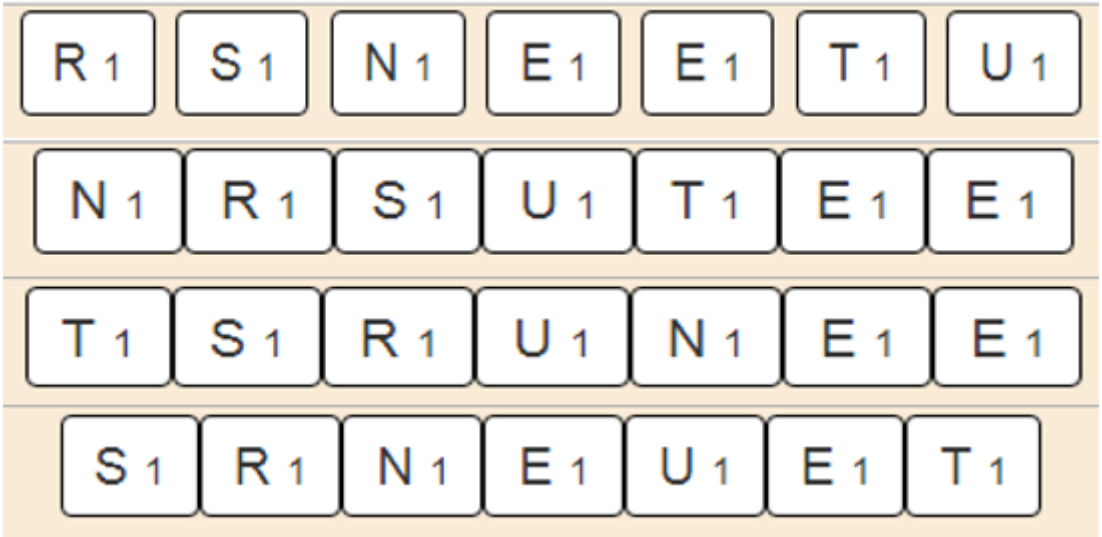


Figure 57. Testing functionality of Shuffle button.

- Clicking on Clear returns all placed rack letters from the board to the rack.
- Clicking on Submit performs client-side validation of the played move and sends data to the server to compute scores and handle the game to the next player.
- Clicking on Get Moves gets a list of possible moves from the server and puts a table behind the Valid Moves button (available only when Get Moves has been used).

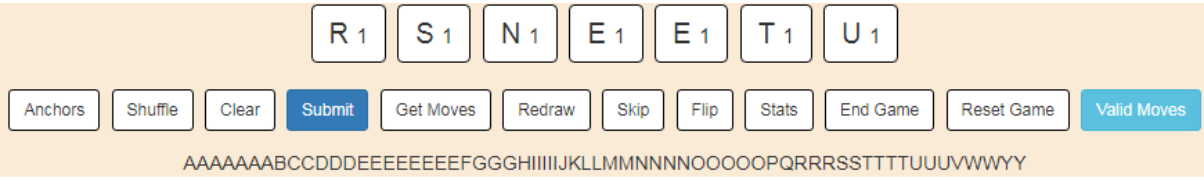


Figure 58. Testing appearance of Valid Moves button.

- Clicking Redraw opens a prompt prompting the player for the tiles they would like to redraw.

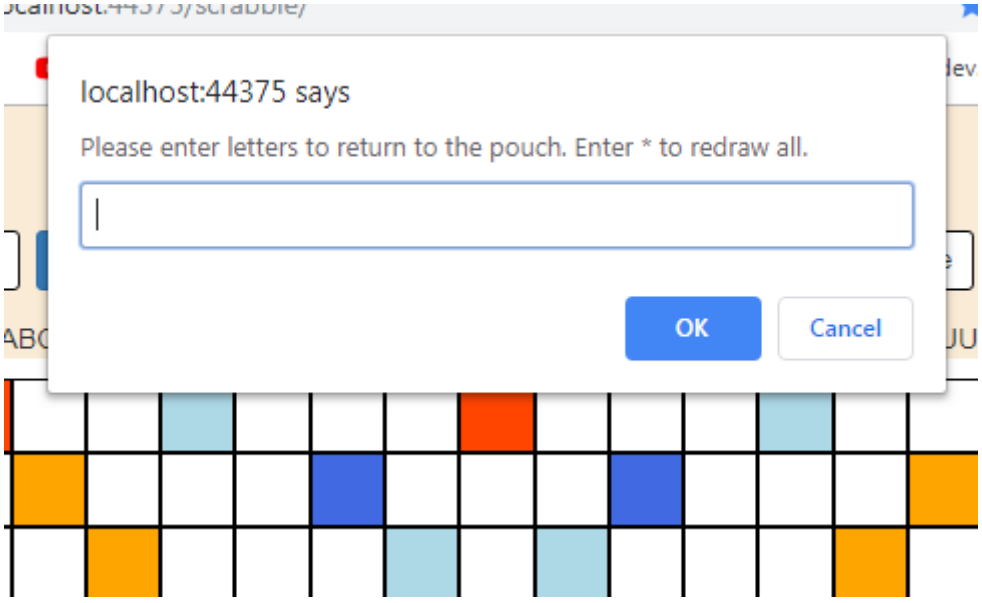


Figure 59. Testing appearance of Redraw prompt.

- Using * for the redraw prompt redraws all tiles from the rack.
- Putting in more letters than there are tiles in the rack with that letter results in an error.

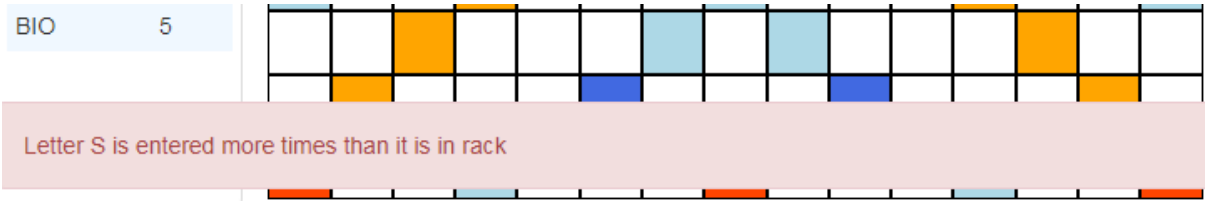


Figure 60. Testing error handling of Redraw prompt (S was entered twice but only available once in the rack).

- Putting in unavailable letters or characters in the Redraw prompt results in an error.
- Putting in a string with a length greater than the count of the rack shortens it to match the count of the rack.
- As well as changing letters, Redraw handles the game to the next player.
- Clicking on Skip handles the game to the next player.
- Clicking on Flip returns the opposite version of the board (untransposed or transposed).

