Fall 2017

# A Scrabble Artificial Intelligence Game

Priyatha Joji Abraham

# A Scrabble Artificial Intelligence Game

A Project Presented to

The Faculty of the Department of Computer Science

San Jose State University

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

by

Priyatha Joji Abraham

December 2017

The Designated Project Committee Approves the Master's Project Titled

A Scrabble Artificial Intelligence Game

by

Priyatha Joji Abraham

APPROVED FOR THE DEPARTMENT OF COMPUTER SCIENCE

SAN JOSE STATE UNIVERSITY

December 2017

Dr. Chris Pollett, Department of Computer Science Date

Dr. Philip Heller, Department of Computer Science Date

Dr. Robert Chun, Department of Computer Science Date

APPROVED FOR THE UNIVERSITY

Associate Dean Office of Graduate Studies and Research

ABSTRACT

A SCRABBLE ARTIFICIAL INTELLIGENCE GAME

by Priyatha Joji Abraham

Computer AI players have already surpassed human opponents in competitive Scrabble, however, defeating a Computer AI opponent is complex and demands efficient heuristics. The primary objective of this project is to build two intelligent AI players from scratch for the Scrabble cross-board puzzle game having different move generation heuristics and endgame strategies to evaluate their performance based on various benchmarks like winning criteria, quality of moves, and time consumption. The first AI selected is the most popular Scrabble AI world champion called Maven. It generates a three-ply look-ahead simulation to evaluate the most promising candidate move and uses four different heuristics for the fast move-generation. The second AI, Quackle, is the strongest alternative Scrabble AI to Maven. It generates its best candidate move by using a three-ply look-ahead simulation and win probability estimation. In this project, we primarily focus on the end-game heuristics because end-game sessions are complex real-world situations where the move options are limited and require expert techniques and model strategies to maximize the reward. Moreover, the basic game heuristics used in the mid-game are not sufficient for an end-game. For this project, we created four variants of Maven AI. After conducting experiments, we observed that Maven Q-Sticking slow-endgame AI performs better than other AI variants like Maven Q-Sticking AI, Maven slow-endgame AI, No Q-Sticking AI and Quackle AI.

ACKNOWLEDGEMENT

I would first like to express my sincere gratitude to my project advisor, Dr. Christopher Pollett for his constant guidance and enduring support throughout this project in spite of his hectic schedule. He always steered me in the right direction by providing valuable suggestions, and clarifications of abstract concepts. I would also like to extend my thanks to my committee members, Dr. Robert Chun and Dr. Philip Heller, for their suggestions and time.

I must express my very profound gratitude to my husband Milce George Rajeev for providing me with unfailing support and continuous encouragement throughout my years of study. This accomplishment would not have been possible without him. Finally, I would like to thank my family and friends for their constant motivation.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

# 1. INTRODUCTION

Artificial Intelligence (AI) is a collection of techniques used to simulate human decision-making skills. Since the 1950's, AI has played a significant role in the game industry [1]. AI strives to build intelligent agents that can perceive and act rationally to accomplish goals. These learning agents respond and resolve well-defined problems. In the case of computer games, if there is no win situation then this intelligent learning agent memorize not to repeat previous mistakes. Thus, games provide an ideal domain for measuring the potential of AI applications. Today, machines defeat human players in the gaming field by playing abstract and strategic board games using the techniques of AI [1]. Scrabble is a popular crossword board game which is interesting from an AI perspective because the players play with gradual revealed information. The goal of this project is to develop variants of two popular AIs for Scrabble, Maven and Quackle, and to perform various experiments on their effectiveness in end game situation — situations in which the amount of information the AI has increases rapidly.

The primary challenge of Scrabble is it is a game of imperfect information as each rack of tiles is hidden from the opponent [1]. Due to this missing information, it is hard to predict the exact successive move of the opponent as its state is unknown to the player and the optimal move is not defined. Secondly, the machine should quickly generate the promising candidate moves for a given rack and current board state. To do that, the computer machine must be modeled to apply efficient opponent strategies wisely on each turn with a quick move generation algorithm. Different heuristics are applied in these

move generation algorithms. Eventually, playing out the unplayable tiles like Q, Z, J during an end-game is crucial and challenging.

To develop our Scrabble AI implementations, we researched related works. Currently, Maven and Quackle are the leading Scrabble AI's. Maven was created in 2002 by Brian Sheppard [3] whereas Quackle is an open source Scrabble AI developed by Jason Katz-Brown [4] and John O'Laughlin [4] in 2006. Maven is also the current world champion computer player and best known AI that has beaten human opponents. Maven uses a three-ply stochastic look-ahead simulation technique for the move generation [3]. In game theory, a ply refers to a player's turn. The heuristics and strategy of Quackle are similar to that of Maven but not exactly same. Quackle incorporates a program module called kibitzer to rank the most promising candidate moves quickly. These moves are further evaluated using a simulation engine that can simulate 100 to 300 random racks for each candidate move followed by a three-ply-look ahead move generation.

Maven and Quackle have both defeated the best human champions in tournaments [3,4]. However, Maven and Quackle have not fought against each other. A related work was done by C. Josephson [6] and R Greene [6] from Stanford in 2016 where an AI is built with Monte Carlo Simulation and Appel and Jacobson's [5] move generation algorithm to beat Quackle. The new AI focused on mid-game scenarios, and they could not implement an end-game strategy due to time limitations.

In this project, we built three variants of Maven incorporating different heuristic algorithms which are especially useful for the critical end-games. We mainly focused on end-game heuristics because Scrabble end-game sessions are very crucial. Each end-game move must be placed very carefully as it can break the leading player. Quackle AI serves as the benchmark to analyze the performance of our Maven AI variants.

This report is structured as follows: Chapter 2 gives a basic background overview of Scrabble game and explains Scrabble terminology. In Chapter 3, we discuss the techniques of world-championship player Maven. Chapter 4 showcases detailed explanations of Quackle AI. Chapter 5 presents the different variants of Maven AI implemented in our project and the heuristics algorithm used in these AIs. In Chapter 6, we compare the simulations of two AI. Chapter 7 discusses the experiments and results conducted to ensure the correctness of the program. Chapter 9 concludes the project and explains the future work on this project.

## 2. BACKGROUND AND BASIC CHARACTERISTICS

In this chapter, we provide the background, basic overview of Scrabble rules, and terminology, game strategies and explains the board representation used in our game. Scrabble is a world-famous classic board game played in tournaments by two to four players invented by Alfred. M. Butts during the 1930s [3]. In this project, we concentrate on a two-player AI as stated before. The ultimate goal of Scrabble game is to build valid English words on the game board and collect maximum points than the opponent by following a set of rules and restrictions. Figure 1 referred below represents a 15x15 grid Scrabble game board used officially.

The game has a tile bag that holds 100 tiles and is shared among all the players. Each group of 7 tiles forms a rack. The letters left out on a rack after each move is called 'rack leave.' Out of the 100 square tiles in the tile bag, 98 tiles are English alphabets, and two tiles are blank. Each tile is associated with a predefined letter, ranging from A to Z, and value, ranging from zero to ten. Blank tiles are the wildcards in this game because they are used in place of any letter. These tiles are worthless and do not earn any points. The point distribution depends upon the tile frequency. High-frequent tiles such as A have a low point of 1 whereas low-frequent tiles like Q have a high score of 10. The tiles are drawn randomly from the tile bag on each turn to form a rack.