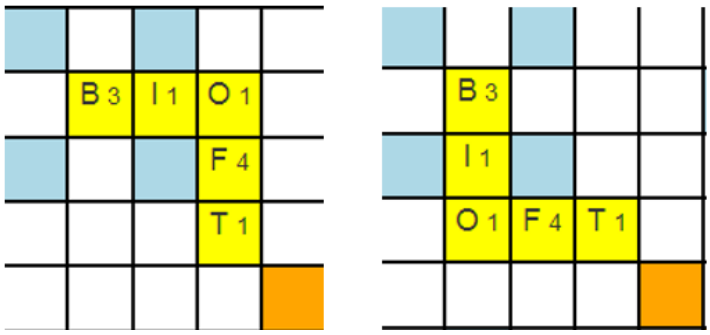


Figure 61. Testing board transposition. Bigger example is available in Implementation section.

- Clicking on Stats shows or hides a table with statistics about the game.

Stats

Name	Score	Highest Play	Avg. Score per Play	Avg. Words per Play
Simeon	4	4	4	1
High Scorer Bot	13	13	13	2
Dobromir	0	0	0	0
Rack Balancer Bot	0	0	0	0

Figure 62. Testing Stats functionality.

- Clicking on End Game ends the game i.e. the controls and board gets locked and the stats are shown.
- Clicking on Reset Game clears the board and player logs and resets the pouch and the player racks.
- Clicking on Valid Moves shows a sortable table with a list of possible moves.

Valid Moves

Show 10 entries Search:

Word	Direction	Extra Words	Row/Column Start	Row/Column End	Row/Column Index	Score
AFREET	Vertical	0 -	6	11	7	10
AFT	Vertical	0 -	6	8	7	6
AFTER	Vertical	0 -	6	10	7	8
ARF	Vertical	0 -	5	7	7	6
AUF	Vertical	0 -	5	7	7	6
CAB	Horizontal	0 -	9	11	9	7
CABER	Horizontal	0 -	9	13	9	11
CABRE	Horizontal	0 -	9	13	9	11
CAR	Horizontal	0 -	9	11	9	5
CARB	Horizontal	0 -	9	12	9	8

Showing 1 to 10 of 47 entries Previous 1 2 3 4 5 Next

Close

Figure 63. Testing Valid Moves functionality.

- Clicking on a move from the table closes it and marks the move location on the board.

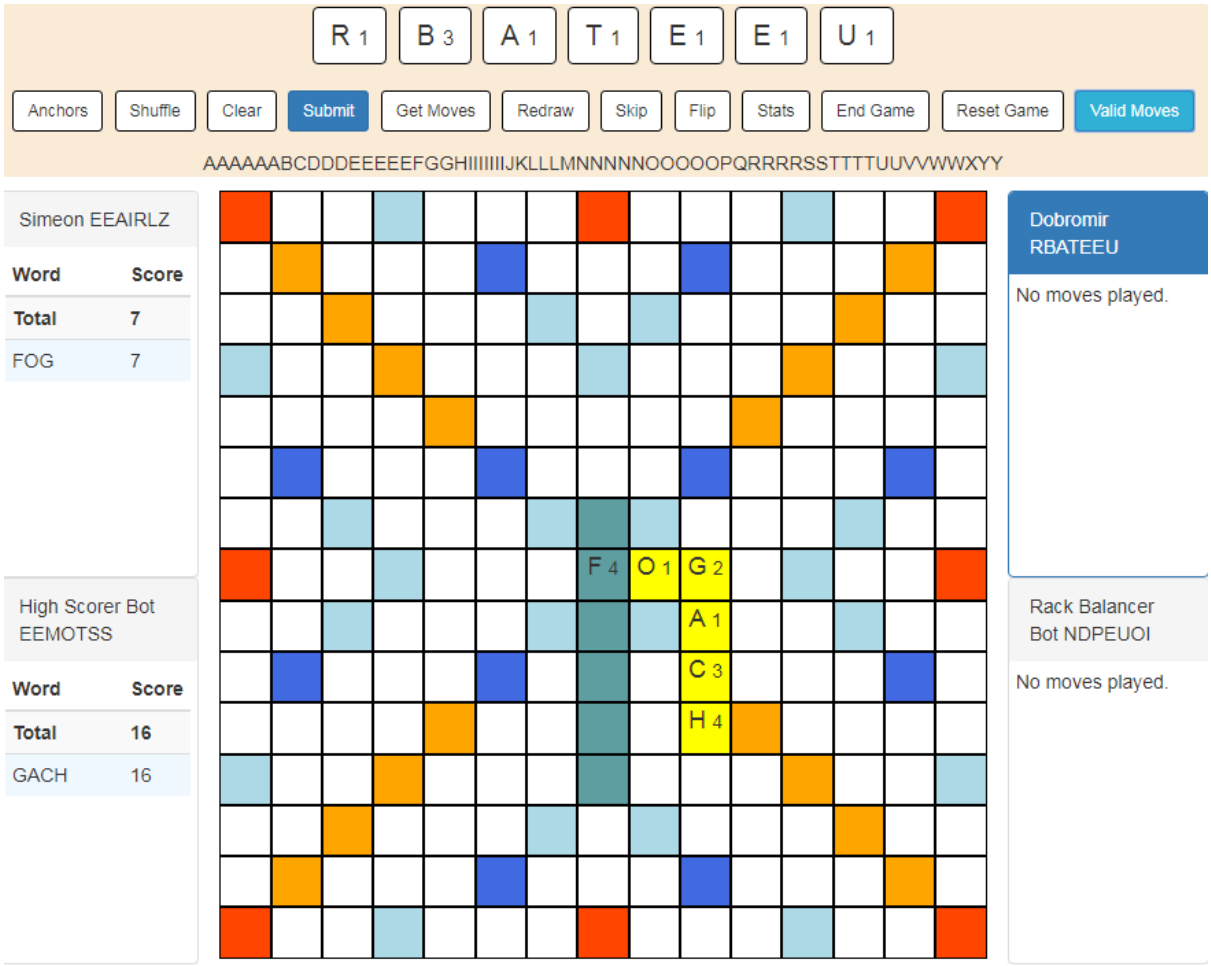


Figure 64. Testing Valid Moves move selection functionality. The word AFREET was picked from the table.

- The racks displayed in the sections of the players are correct.
- The contents of the pouch displayed above the board are correct.
- In the list of moves of each player, words with the same stripe are part of the same move.

Figure 65. Testing correctness of move stripes. In Dobromir's log, OE and EA were played in the same move while RE was the only word from his last move.

6.2 Server-Side

- As a human player, playing an incorrect word results in an error from the server.