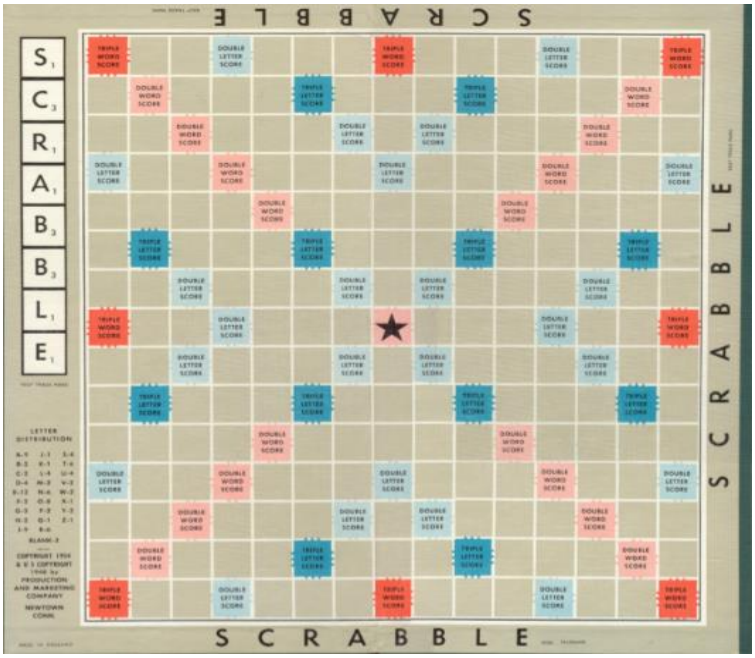



**Figure 1. Scrabble game board [5]**

## 2.1. Scrabble Bonus Points:

In Figure 2 shown below, we can see that some squares are marked as ‘DW,' ‘TW,' ‘*3,’ ‘*2’ that runs diagonally across the board. They are premium squares such as ‘Double Word,' ‘Triple Word,' ‘Triple Letter,’ ‘Double Letter’ and laying a tile on these bonus squares can earn a double or a triple score of the word or letter. If a player uses all the tiles on the seven tiles on his rack, then it is called a ‘Bingo’ moves.

As we are using computer players, we have an added benefit than human players because a computer player can quickly find the bingo words from their dictionary in memory. In this project, we always give preference to placing bingo words because they can score a bonus of 50 which constitutes 1/8th of the total score in tournament games [7].

## 2.2. ASCII Board Representation:

In our project, we created an ASCII board representation of the Scrabble board. As the game board is a 15x15 square matrix, we numbered rows from 0 to 14, and labeled columns as alphabet letters from A to O. To represent the blank tiles in the rack, '_' is used in the program. When a player draws a blank tile from the rack, the program automatically generates a replacement letter. Figure 2 referred below shows the ASCII Scrabble board notation used in this game.

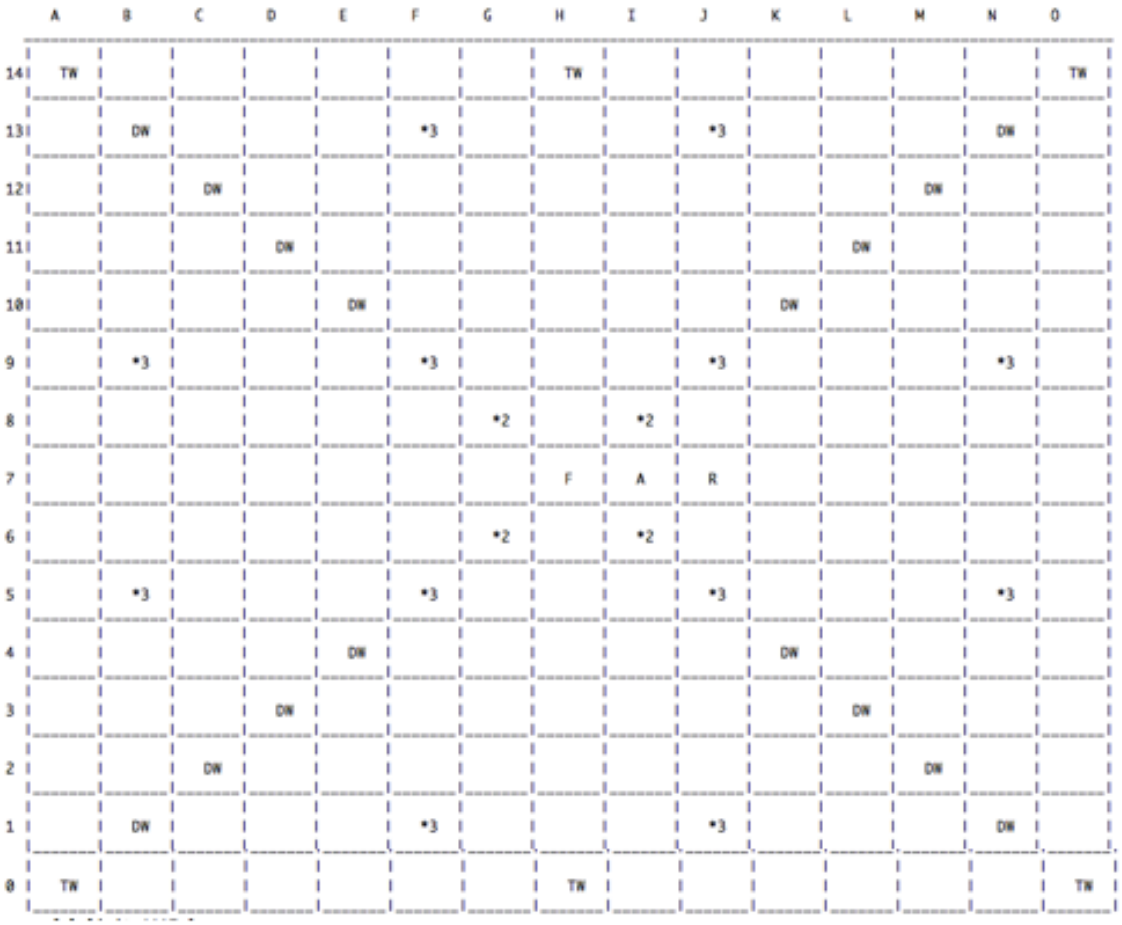



**Figure 2. Scrabble board notation using ASCII characters**

## 2.3. Legal Moves:

Scrabble has a strict ruleset for placing the words on the board. The tiles are often placed in a cross-word manner, but all the moves must be legal according to the Scrabble ruleset. A Scrabble game does not allow diagonal plays. Hence the player can only lay the words in either horizontal or vertical direction. During the opening of a Scrabble game, a player must set the tile beginning from the center of the board. Subsequent plays would be an extension to the words formed in the preceding game. The letters that are already on game board are called anchor tiles, and the letters we use as an anchor tile extension for the current turn are called hook tiles. Each player can only place the letter tiles from his rack, and the hook tiles must be anchored to the anchor tiles. If there are no proper anchor tiles on the board, then the player could pass the turn that scores zero points.