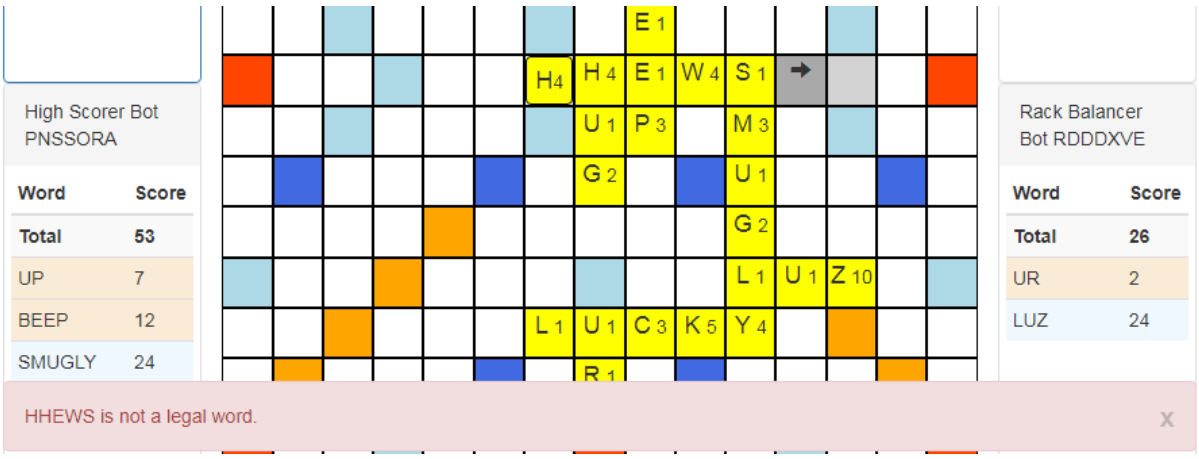


Figure 66. Testing server-side validation of words.

- Every word played by the computer players is contained within the dictionary the game is using.

- Every word played by the human players is contained within the dictionary the game is using.
- Moves made by either player use a valid anchor.
- Moves made by either player do not make illegal moves in regards to the cross-check sets of each board tile.
- The board is correctly updated on a move.
- Playing letters from a rack removes them from the racks of the players.
- Depending on the number of player rack letters, just as many are drawn from the pouch.
- The tiles drawn from the pouch are random and unpredictable.
- Drawing tiles from the pouch causes the letters in the pouch to decrease.
- The board displayed to the user is an accurate representation of the board object that the game object is using.
- Playing words from a rack places the used rack tiles on the board permanently.
- Moves made by human player are retrieved correctly.
- Word scores are calculated correctly.
- Premium tile bonuses are applied correctly.
- Making a move containing multiple words scores every word played correctly.
- Premium tile bonuses are removed once the tile has been used.
- Anchor list is built correctly on move generation.
- Valid cross-check sets are built correctly on move generation.
- Move generation has been extensively debugged and always provides correct words.
- Database is updated correctly on game updates.
- Dawg dictionary is built correctly.
- The computer opponent redraws its rack if it cannot make a valid move and the pouch is not empty.
- The computer opponent skips its turn if it cannot make a valid move and the pouch is empty.
- The game is ended if any of the racks in the game are empty.
- Ended games penalize the scores of player whose rack is not empty and rewards the rack of the player that managed to play the entirety of his rack.

6.3 End of Testing Section

This section has tested most of the functionality that happens on the client-side and server-side of the application. All tests being successful has resulted in the development of a solid Scrabble playing application for human and computer players.

The next section will compare requirements met of the finished project with the originally planned requirements and it will cover some of the flaws of the project and what can be done to improve it.

7. Evaluation

The project has been evaluated by myself and flatmate Dobromir with whom we've played games together. Functionality wise, the game makes no mistakes and works correctly.

In regards to the original requirements, all requirements of priority MUST and SHOULD have been implemented and most requirements of priority COULD have been implemented as well, the only ones that haven't been implemented are for the AI to perform simulations in order to pick better moves and for the AI to change its strategy during various phases of the game. The issues around their development are connected to the biggest flaw of the project and main topic of future work.

In regards to the original aim and question of the project - can the game be re-implemented with new technologies given current research, implementations and techniques available and can it be improved using the available information – it is a success. C# and .NET are modern tools at the moment and their usage have allowed the game to be re-implemented with features such as multiple players (human and computer) and multiple languages as a responsive single-page web application.

7.1 Future work

7.1.1 Issues caused by technologies and language used

While move generation works, it is slow. Without a time limiter in place, a game that is halfway done can take the server up to 10 minutes to select the best move. A&J have stated the algorithm takes seconds to identify all possible moves on the VAX 11-780 they have used. The language and technology used is not mentioned in their paper but it is my personal assumption that a much lower-level language was used than C# in a less resource-intensive environment (unlike in a server used for hosting a web application). While I believe this would be a key factor in optimising the speed, the project logic can do with some improvements. For example, there is an abundance of object creating when generation moves such as recreating the original state of the board if a valid move is detected. The amount of object generation could be reduced to improve the speed of the algorithm which might involve changing the architecture of the software.

In the future, better research will be performed in regards to the performance of various technologies and languages used for resource-intensive operations such as move generation (which uses many recursive calls and generates many objects).

7.1.2 Issues related to optimisation techniques

The longest operations are related to word generating and validity checking. One improvement would be optimising of the generation of anchors and cross-check sets. At the moment in order to get these, every board tile on the board is scanned and checked if it's a part of a vertical word or an anchor. The workload of this check takes around 10 seconds and can be reduced by reducing the number of scanned tiles to just the ones that have been affected by a recent move as nothing about the other board tiles will have changed. Another improvement would be to store the cross-check sets in the BoardTile objects to avoid repeating the validation of some of the words they are part of. Both optimisations were attempted but were unsuccessful for various reasons and due to the need to focus on other parts of the project, they have been saved for future work.

Another optimisation, which comes at a cost, is to use a GADDAG instead of a DAWG for the dictionary as it can be around two times faster than a DAWG when checking for word validity. As hundreds or thousands of words may be checked during move generation if they are valid, halving the time for this check may drastically improve speed results. But the memory usage for the dictionary is 5 times larger although that may be a non-factor – it is worth attempting in future work.

7.1.3 Issues related to simulations

The issues mentioned above in regards to speed are what have made the implementation of move simulation fruitless for the moment. Move simulation requires having a full list of valid moves, picking a move, predicting any move from the opponent to respond to that move and then generating a new list of valid moves again and picking a move to respond to the opponent's move. This process is to be repeated for a given number of possible moves that can be made by the opponent. By building up a list of scenarios, the computer can pick a move that would lead to the best scenario rather than pick a move that is the best for the current state of the board. But such intelligent play requires multiple calls to the move generation algorithm and that would take an extensive amount of time. The amount of time would need to be reduced by performing the optimisations listed in the previous sections and possibly more to even think about simulations.

7.1.4 Other improvements

Personal wishes for improvements would be to implement a front-end framework such as AngularJS to the validation of a client easier. At the moment, many JavaScript functions are used which have led to a somewhat big and disorganised JS file. Angular would allow to create the objects used in the JS file through HTML notation and the logic the Angular provides should result in a more manageable code base. It is unfamiliar but trendy new technology which I would like to work with in the future.

Overall, experimentation with new technologies would be a good focus for future work.

7.2 Legal, social, ethical and professional issues

As this project was made for personal use, through the use of publicly available packages and libraries and is not intended to be released for commercial purposes, no such issues have been created.

7.3 End of Evaluation Section

This section has compared the finished project to the originally planned project and discussed some flaws in it and ways of optimising it.

The next section is the conclusion which will share some final words and the following sections will be References and Appendices.

8. Conclusion

While flawed, the project has been an overall success as all of mandatory and should-have requirements have been implemented along with most of the could-have requirements.

Each chapter played a key role in the writing of this report. Abstract and Introduction presented to the reader a summary of the project, the goals of the project and the intention behind all of the following sections. Literature Review introduced the game's history, algorithms and implementations used in the past. Design and Methodology introduced the technologies used in the implementation. Implementation demonstrated how the technologies were developed and how they worked together. Testing tested all of the features of the complete implementation to prove it as fully working. Evaluation reflected on the project in regards to what requirements it met, discussed current issues with it and how they would be handled in future work as well as possible features that can be added.

Building a web application such as this from scratch proved difficult at times but the final result, even with its current flaws, is still a great success and it has left me much more confident for future roles in web development as I have managed to build a fairly solid web application from scratch using modern technologies.

Acknowledgements are given to my supervisor David Lonie without whom the project would not have been as functional and presentable.

9. References

The literature review contains all the published papers referenced in the report. For the development of the project only guides and tutorials were used. The following links provided the most help with the implementation of the project and have been used to an extent that is too great to put the entirety of in this report.

https://docs.microsoft.com/ - documentation form Microsoft which greatly helped in using their technologies

https://stackoverflow.com/ - used for a variety coding related questions

Journals, papers and useful links have been put below:

1. Janakshi Dulanga D., *An Artificial Intelligence Based Scrabble Game for Sinhala Language*, Proceedings of APIIT Business, Law & Technology Conference, 2017 Colombo, Sri Lanka. ISBN978-955-7678-02-3
2. APPEL, A. and JACOBSON, G. (1988). The World's Fastest Scrabble Program. *Communications of the ACM*, 31(5).
3. Sheppard, B. (2002). *World-championship-caliber Scrabble*☆☆SCRABBLE® is a registered trademark. All intellectual property rights in and to the game are owned in the USA by Hasbro Inc., in Canada by Hasbro Canada Corporation, and throughout the rest of the world by J.W. Spear & Sons Limited of Maidenhead, Berkshire, England, a subsidiary of Mattel Inc. *Artificial Intelligence*, 134(1-2), pp.241-275.
4. Rajkumar, P. (n.d.). *A Survey of Monte-Carlo Techniques in Games*. Master's Scholarly Paper.
5. Di Maria, F. and Strade, A. (n.d.). *An Artificial Intelligence that plays for competitive Scrabble*. Master Degree in Computer Engineering. Alma Mater Studiorum, Bologna.
6. Richards, M. and Amir, E. (n.d.). *Opponent Modeling in Scrabble*. Computer Science Department. University of Illinois at Urbana-Champaign.
7. Gordon, S. (1993). *A COMPARISON BETWEEN PROBABILISTIC SEARCH AND WEIGHTED HEURISTICS IN A GAME WITH INCOMPLETE INFORMATION*. Department of Mathematics. East Carolina University.
8. Connolly, F. and Gren, D. (2012). *Smart Scrabble playing - strategies and their impact*. DD143X. Kungliga Tekniska Högskolan.
9. Gordon, S. (1994). *A faster scrabble move generation algorithm*. Software: Practice and Experience, 24(2), pp.219-232.
10. Abraham P.J., *A Scrabble Artificial Intelligence Game*, MSc Thesis, San Jose State University (2017)
11. Shapiro, S. (n.d.). *A SCRABBLE CROSSWORD GAME PLAYING PROGRAM*. Department of Computer Science. State University of New York at Buffalo Amherst, New York 14226.
12. Shah, A., Ansari, A. and Ismaili, I. (2005). Speech Enabled Algorithm for Multilingual Educational Game. *IEEE --- 2005 International Conference on Emerging Technologies*.
13. Gunkies (n.d.). *VAX-11/780 - Computer History Wiki*. [online] Available at: http://gunkies.org/wiki/VAX-11/780 [Accessed 26 Oct. 2018].