## 2.4. Common Game Strategies:

Game strategies play an ideal role in Scrabble AI. These strategies help the player to perform actions or decision-making skills to win the game. The two kinds of game strategies which are adopted by all the AI players are straightforward strategy and defensive strategy.

In the straightforward strategy, the greedy players place the longest word on each turn that earns the highest score [2]. So we can also call it as a greedy strategy. A drawback of this method is that it can generate a lot of hot spots for the opponent [2]. These hot spots on the game board let the competitors place a high score move in the next turn. This hot spot move may be a bingo move or a premium bonus move. Therefore, we

must remember that the highest scoring move is not always the best one. An intelligent AI player must efficiently apply different strategies and heuristics at each turn depending upon the current board state.

In the defensive strategy, the player tries to minimize the hot spots for opponents by playing two-letter words. B. Sheppard [3] explains three kinds of defensive approaches played by his computer player, Maven during the end phase of the game. The next chapter explains these heuristics in detail. Our Maven AI executes all these four tactics using a three-ply look-ahead simulation algorithm whereas we found that Quackle does not adopt all the defensive approaches.

## 2.5. Scrabble Game Terminology:

- Board: 15x15 grid game board of Scrabble
- Tiles: Fills each board square; each tile has a letter and value
- Tile Bag: Holds 100 tiles, 98 are letter-point alphabet tiles and two are blank tiles
- Blank Tile: Tile with no value and earns zero score; denoted as '_' in the program
- Rack: Group of seven tiles hold by each player
- Rack Leave: Letters left out on the rack after one play
- Bingo: A word play uses all the seven letters on the rack, earns a bonus score of 50
- Premium Squares: Squares that earn bonus points like DW, TW, *3, *2
- DW: Double word bonus square runs diagonally, earns double the score of total word
- TW: Triple word bonus square is a premium square, earns triple the score of total word
- *3: Triple letter bonus square is a premium square that scores triple of the letter

- *2: Double letter bonus square is a premium square that scores double of the letter
- Anchor Tile: Extended Tile on the board placed in any of the previous game
- Hook Tiles: Tiles from the rack used to hook or extend
- Hot Spots: Excellent squares on the board with excellent bonus-scoring opportunities
- Ply: Turn of a player or half a move in a two-player game
- End Game: The last phase of the game when there are no tiles left in the bag

# 3. MAVEN AI OVERVIEW

As Maven is the best-known AI, we chose it as our first baseline AI and implemented from scratch. The Maven AI was created by Brian Sheppard [3] in 2002. It uses a Directed Acyclic Word Graph (DAWG) data-structure for dictionary representation and word generation [3]. In Maven, each move is influenced by three main factors. These three elements are the current score, the player rack, and the current game board. In our project, we focused on the end game heuristics because Scrabble endgames are crucial real-world scenarios and we cannot apply basic greedy human-strategies that gains maximum points for an endgame. The endgame determines the success or failure of Scrabble games, especially in tournaments. Moreover, tournaments play is time-limited (25 mins to finish all plays). Maven uses the concepts of move generation, heuristics, board evaluation and a three-ply look-ahead as explained below:

## 3.1.Dictionary:

An Official Scrabble Players Dictionary (OSPD) contains over 100,000 words and is freely available on the Internet. Maven uses this dictionary as their knowledge base [3]. In our AI, we initially used a similar Scrabble dictionary, downloaded from the internet, with 178,696 words (1.8 MB size). However, when we tried to simulate an end-game situation of the game board, we found that considering these many words is time-consuming and unnecessary. Therefore, a trimmed dictionary was created for our testing purposes that consist of 232 words (1 KB size). The irrelevance of words is due to the existence of the redundant letter words which are unusable for Scrabble. For instance, if you look into the tile distribution of Scrabble there is only a single ‘Z’ in the tile bag.

However, our dictionary comprised of words like ZYZZYVAS, ZZZ which was unnecessary to keep because in a legal play we can only use at most 2 Z's in a word (including the blank replacement tile).

## 3.2.Lexicon Representation:

Maven uses a Directed Acyclic Word Graph (DAWG) data–structure presented by Appel and Jacobson as cited in [5] for storing the words in OSPD. DAWG's finite-state automation uses a minimal representation of a trie data structure to represent the entire dictionary or lexicon for fast move generation algorithm. A Trie is an ordered prefix search tree with an empty root and prefix labeled nodes. A common-prefix is shared by two or more words. The leaf of the trie holds the complete word and is known as the terminal node. However, tries can consume more space because it can contain duplicates. DAWG stores an entire lexicon in a reduced trie form so that the structure consumes only minimum memory or space as compared to the search trees. Figure 3 and Figure 4 shown below represents the trie data-structure and DAWG data structure of the given lexicon respectively.

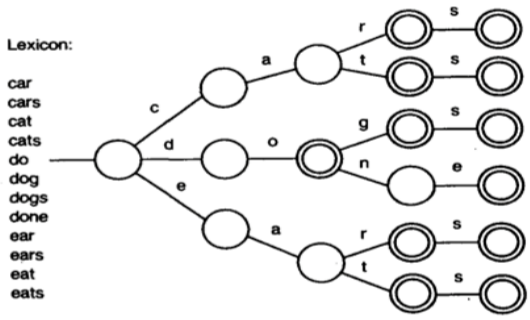



**Figure 3. Trie with its lexicon [5]**

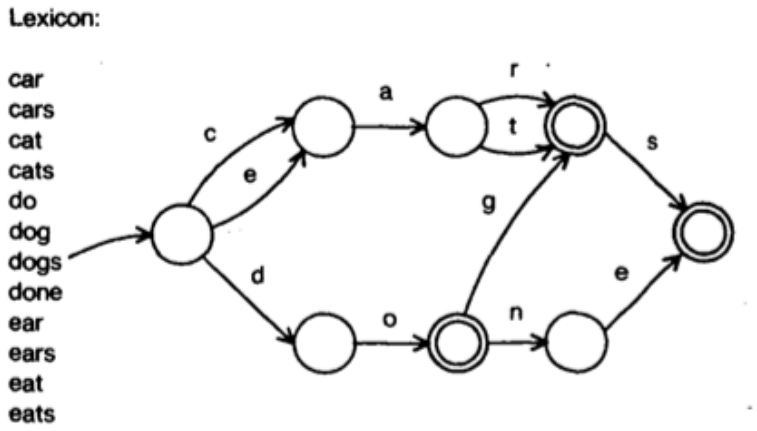



**Figure 4. DAWG with its lexicon [5]**

Thus, the DAWG representation reduces the lexicon representation storage space to 175 KB, which allows to keep it entirely in the memory [5]. DAWG is also fast because it has a backtracking search strategy. In our case, as we had a trimmed dictionary of 232 words of size 1 KB, we were satisfied with either trie or the word-list representation.

### 3.3. Move Generation:

Maven's move generator discovered the possible legal moves for the current game state and assessed the future score based on the current game. To achieve that, it applied different kinds of heuristics efficiently on each turn and found up to 20-30 candidate moves [3]. The project follows the same move generation algorithm. We will explain the exact algorithm used in our project which is similar to Maven move-generation algorithm. For each move generation, we look into our rack state, board state, and score. Suppose the player have a rack like this 'MFRFGTH' and have a random board state with many words on it. Consider a situation where we selected 'FARM' as the best possible word which must be placed across the board word 'AND' anchored to the letter A. To play 'FARM,'

six main constraints have to be checked during move generation before placing the letter on the board.