14. Fatsis, S. (2014). *Slate's Use of Your Data*. [online] Slate Magazine. Available at: https://slate.com/human-interest/2014/09/scrabble-copyright-dispute-hasbro-says-it-owns-the-scrabble-dictionary-players-beg-to-differ.html [Accessed 7 Oct. 2018].
15. ABC News. (n.d.). *Scrabble Offensive Launched on Facebook regarding Scrabulous*. [online] Available at: https://abcnews.go.com/Technology/story?id=4603362&page=1 [Accessed 7 Nov. 2018].
16. Eldon, E. (2009). *F-A-I-L: Official Scrabble Facebook apps still smaller than Scrabulous was*. [online] VentureBeat. Available at: https://venturebeat.com/2009/01/06/f-a-i-l-official-scrabble-facebook-apps-still-smaller-than-scrabulous-was/ [Accessed 7 Nov. 2018].
17. Ybarra, B. (2013). *Defeating mobile game clones: Why copyright protection is not enough*. [online] VentureBeat. Available at: https://venturebeat.com/2013/03/16/defeating-mobile-game-clones-why-copyright-protection-is-not-enough/ [Accessed 8 Nov. 2018].
18. Fahey, R. (2017). *Tackling mobile's "game cloning" issue*. [online] GamesIndustry. Available at: https://www.gamesindustry.biz/articles/2017-07-14-tackling-mobiles-game-cloning-issue [Accessed 8 Nov. 2018].
19. Levy, S. (2012). *How Words With Friends Is Killing Scrabble... and Why It Matters to Lawyers – Slaw*. [online] Slaw. Available at: http://www.slaw.ca/2012/02/27/how-words-with-friends-is-killing-scrabble/ [Accessed 5 Nov. 2018].
20. Alexander, S. (n.d.). *Legal issues related to Scrabble*. [online] Teleport. Available at: http://home.teleport.com/~stevena/scrabble/legal/issues.html [Accessed 5 Nov. 2018].
21. Copyright in Games. (2016). [ebook] 101 Independence Ave SE · Washington,DC20559-6000: Library of Congress · U.S.Copyright Office. Available at: https://www.copyright.gov/fls/fl108.pdf [Accessed 5 Nov. 2018].
22. Hasbro. (n.d.). *Scrabble History | Making of the Classic American Board Game*. [online] Available at: https://scrabble.hasbro.com/en-us/history [Accessed 31 Oct. 2018].
23. Hasbro. (n.d.). *Scrabble Rules | Official Word Game Rules | Board Games*. [online] Available at: https://scrabble.hasbro.com/en-us/rules [Accessed 31 Oct. 2018].
24. Wikipedia. (2014). *GADDAG*. [online] Available at: https://en.wikipedia.org/wiki/GADDAG [Accessed 27 Oct. 2018].
25. Wikipedia. (n.d.). *Scrabble*. [online] Available at: https://en.wikipedia.org/wiki/Scrabble [Accessed 27 Oct. 2018].
26. Srivastava, B. (2017). *History Of C# Programming Language*. [online] C-sharpcorner. Available at: https://www.c-sharpcorner.com/blogs/history-of-c-sharp-programming-language [Accessed 1 Apr. 2019].
27. Purdy, K. (2011). *What Is the .NET Framework, and Why Do I Need It?*. [online] Lifehacker. Available at: https://lifehacker.com/what-is-the-net-framework-and-why-do-i-need-it-5791578 [Accessed 1 Apr. 2019].
28. Roth, D., Anderson, R. and Luttin, S. (2019). *Introduction to ASP.NET Core*. [online] Microsoft. Available at: https://docs.microsoft.com/en-us/aspnet/core/?view=aspnetcore-2.2 [Accessed 2 Apr. 2019].
29. Atwood, J. (2008). *Understanding Model-View-Controller*. [online] CodingHorror. Available at: https://blog.codinghorror.com/understanding-model-view-controller/ [Accessed 2 Apr. 2019].
30. MDN Web Docs. (n.d.). *Client-Server Overview*. [online] Available at: https://developer.mozilla.org/en-US/docs/Learn/Server-side/First_steps/Client-Server_overview [Accessed 2 Apr. 2019].
31. Miller, R., Vega, D., Dykstra, T., Wenzel, M. and Lambson, B. (2016). *Overview - EF Core*. [online] Microsoft. Available at: https://docs.microsoft.com/en-us/ef/core/ [Accessed 2 Apr. 2019].

32. Hui, K. (2018). *Developing Web Apps*. [online] Campusmoodle. Available at: http://campusmoodle.rgu.ac.uk/mod/resource/view.php?id=3135893 [Accessed 3 Apr. 2019].
33. Morris, S. (2018). *AJAX—What It Is, How It Works, and What It's Used For*. [online] Skillcrush. Available at: https://skillcrush.com/2018/04/26/what-is-ajax/ [Accessed 3 Apr. 2019].
34. Ahmed, R. (2019). *What Is Git | Explore A Distributed Version Control Tool | Edureka*. [online] Edureka. Available at: https://www.edureka.co/blog/what-is-git/ [Accessed 3 Apr. 2019].
35. Gibb, R. (2017). *What is Lazy Loading? | StackPath Blog*. [online] StackPath. Available at: https://blog.stackpath.com/glossary/lazy-loading/ [Accessed 4 Apr. 2019].
36. bzaar. (2018). *DawgSharp*. [online] GitHub. Available at: https://github.com/bzaar/DawgSharp [Accessed 4 Apr. 2019].
37. Chortle.ccsu.edu. (n.d.). *Transpose*. [online] Available at: https://chortle.ccsu.edu/VectorLessons/vmch13/vmch13_14.html [Accessed 4 Apr. 2019].

10. Appendices

10.1 Original project proposal

10.1.1 Detailed Project Proposal

First Name:	Simeon
Last Name:	Dobrudzhanski
Student Number:	1406444
Supervisor:	David Lonie

10.1.2 Project title

A Scrabble game with the goal to improve on existing iterations and provide better features and accessibility for casual and professional players

10.1.3 Background

I will be researching documentation on existing artificial intelligence used for other Scrabble games. The game needs to be playable by the computer and that will involve researching techniques and algorithms on how current AI works. If this is not documented, I will research more general techniques and provide the most sophisticated computer player I can develop. To take the application online I will also need to research available technologies and frameworks to put the game online and research the easiest ways for players to connect. For customization I would need to dive deeper into available Scrabble literature and read about the different rules and variants available. Potentially I would also need to analyse games from professional players to get an idea of their thinking to improve the algorithm as well as to see what changes can be made to the game to tailor it for professional players.

On the internet and mobile stores there are quite a few Scrabble games already but me and my supervisor believe they can be improved. A lot of existing products require sign-ups or payments to play and they are also quite "fancy", resulting in the need for external technologies such as Flash to run and are overall memory intensive. This project will try to create an iteration that focuses more on functionality over looks. Ideas discussed with the supervisor include multiplayer support (including computers and real players), support for multiple languages, customization of the rules

and playing field, text chat and easier access to the game. More ideas might develop over time. The bare minimum for the project is an offline two-player scrabble game, playable by a player and a computer. That would be enough to cover the project requirements. The additional ideas will be developed given the needed time and resources.

The project is important as, just by the number of Scrabble games available, it is apparent that there is a large community of Scrabble players from all over the world but there are various factors that are limiting them from connecting to one another. The aim of the project is to provide my idea of the best iteration of the game, one that works when playing with computers, is quick and easy to get into for real players, makes it easy to connect with players and provides enough customization and feeling of "comfort" for the players to appreciate and stay in.

10.1.4 Motivation

The motivation is to provide an accessible game for casual and professional players. As a player I have noticed that there are no online versions of the game played in languages other than English. After discussions with the supervisor we have noted that professional players might be playing with rules different from other players or regions. The goal is to provide a game with extensive customization features while making it easy to play and connect with friends. If successful, in an ideal situation the resulting web application will be the go-to place for Scrabble players to play and practice.

10.1.5 Aim & Objectives

Aim: Create an accessible Scrabble game with offline, multiplayer and customization features.

Objective 1: To meet the minimum requirements, the main problem is the game AI. Thus research is needed on currently existing AI used in other iterations to learn what the best techniques and algorithms are for the AI's decision making.

Objective 2: Develop an offline implementation of a two player game, playable by a real person and a computer.

Objective 3: The following objectives will be the one elevating this Scrabble honours project to a sophisticated product. Multiplayer functionality will be added (for multiple people or AIs – or both).

Objective 4: Add customization such as changing the board size (or create a custom one), rearrange tiles and multiple language support.

Objective 5: Extend the functional part of the project (back-end) to include a web-based front end as well.