The pseudocode of the move generation algorithm is explained below:

1. Check if an empty square exists to the left of the 'F' as 'F' is the first letter.
   1. a) IF TRUE, continue to next step
   1. b) ELSE, check if a horizontal legal word formation exists.
2. F is either present in our rack or board as anchor letter
3. A is either present in our rack or board as anchor letter
4. R is either present in our rack or board as anchor letter
5. M is either present in our rack or board as anchor letter
6. Check if an empty square to the right of the last letter M exists
   6. a) IF TRUE, just place the word on game board permanently.
   6. b) ELSE, temporarily place the word in the desired direction and continue checking legal formations using or *placeTilesVert()* explained in Section 3.4 and continue the process.

## 3.4. Legal Placements:

As we mentioned before, all the moves placed on the game board must be valid legal moves as per Scrabble rules. We use the java class 'LegalPlacement' in our project to validate the correct placement of each move. For each move, we need to identify the direction whether it is a horizontal or a vertical move and check the proper word formation with any other letters on the top, bottom, left and right squares of the placement letter. The functions *placeTilesHoriz()* and *placeTilesVert()* in this class are used to verify the valid word formation with the adjacent non-empty cells in the board when the words are laid horizontally or vertically respectively. The pseudocode of these functions are explained below:

i. *placeTilesHoriz()*

   1. If possibleWord contains boardWord, then split the possibleWord that needs to

check legal formation with boardwords into leftStr and rightStr, leftStr contains the substring before boardword and rightStr contains substring after boardWord.

2. If the direction is horizontal, then do the following:

2.a) Place the leftStr on board and check the below steps:

2.a.1) If the leftStr makes a legal word formation with the non-empty left cell, raise the leftcellTrue flag

2.a.2) If the leftStr makes a legal word formation with the non-empty top cell, raise the topcellTrue flag

2.a.3) If the leftStr makes a legal word formation with the non-empty bottom cell, raise the bottomcellTrue flag

2.b) Similarly place the rightStr on board and check the below steps:

2.b.1) If the rightStr makes a legal word formation with the non-empty right cell, raise the rightcellTrue flag

2.b.2) If the rightStr makes a legal word formation with the non-empty top cell or bottom cell, raise the topcellTrue or the bottomcellTrue flag correspondingly.

2.c) If any of the leftcell/ rightcell/ topcell/ bottomcell flag is true for leftStr or rightStr,
then collect the left/ right/ top/ bottom words on the board respectively and check if the collectedwords on the board form a valid word as per the OSPD.

3. If the new possibleWord is a valid word, then calculate its score and place the word horizontally.

4. Else, ignore the word.

ii. *placeTilesVert()*

1. If possibleWord contains boardWord, then split the possibleWord that needs to check legal formation with boardwords into topStr and bottomStr, topStr holds the substring before boardword and bottomStr holds substring after boardWord.

2. If the direction is vertical, then do the following:

2.a) Place the topStr on board and check the below steps:

2.a.1) If the topStr makes a legal word formation with the non-empty right cell or left cell , raise the leftcellTrue flag or the rightcellTrue flag respectively.

2.a.2) If the topStr makes a legal word formation with the non-empty top cell, raise the topcellTrue flag

2.b) Similarly place the bottomStr on board and check the below steps:

2.b.1) If the bottomStr makes a legal word formation with the non-empty bottom cell, raise the bottomcellTrue flag

2.b.2) If the bottomStr makes a legal word formation with the non-empty left cell or right cell, raise the leftcellTrue or the rightcellTrue flag correspondingly.

2.c) If any of the leftcell/ rightcell/ topcell/ bottomcell flag is true for topStr or bottomtStr, then collect the left/ right/ top/ bottom words on the board respectively and check whether the collectedwords on the board form a valid word as per OSPD.

3. If the new possibleWord is a valid word, then calculate its score and place the word vertically.

4. Else, disregard the word.

### 3.5. Heuristics:

Heuristics are a set of evaluation parameters which are essential to instantly solve the problems when the traditional approaches are often slow, especially in AI. Many of the search algorithms use a heuristic function at each step of branching. The primary goal of a heuristic function is to find the best-matching solution of the original answer.

Maven uses four main heuristics to select the most promising moves for a player. The first heuristic is called Vowel – Consonant Balance for balanced rack management [3]. This heuristic examines if the rack has a right mix of Vowels and Consonants. To do that, we can take the ratio of Consonant/Vowel in our rack and evaluate if it is higher than a threshold value, assign a positive value. If the rack holds either a full set of Vowels or Consonant, then we assign a negative value. However, Maven did not consider a good rack-leave.

The second heuristic is known as U-With-Q-Unseen [3]. Using this heuristic, we give priority to play the words that contain a combination of Q and U. If any candidate words in our most promising word list have a mix of Q and U, then assign a high reward to that word. Words with Q and U must be played very quickly. Else it could get stuck in our rack which creates a 'Q-Sticking' scenario. Sticking with these tiles is a risky situation for the player. Therefore, we must play the words that have a Q and U.

The third heuristic is known as Hot-Spot Block where a board-square near the premium squares are blocked by the player in the current turn. It is a common human-

move generation defensive strategy. These hot spots offer hints to the player to where the tiles must be placed. Also, it is essential to know when to use this strategy because we cannot use the defensive strategy in every turn. Particularly during a mid-game, it is favorable to use a straightforward strategy instead of a defensive one. However, we use a defensive approach in end-games.

The fourth heuristic is known as First-Turn Open [3]. This heuristic implies the importance of playing the first turn with a few tiles. Placing smaller words on the game boards eliminates the creation of hot spots for the opponent for the next turn. Moreover, Maven prefers to play for the first turn as in some cases the first player have got a higher probability of winning the game.

### 3.6. Maven Game Phases:

There are three major search phases for Maven: Beginning to Middle game phase known as Mid-game phase (Normal game), Pre-end game phase and End game phase [3]. The mid-game phase is the duration of the game from the beginning until nine or fewer tiles are left in the bag [3]. The pre-end game phase begins when we have a total of 16 unseen tiles which means seven tiles the opponent's rack and nine tiles in the tile bag. The pre-end game is comparatively slower. The final game phase is the end-game phase and commences when no tiles are left in the tile bag or for an empty tile bag [3]. Scrabble endgames are vital and determine the fate of an opponent. It can completely change the game and the leading player. Hence they need special strategies to resolve the endgames. End-games are mostly perfect information games. However, humans cannot win it using a highest scoring greedy approach. Instead, an intelligent player must deceive the

opponent with tricky tiles. Computer game engines are noteworthy in end-game situations because of its exceptional speed and advanced simulation heuristics.

### 3.7. Board Evaluation:

Maven tries to avoid generating hot spots for the opponent. It is an essential strategy in end-games. However, in mid-games and pre-end games sacrificial of few points for a hot spot does not concern Maven lest it is a grand spot such as a Triple Word premium board square. Moreover, when the game commences, it is better to consider the third heuristics explained in 3.5, i.e., Maven must open the board with a few number of tiles. This heuristic provides an opportunity to play in a controlled manner and not letting an opponent to lead in the game.