10.1.6 Key Techniques

There are various Scrabble AI documents available from scientists and universities. The techniques involve the AI performing look-ahead simulations of how the game will go. Techniques for word-searching and data structures will also be looked at for creating an efficient opponent. General AI decision techniques learned from university will be researched further if needed. For the front-end, once the technology is chosen, the app will be written in the language to make the transfer to online possible. This is how I will learn new technologies out of the project.

10.1.7 Legal, Social, Ethical, Professional and Security issues

A lot of the available games do NOT use the name Scrabble, implying that the name is protected, thus the name I pick for the project must be different and one that does not conflict with any of the other games.

The project will likely be using techniques from scientific documents available already. Consideration will be needed whether I might be sued for using those techniques.

The used dictionaries for the game should avoid using inappropriate words (a language filter option could be added?)

The game will need to look different from other iterations.

10.1.8 Project Plan

Research available documents for techniques and pick the best ones for the AI to use.

Read up on chosen language and incrementally develop AI and extensively test it in-game.

Test out how the AI would perform in different situation with different rules. Additionally test if playing with multiple AIs would create issues.

Once functional, additional reading on picked web technology will be done to move the application online with a clean, minimalistic GUI and capability of connecting players and adding AIs.

10.2 Ethics Form

Student Name	Simeon Dobrudzhanski
Supervisor	David Lonie
Project Title	Scrabble Type Game
Course of Study	Honours Year
School/Department	Computing

Part 1 : Descriptive Questions			
1	Does the research involve, or does information in the research relate to:	Yes	No
	(a) individual human subjects		x
	(b) groups (e.g. families, communities, crowds)		x
	(c) organisations		x
	(d) animals?		x
	Please provide further details:		
	Game development, research will look into player thinking patterns at most.		
2	Will the research deal with information which is private or confidential?	Yes	No
			x
	Please provide further details:		
	Only research documents available online will be used	x	

Part 2: The Impact of the Research			
3	In the process of doing the research, is there any potential for harm to be done to, or costs to be imposed on	Yes	No
	(a) research participants?		x
	(b) research subjects?		x
	(c) you, as the researcher?		x
	(d) third parties?		x
	Please state what you believe are the implications of the research:		
	No physical work required		
4	When the research is complete, could negative consequences follow:	Yes	No
	(a) for research subjects		x
	(b) or elsewhere?		x
	Please state what you believe are the consequences of the research:		
	A smarter AI opponent to play the game		

10.3 Poster

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

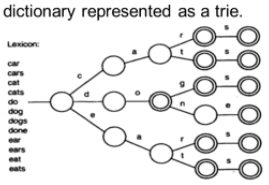
Below is an example of a game in progress with four players – two humans and two bots. High Scorer Bot plays the highest scoring word it has come up with in a limited time frame. Score calculation takes into account the usage of premium tiles as well as extra words that can be played with a given move, 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.

Bot Implementation

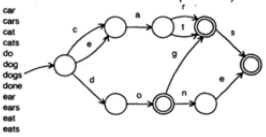
If the active player is a bot, the server uses a set of steps to get a collection of possible moves. The steps include determining the anchors (the empty tiles where a word can be connected), calculating the cross-checks for each tile (the set of letters that can be placed on a given board tile that 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).

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

10.4 Meetings Log

Meeting location and time	Meeting Details
05/10/2018, 13:00 at N432	Initial meeting with supervisor. Discussed the project (Scrabble implementation), the challenges to be faced and potential goals to meet. Also discussed how to schedule meetings and the process of the module (initial forms, project proposal, literature review, final report etc.)
02/11/2018, 17:00 at secluded room near staff office on 4th floor	Literature review had been started. Initiated a meeting with the supervisor to get feedback on it in regards to length, sections, structure and content. Advice was given on review styling, usage of headings and content addition and trimming.
06/11/2019, 17:10 at N529	Additional meeting regarding literature review progress and improvements to make.
08/11/2019, 17:15 at N529	Additional meeting regarding literature review progress and improvements to make.
09/11/2018, 21:45 over email	Sent an email to supervisor with near final version of the literature review. Asked for quick read-through and feedback. Additional elements that needed adding were table of contents and introduction. Structure needed improvement.
10/11/2018, 09:45 over email	Received feedback from supervisor regarding attached literature review from previous night. Report had markings in red to remove content, in green to add content and in blue for general advice on what to do with a particular section. Reminder was given to add section numbers.
11/11/2018, 18:45 over email	Received approval from supervisor to submit literature once he had a quick look of it once it was updated after following previous advice.
20/11/2018, 16:30 at secluded room near staff office on 4th floor	Initiated meeting to discuss Requirements Analysis. Advice was received on length, content to include such as MOSCOW method for prioritization, structure, functional and non-functional requirements, technologies to use.
29/11/2018, 15:00 at N528	Initiated meeting with supervisor to receive feedback on the progress of the Requirements Analysis regarding content, styling, structure.
30/11/2018, 10:10 over email	Received feedback on previously attached Requirements Analysis that was near complete. Advice was received to correct spelling errors, improving the look of the tables by adding columns with numbers and to provide expanded reasons on technology selection with reference back to the literature review. Analysis was afterwards submitted.
12/02/2019, 14:00 at N528	Initiated meeting with supervisor to discuss progress on the implementation of the project. I had forgotten the EU to UK adapter for the laptop charger and could not show it. Instead I drew up the web page and talked about the current structure of the solution (talked about models, single web page, mix of technologies and how they worked together). Mentioned difficulties in automatically generating a database from the implemented models using Visual Studio's/Entity Framework Core's features. Advice was given regarding user input – instead of using a text field to put in letters on the board, to use buttons for the letters instead due to easier implementation. Positive feedback was given regarding solid start and chosen direction of the implementation regarding the choice of technologies and their usage.
26/03/2019, 12:30 at N533	After finishing courseworks for other modules and making good progress on the honours project, showed project to supervisor which now had the capabilities of a human-only game with an implemented dictionary and word validation, front-end and back-end validation for various user errors