### 3.8. Look-ahead and Simulation:

Maven employs a three-ply look-ahead to search and analyze the future opponent playable moves. According to the game theory, a ply implies a player's turn in a two-player game. A move is equivalent to two turns. We implemented a three-ply look-ahead in this project as follows:

Ply 1: Place the best scoring candidate word for a given rack and board state

Ply 2: Find the opponent's possible candidate word and resultant score if that word is placed

Ply 3: Evaluate the current state and find the best word to block that move in an endgame.

A deeper ply yields different results. However, in our case, we do not need a ply deeper than three-ply, especially during endgames. The reason is, as end-games are complete information states, evaluation of possible opponent word formation for each rack and board tiles, horizontally or vertically, can slow-down the program. Therefore, we can say that a major limitation of simulation is that they are lengthy and time-consuming. Moreover, our project is built on Java whereas original Maven or Quackle is written in traditional C and C++. Hence the execution speeds are comparatively slower than its original program speed.

# 4. QUACKLE AI

Quackle is an open source Scrabble AI tool originally written in C++, a competitor of Maven AI. We implemented the underlying heuristics and components of Quackle in our project from scratch in Java. Quackle is similar to Maven in heuristic applications, but it has advanced tool for analysis. Quackle has two critical components called kibitzer and simulation engine. This section will overview the workflow, powerful features of Quackle and presents the strengths and limitation of the AI.

## 4.1. Quackle Schematic Diagram:

A Quackle engine uses a program module called kibitzer that applies a different kind of static evaluation function than Maven to find the most promising candidate in the game. Figure 5 shows the Quackle flowchart.

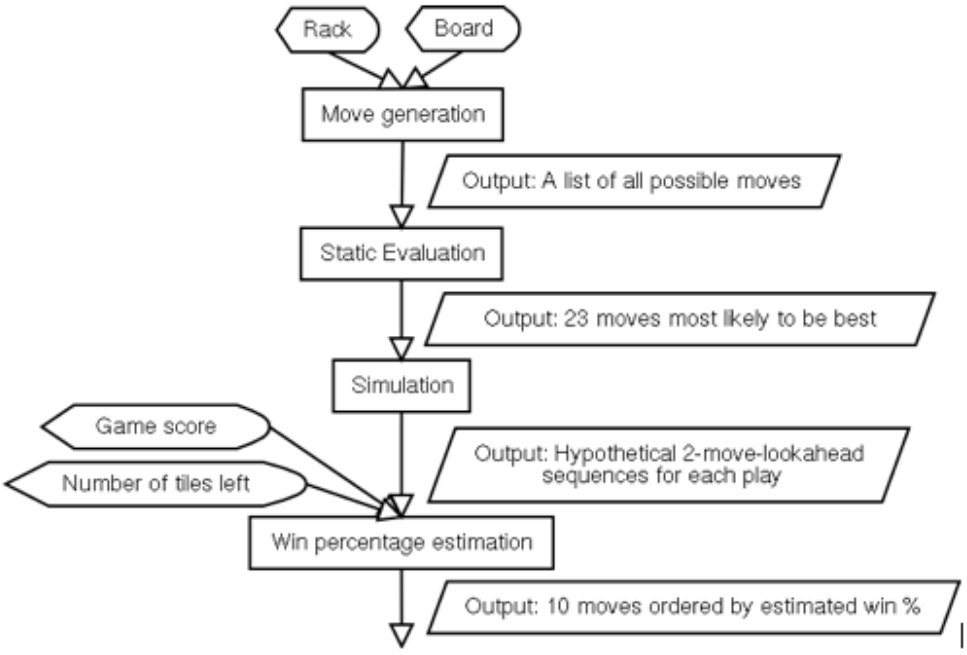



**Figure 5. Quackle flowchart [4]**

## 4.2. Quackle Move Generation:

In this block, we generate a set of candidate moves from our input rack and board. Quackle uses a Generalized Direct Acyclic Graph (GADDAG) data structure for move generation [4]. Generalized Direct Acyclic Graph (GADDAG) algorithm, developed by Steven Gordon in 1994 [8] surpassed the shortcomings of DAWG algorithm with a tradeoff. GADDAG is comparatively twice as fast as DAWG, but GADDAG is five times larger than DAWG. Consequently, GADDAG is costly as it consumes a large amount of memory. Due to its high computational cost, it is not desirable to use GADDAG in our Scrabble computer player. We used a trie and Array List data structure for storing the entire dictionary. However, the time consumption for trie was higher in an endgame simulation. So we used a trimmed dictionary of small size and stored it in an Array List.

## 4.3. Static Evaluation Function:

The candidate moves from the move generator are given as input to this block. Once it receives a collection of candidate moves, the static evaluation function calculates a rack leave value for each of the promising candidate words in the collection. To calculate the leave value, we first need to identify the letters remained on the rack after playing the corresponding word. In our project, we call the function *calcStaticEvalFunc()* to calculate the scoreLeave value which is the sum of move score and estimated leave value called leaveEstimate. Subsequently, the function returns the 23 highest scoring plays. To make this block run quicker, we used a precomputed database to lookup the leave estimate value for each of the letters.

For instance, suppose our rack holds AISROCS, and Quackle's move generator first creates a list of possible candidates move such as {OS, IS, SIC, SO, SICS, OR, AS, …} from this rack. Afterwards, Quackle applies the static evaluation function for each of these values in the candidate list. Consider the first word OS, if we play this word on board the rack leave will become AISRC. So we calculate the leave value for the rack leave, and it turns out to say 2.5. Hence, OS scores 2 points on the board as move score and 2.5 as its leave value. Then the static evaluation leave value will be the sum of move score and leave value which is equal to 4.5. This static evaluation function is then iteratively applied for each word in the candidate list and returns the top 23 words based on the total score.

We also apply some heuristics here. An empty rack-leave indicates a Bingo move. Our Quackle AI award higher leave values to the bingo moves so that they have good chances of playing for the current turn.

### 4.4. Quackle Simulation:

Quackle runs hundreds of simulation for each candidate move. The official Quackle player even has an option to extend it to 1000 times. However, it will be time-consuming to run 1000 times compared to 100 times. Moreover, it uses a three-ply look ahead to identify the best promising word that must be played against the opponent. For instance, consider the above example we discussed in Section 4.3. Let OS be one of the highest scored candidate move returned. We need to analyze if OS makes a good candidate move while running simulation. The three-ply look-ahead follows the steps below:

Ply1: Current player plays OS having a score leave value 4.5 and rack leave AISRC.

Ply 2: A random rack is created for opponent and it plays word JEWEL scoring 20 points as score leave and leaves PM in the rack.

Ply 3: Now, current player arbitrarily adds new shuffled tiles, say PT into his rack and play the best word CA(R)T for score 5.5 leaving ISRP in the rack. R is the anchor letter on the board.

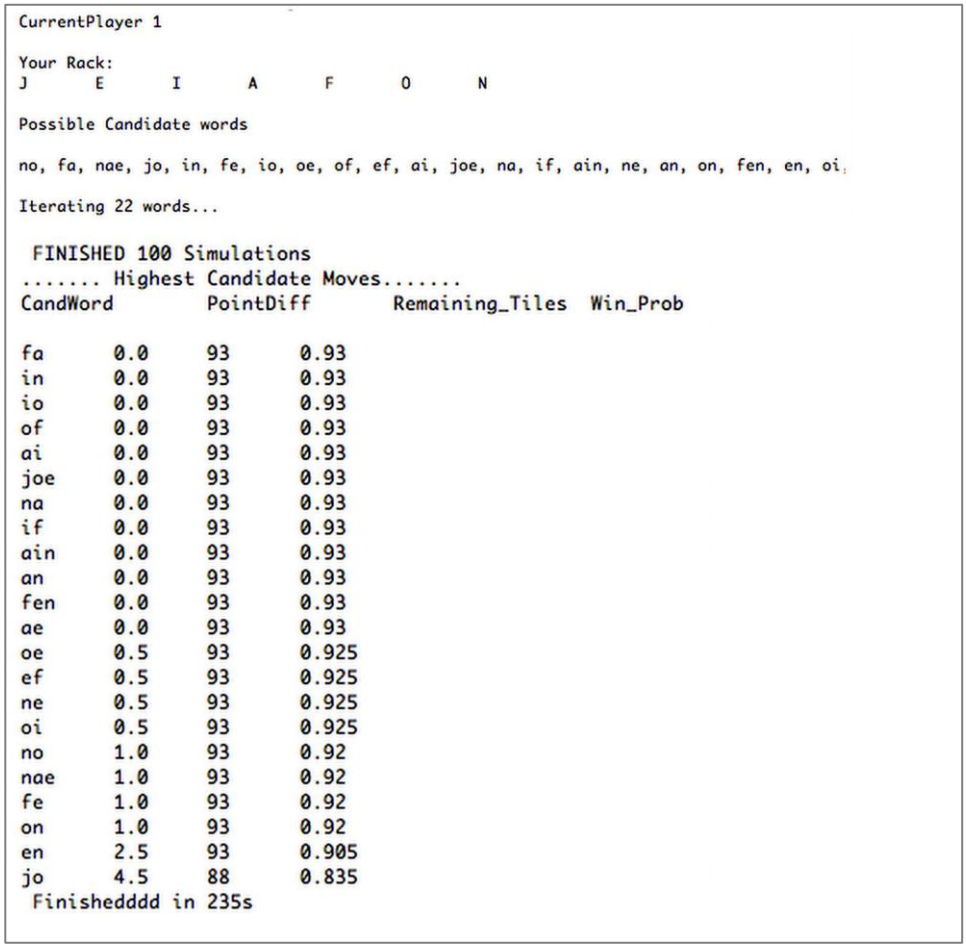
Eventually, the point differential or equity score is calculated for each word as Our Total Score – Opponent's Score + Residual rack leave value estimate [3]. The point differential can be positive or negative values. A positive point differential implies the score lead and a negative value refers how much the player is behind.

**4.5. Win Probability Percentage Estimation:**

Once the simulation identifies the three-ply look ahead moves, this block calculates the winning probability based on the point differential and the remaining tile count for each candidate move. After computing the win percentage, the candidate move list is sorted in the descending order of winning probability percentage. If two words are found with the same probability, we break the tie by selecting the word with lowest rack leave value.

Figure 6 represents an example move of Quackle where player's rack consists of letters JEIAFON, and the kibitzer founded 22 possible words that can be formed with this list. Each candidate word in the list is simulated 100 times, and we calculate an averaged point differential and its corresponding win probability. Thus a total of 2200 simulations

happened for all the 22 candidate words in the list. The time taken for 2200 simulations is 235 secs which we consider reasonable. From the figure, we can realize that the candidate words like {FA, IN, IO, OF, AI …} has a point differential of 0.0 when the remaining tiles in the tile bag are 93. From the results in figure 6, we can realize that words with low point differential have a high winning probability.



```
CurrentPlayer 1

Your Rack:
J      E      I      A      F      O      N

Possible Candidate words

no, fa, nae, jo, in, fe, io, oe, of, ef, ai, joe, na, if, ain, ne, an, on, fen, en, oi,

Iterating 22 words...

 FINISHED 100 Simulations
..... Highest Candidate Moves.....
CandWord        PointDiff       Remaining_Tiles  Win_Prob

fa      0.0     93      0.93
in      0.0     93      0.93
io      0.0     93      0.93
of      0.0     93      0.93
ai      0.0     93      0.93
joe     0.0     93      0.93
na      0.0     93      0.93
if      0.0     93      0.93
ain     0.0     93      0.93
an      0.0     93      0.93
fen     0.0     93      0.93
ae      0.0     93      0.93
oe      0.5     93      0.925
ef      0.5     93      0.925
ne      0.5     93      0.925
oi      0.5     93      0.925
no      1.0     93      0.92
nae     1.0     93      0.92
fe      1.0     93      0.92
on      1.0     93      0.92
en      2.5     93      0.905
jo      4.5     88      0.835
 Finishedddd in 235s
```

**Figure 6. Quackle simulation example**

# 5. END GAME STRATEGIES

Our project implements three variants of end-game strategies for Maven AI and a general win percentage probability approach for Quackle AI. Figure 7 shown below illustrates the three kinds of endgame strategies used in the project. As endgames are essential determining the success of the game, we realized that having a right approach can let a losing player winner. Human players lack this skill as endgames are complicated and in tournaments or casual games, the player must make a move within the stipulated time limit. Towards endgame, human players usually play carelessly and hurriedly and eventually lose the game. Nevertheless, computer players play in a controlled manner by applying specific endgame strategies. These AIs use a three-ply simulation to evaluate the future board moves and tries to trap the opponent, by expanding the winning margin. Chapter 5 narrates the end-game strategies used in this project.

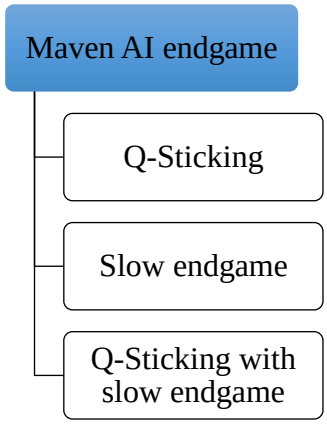



**Figure 7. Three variants of Maven end-game AI**

## 5.1. Q - Sticking:

Q-Sticking is an endgame scenario where the opponent is stuck with an unplayable tile like Q, or V. Maven Q-Sticking AI block the opponent's hot spots on board such that

they cannot get rid of tiles like Q or V which are hard to play due to its fewer word combinations. Once it blocks opponent's move, this AI carries out the favorite straightforward strategy of Maven used in mid-game scenarios where the highest scoring word is placed to gain maximum reward. If an opponent cannot make a play, it gives a pass, and the player scores an extra 20 points for each pass. Thus, the opponent penalty scores boost the final aggregate score of this AI. An AI which plays a straightforward strategy to score maximum points in Q-stuck scenarios is called No Q-Sticking AI.