	and logs of user moves and scores. Supervisor tried the game and approved the current implementation as demo-ready and made suggestions on making the user logs less text-heavy (eventually turned into rows containing word and score). The only major component missing at this stage was an AI opponent and discussed how it would fit in the solution.
08/04/2019, 13:00 at N424	Initiated meeting with supervisor to demonstrate near complete implementation on project. Displayed a working example of a game with human and computer players and demonstrated the ability of AI opponents to make moves. Also demonstrated improved front-end design and a game in a foreign language (Bulgarian). A major drawback noted by myself and the supervisor was the time taken for the computer to take a move and discussed options for shortening it (such as by optimizing checks for anchors and cross-check sets). Other than that, there was not much more feedback given as it was near complete.
17/04/2019, 12:30 at N528	Initiated meeting to receive feedback on poster. Advice was given in renaming sections (such as changing Methods to Technologies Used and the best naming of other sections for current content on poster). Discussed level of detail to add on poster and it was agreed to focus on AI implementation and to add a graph or picture that quickly demonstrates the workings of the algorithm for move generation.
23/04/2019, 15:00 at N533	Initiated meeting to receive feedback on final report. The covered section up until Implementation and a half-complete Implementation section was ready. Provided a quick run-through of what each section talks about, what future sections will be about and asked for feedback. Feedback included to add numbers to headings, to remove unnecessary content (such as comparison of low-level and high-level languages which was deemed as basic knowledge for the report readers), to adjust some figures and to add more captions to them. Other than that feedback was positive.
28/04/2019, 16:00 over email	Mostly-complete report was sent to supervisor for feedback. Many useful comments were added regarding reference structure, section renaming, adding text to improve narration, adding or changing figures, moving some text from figures to section body, putting some content in tables, adjusting text-size in some figures, adjusting consistency of section heading indentation and to provide appendixes.
29/04/2019, 16:00 at N533	Initiated a meeting to get approval on the changes done with respect to the advice given from the previous email. Went over the marking grid to ensure each requirement for an excellent grade was met. Feedback was positive and report was to be sent on the same date.

10.5 Project Log

The project source code is available on moodle. As Git was used for version control, the following is a log of around 60+ commits.

```
commit 7967196539767b21ca27d4e7e46054211f427fed
Date:   Wed Apr 24 04:39:04 2019 +0100
Improvements to code for demo

commit 52bada5b6e45c5e07d7abc3870e8b373c969154e
Date:   Tue Apr 16 15:39:41 2019 +0100
Added more documentation

commit efa216f3bf880c53cb272227710c8475c66d0529
```

Date: Mon Apr 8 23:38:43 2019 +0100
Added enums, bug fixes in controller, BotType and BoardTileType models, some documentation

commit 964bec81314c2525d21f0506171662ab04e03c30
Date: Mon Apr 8 18:47:24 2019 +0100
User can pick tiles to redraw, board tiles score bonuses get removed when filled.

commit 5a50ef852a41499402e9c910d635889a4c339f40
Date: Sun Apr 7 19:42:01 2019 +0100
Added Rack Balancer Bot and ability to end game and redistribute letter scores on end

commit 3a15e65e96fbf8071df1c796d7e1cfcecc084619
Date: Sun Apr 7 01:47:05 2019 +0100
Fixed bug in move making logic, combined anchor and cross-check validation and detection, added ability to end game, bingo scoring, 4 player play, statistics, play counter for players (play may have multiple words/moves)

commit ca0a1023665e7b58911063cd54b18ab6a66c714d
Date: Fri Apr 5 18:10:00 2019 +0100
Added game in bulgarian and shuffle functionality

commit 4e7ff407e49c4fe6eaed9c300229a4a239688a29
Date: Thu Apr 4 21:39:24 2019 +0100
Some cleanup

commit 28a1a13ef141a1d669435306ff817a6cc0e66da6
Date: Thu Apr 4 01:11:12 2019 +0100
Players can now mark every word played, new words come up as new

commit 02a2441045a0e1e6985f3825de0b864002083a72
Date: Wed Apr 3 20:18:11 2019 +0100
Added keyboard controls and loading spinner to game

commit 55991e6601280291aaf27d141620e3b776a35076
Date: Tue Apr 2 04:06:25 2019 +0100
Computer players now make moves

commit 648f08bcd89ec9ba2b3690baf633ac29d3dc896c
Date: Mon Apr 1 22:41:36 2019 +0100
Added timer for move generation, randomized generation, optimized code

commit b70010b5c60dce78df9f368383d0290d288821f2
Date: Mon Apr 1 06:20:38 2019 +0100
User can now click on a generated move to have it displayed on the board

commit 48bb5928b640d4372b20e0da6c1a462847afcf28
Date: Mon Apr 1 02:26:55 2019 +0100
Added move calculation for plays in opposite direction when making a move.

commit 10873d44567faf3862e15415f9fdf9f8cdc05d8a
Date: Sun Mar 31 23:43:34 2019 +0100
Anchors used are now those around placed tiles, allowing for much greater move generation. Move generation cannot get much better.

commit 95e87049637b43e12a90b5bf43037d81f4205e89
Date: Sun Mar 31 18:40:51 2019 +0100
Improvements to move generation logic. Left and right extensions work correctly. Need to test left limit. Anchors are tiles on board, will try to add potential squares.

commit 4f5c186da4d4f8eb9c46dbfc23bd5c6c1ee1e62c
Date: Sun Mar 31 00:04:33 2019 +0000
Added blank tile play and validation, improved move generation logic

commit e5b51d58da0d351896e67b6d165e40a970026717

Date: Sat Mar 30 04:50:39 2019 +0000
Improved computer logic, moves have been correct assuming from debugging. Added move generator on page.

commit d8db3cd3a80746b5831a3f8d6bbfd326780a1d14
Date: Fri Mar 29 08:02:58 2019 +0000
Improvements to move generating, fixed cross-checking logic, changed everything vertical to transposed

commit 8ae07c666a0d19d97c4d0946d333a70159d7c266
Date: Thu Mar 28 23:04:24 2019 +0000
Style improvements, more functionality added

commit 48fd9b4dabe8786ac2c76938ef70d300fc07a64b
Date: Thu Mar 28 20:24:41 2019 +0000
Move generation working for both horizontal and vertical plays, improvements in look

commit 96789104342f955df10c0e6de1733fae3ab90748
Date: Thu Mar 28 13:28:49 2019 +0000
Added move generation for horizontal board and some overloaded classes in Rack to help

commit a6fbfc1f60e04eb2dea60c07ec2fcc197d5b2d28
Date: Wed Mar 27 22:24:11 2019 +0000
Cross checks from horizontal and vertical plays are now combined, added official Scrabble word list from 2015

commit c9c152d9d67b003aa65c90401f3e289d7baaffa7
Date: Wed Mar 27 17:13:21 2019 +0000
Added logic on server for transposing board, getting coordinates of anchors from boards, getting horizontal and vertical cross checks. Get Moves button on page still in development. Added CoordinatesEquailityComparer class to compare arrays of coordinates

commit 8916e59af0209a09ebc42e26a5826721df5dafe1
Date: Wed Mar 27 00:51:02 2019 +0000
Player history of moves is stored in the database, improvements in look and responsiveness

commit c618ff4dfb9d1bbb1b46ce3520d44590a5c1a1a3
Date: Tue Mar 26 00:49:37 2019 +0000
Added most validation checks to server logic, still need to do board transposition. Added used rack tiles to submission from JS, adjusted methods in Helper and added new ones.