### 5.2. Slow-endgame Strategy:

In this strategy, Maven traps the opponent player by stretching the game and plays one tile at the moment. The primary intention of this approach is to gain maximum score by slowing down the game using low-scoring tiles. For example, consider the possible candidate Maven AI word list as {HOPERS (score = 11), EPOCH (score=9), EH (score = 5), OH (score =5), OHS (score = 6), TOD (score =4), OI (score=2)}. Suppose there is an anchor letter O on board that can benefit from a premium bonus (*2), and Maven has the choice of playing the highest scoring word HOPERS on board. Nonetheless, the slow-endgame AI plays the OH vertically and for the first turn by placing H next to O which scores 6 points. On the subsequent turn, Maven extends the game by placing S next to OH to make the word OHS that scores 7 points.

### 5.3. Q-Sticking with Slow-end game Strategy:

For instance, the possible candidate words of Maven be {FAITH (score = 10), ENERGETIC (score=21), IT (score = 4), BIT (score = 3), AI (score =2)} and Quackle

has a hot spot on board to place word QI. In this case, instead of playing the highest scoring word FAITH with 'I' as anchor letter, Maven plays the word IT on board on first chance taking away the opponent's opportunity because if we place FAITH on the board we can only score 10 and the rack leave becomes zero. However, if it plays slowly, then Maven can gain high scores in the subsequent plays by retaining a good rack leave. In the next turn, Maven plays word BIT by placing it on premium word square and scores 9. So the combined score in both the plays will be 4+10= 14 which is greater than 10 (score of FAITH). Figure 8 shown below represents a random endgame scenario that we simulated in our program. The current number of tiles equals zero implies that our tile bag is empty and the game has entered the end-game phase.

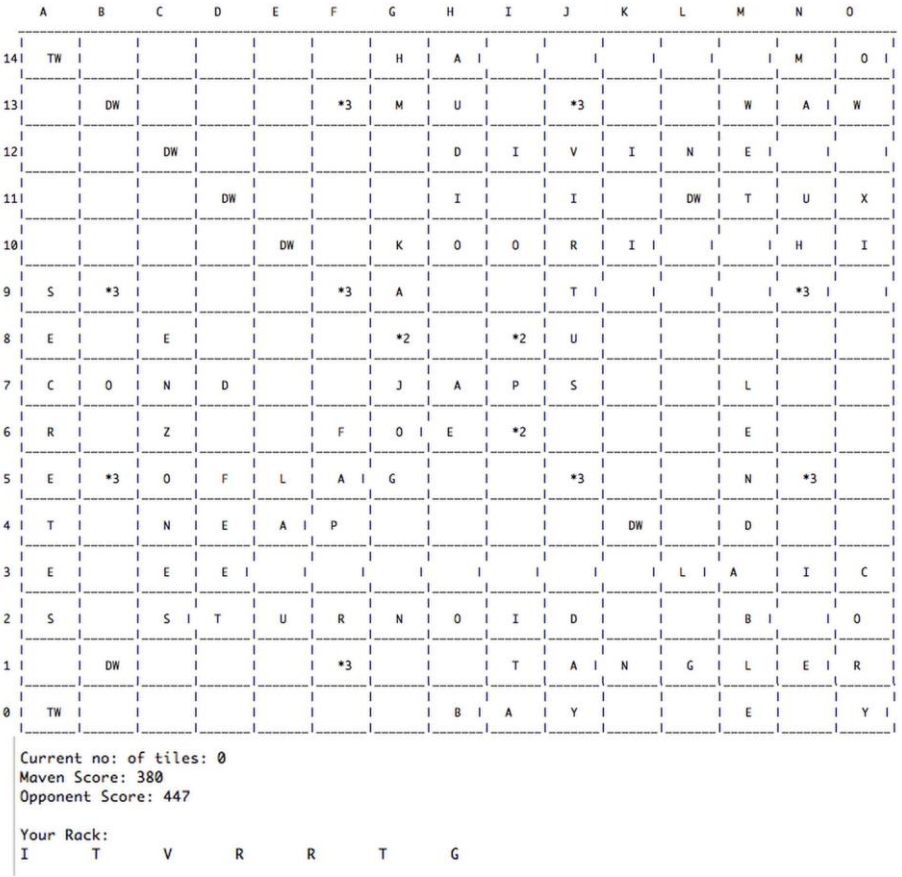



**Figure 8. Q-Sticking end game scenario**

## 6. DESIGN AND WORKFLOW OF THE PROJECT

We now present the overall design and work-flow of major classes of this project as shown below in Figure 8. First, we see the relationship between ComputerPlayer and Player classes. Player is an abstract class, and the two ComputerPlayer classes that are sub-classed from the abstract Player class implements the abstract *play()* method. ComputerPlayer class includes the implementation of Maven AI, and ComputerPlayer2 refers to Quackle AI.