commit 5561d8ff46b976902b851a63121814f4e0760326
Date: Mon Mar 25 19:39:22 2019 +0000
Added anchor checks in server logic

commit 7007bfdd26f6ba40252b025a09c36765f6223ee5
Date: Mon Mar 25 16:50:55 2019 +0000
Adjusted seeding to match real game, improved scoring, making moves now removes tiles from rack and takes some from pouch

commit 3882cc918312a30c12f87fb8e9513573f758b770
Date: Mon Mar 25 12:35:37 2019 +0000
Added player scoring

commit 071671b552cb6295b396604bc1d4d0e383d1cec1
Date: Mon Mar 25 03:47:13 2019 +0000
Improvements to play-connection checking and anchor checking logic

commit fe45036f9ff6c04de0aa774196a0914408399c2d
Date: Mon Mar 25 02:14:20 2019 +0000
Added DawgSharp nuget package and English dictionary checking, moved some code to separate file and methods

commit 2a7dfcd1f2531208281754b32adcc9e4c112ef5b
Date: Sun Mar 24 23:13:59 2019 +0000
Major changes. New moves are correctly detected and calculated on server. Anchor, starting move and play-connection checking has been fixed. Added check for reporting isolated tiles.

commit 7ca50a462d26aa7902181e3c2e6efb0855fbab24
Date: Sun Mar 24 17:51:11 2019 +0000
Changed logic, play checking now uses a transposed board to verify plays. Submission only includes played tiles and not whole moves. Need to amend server logic and horizontal+vertical moves get counted as invalid.

commit b142ecd1fd6757662f14892edadf6d23c3449979
Date: Sat Mar 23 23:39:46 2019 +0000
Single tiles placements have been fixed by checking surrounding tiles to determine type of play. Does not yet work for plays that can go both ways. Fixes in anchor and play-connecting logic.

commit 11d455dff21e476ef2dc7e85066bc99376b12866
Date: Sat Mar 23 22:04:48 2019 +0000
Now checks ancestors and successors (gets all connected tiles) Does not work well when placing a single tile. Logic should probably be in server.

commit 8e07d848ece68c3ca7d2820c14a3b0e6669acea6
Date: Sat Mar 23 19:22:50 2019 +0000
Added scoring. Word submissions now include tiles between start and end of play but not before or after play yet

commit ce743723da5a499fb11ab70bafc3ebc49a09dd47
Date: Sat Mar 23 16:52:09 2019 +0000
Added output next to board

commit 1d537f9bf703ca563e87f9f2605e3f683f7d8b52
Date: Sat Mar 23 15:05:41 2019 +0000
Anchor and play connecting logic improved, added animated updates and clear button, JS cleanup

commit 05a950d5966c17e8a221a4e1f66354ccc90d5177
Date: Sat Mar 23 01:04:00 2019 +0000
Added check for connected plays

commit 9221e95e3962f3a54d58705cdfed4a4879a2e513
Date: Fri Mar 22 23:49:59 2019 +0000
Now cannot play without using anchor

commit 21cd7bac9355765658d7a58b39c0320782adc727
Date: Fri Mar 22 22:00:06 2019 +0000
Added anchor checks, start check, resetBoard method

commit 358ce7d7d0fd52e7b9639c4e42268ad40c4a9730
Date: Fri Mar 22 18:50:29 2019 +0000
Submitting tiles on board saves them in db

commit f1d529e3d405e8a9d204c7ef19b4b50db08bce98
Date: Fri Mar 22 15:24:41 2019 +0000
Added WordDictionary to Game and CharTile, ajax call to Index to refresh content

commit fc07e90f545648af4ebe8d766aca1f2b245cf392
Date: Thu Mar 21 21:07:18 2019 +0000
Restructured models, seeded db, changes on look

commit 18dee968a12e874b31d6e076d66e786c7cad395e
Date: Tue Jan 8 02:58:39 2019 +0200
Need to seed and test

commit 164a7ad9bcccf04160c09f8b51b2f281570fa3d8
Date: Mon Jan 7 16:38:00 2019 +0200
Fixed Index.cshtml

commit 42c47e4a34b5ce8999078bb36171ab20a2ceb205
Date: Mon Jan 7 16:34:34 2019 +0200
Added NotMapped in Game class

commit ec4fa90d3ed6a702cbe81f59c8b6d0622fd0be2d
Date: Mon Jan 7 16:16:31 2019 +0200
Relationship improvements

commit 9545c364251af03d402be5be2880e796e328cfe8
Date: Mon Jan 7 15:33:38 2019 +0200
Added foreign key constraints

commit 80443a300cc8e1e10a30265bf61983e0cf10e6ab
Date: Mon Jan 7 15:14:15 2019 +0200
Added relationships for EF Core

commit 30d7b206d786d36451a95723a23a85d4e50e5ba6
Date: Sun Jan 6 16:53:13 2019 +0200
Added empty constructors for all models

commit d1b8cec42e1aa6357a1307f8a9d1d84530bc223e
Date: Sun Jan 6 14:10:40 2019 +0200
Changed abstract class GameLanguages to enum Language for db testing

commit 63242a92bd0a4a5b1c3425e1b39545bda98fcea8
Date: Sun Jan 6 00:32:51 2019 +0200
Added random IDs for all models

commit 007383c3da12b27040657e4fd1d82617f0c02714
Date: Sat Jan 5 23:55:48 2019 +0200
Added parameterless constructors for all models

commit e19d5e6218c0a3145a27b63492cf91534a44ef56
Date: Sat Jan 5 23:30:05 2019 +0200
Some IDs for EF added, 2d array changed to list for BoardTiles

commit 89e7c2a533a1912437518247d529c5793b958f81
Date: Fri Jan 4 03:14:34 2019 +0200
Already placed tiles are usable without being deducted from player rack

commit f37d1734673c364c0865f68c5b9508b56acd62cf
Date: Fri Jan 4 00:04:54 2019 +0200
Vertical plays added by clicking twice on tile
commit fec086a2caf1a11228d882cd404280bd10804231
Date: Thu Jan 3 19:20:24 2019 +0200
Input length limited based on board length, tile selection resets on backspace

commit dd250105d13a804bf60c5c7c7812924ead9a81ec
Date: Thu Jan 3 19:00:25 2019 +0200
Board text clears on active tile changes

commit bad54cb416d17362fc9fdc92f15950c63047379c
Date: Thu Jan 3 18:23:22 2019 +0200
Improved JS rack/input live updates

commit 0db3bcde70fc3e5bb1c07ddc589f59e895701ab3
Date: Sat Dec 29 04:50:20 2018 +0200
Added word entry, rack, board updates, css classes for active tiles

commit d8a151383a7f16bb9283f0fe1338773ef59bab06
Date: Fri Dec 28 18:34:40 2018 +0200
Inititial release

10.6 Source code

Source code can be found attached to submission and at https://github.com/simmeon1/Scrabble/ .