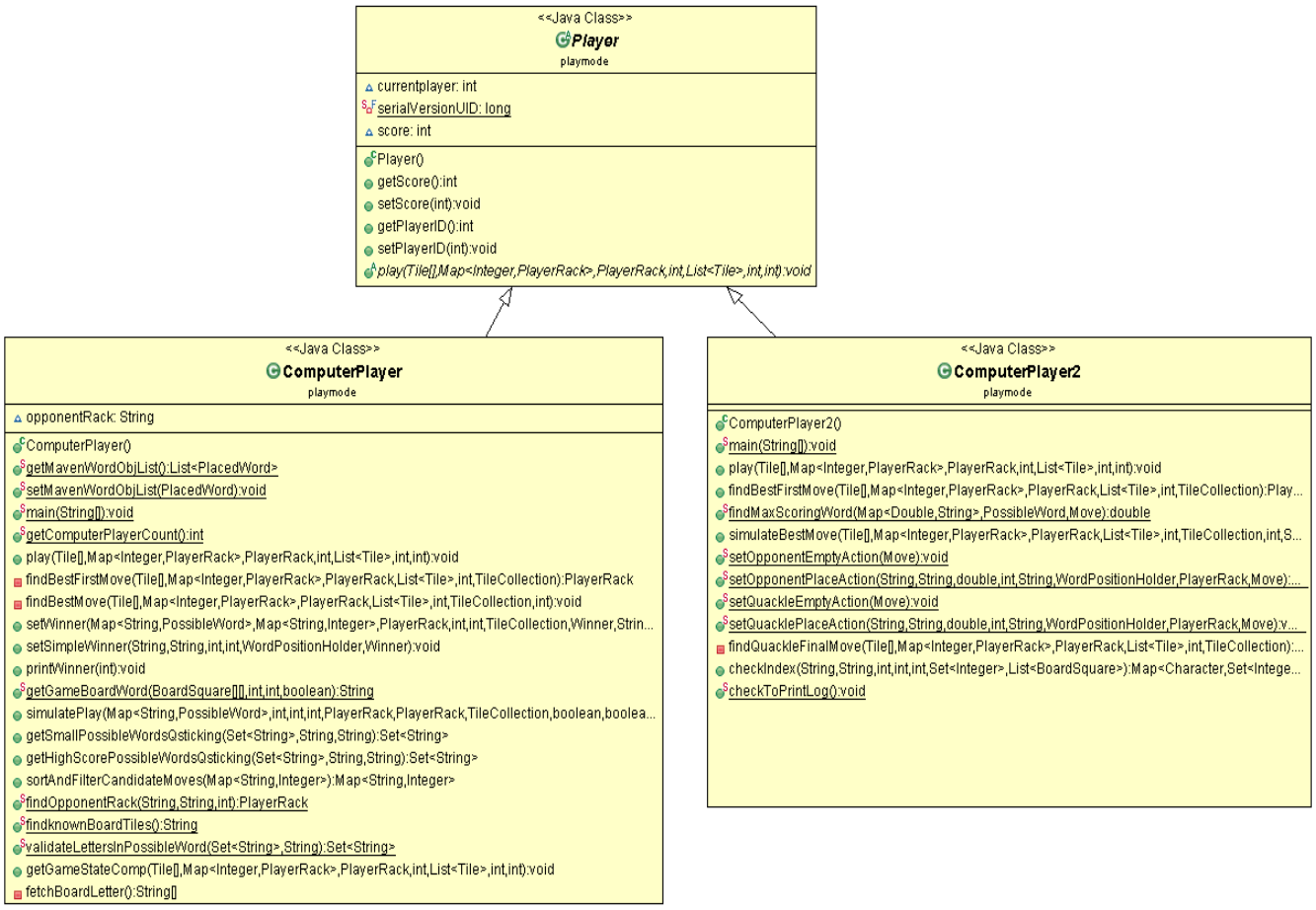



**Figure 9. Class diagram of Player Class and Computer Players**

Each *play()* method in ComputerPlayer and ComputerPlayer2 class contains call to other functions such as *findBestFirstMove()* and *findBestMove()*. The former one finds

the best first move for the first play, and the latter one finds the best move in the subsequent plays other than the first play. Figure 9 presented below shows the flowchart of the game flow.

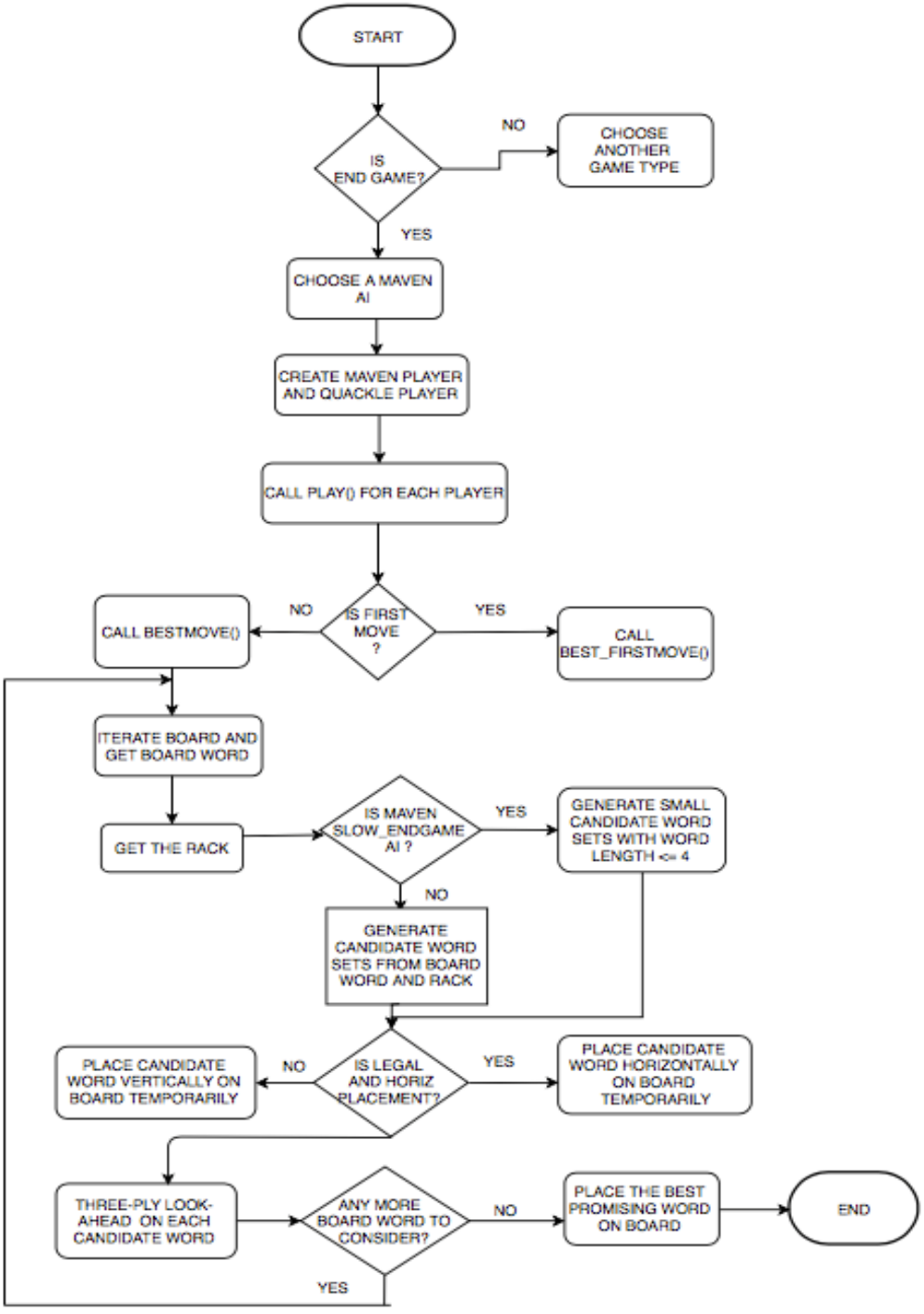



**Figure 10. Overall workflow of Scrabble game AI**

# 7. EXPERIMENTS AND RESULTS

This chapter discusses the experiments and results conducted in our project. The goal of this project is to let two AI's play against each other as opponents and determine the performance of AI's based on the quality of moves, winning criteria and time consumption. Since we have three AIs of Maven built on different endgame strategies like Q-Sticking, slow-endgames, Q-Sticking combined with slow-endgame, and another AI based on the Quackle AI and Maven No-Q-Sticking; we let each variant of Maven AI to fight against Quackle.

Maven and Quackle were originally written in C and C++ respectively for faster speed. However, we rewrote their heuristics in Java due to our comfort with the language. We focused on a two-player game and experimented with pairs of four AIs as follows:

a) Maven Q-Sticking AI versus Quackle for Endgame

b) Maven slow-end game AI versus Quackle for Endgame

c) Maven Q-Sticking with slow-end game AI versus Quackle for Endgame

Maven Q-Sticking versus Maven No-Sticking for Endgame

All these experiments are executed on a Mac Machine having 16 GB RAM-quad-core processor running Java version 7. We ran ten randomly generated Scrabble endgame for each experiment which was taken from real-world end-game scenarios used in tournaments as in [9, 11, 12] and also tested the Q-Sticking scenarios generated from a normal game after running the program. We chose these particular endgames because we needed to analyze the likely winning moves of our program during a Q-Sticking endgame

situation exhibited by these games. As the next step, these real-world endgame states were loaded into our project by considering the current game board state, player rack, player score, tile bag and score lead.

## 7.1. Experiment 1: Maven Q-Sticking AI versus Quackle for Endgame:

This experiment was conducted on 10 randomly generated Scrabble endgames between the two AI players, Maven-Q-Sticking AI and Quackle. Maven Q-Sticking implies Maven's strategy when the opponent Quackle is stuck with unplayable tiles like Q or V. To do this experiment, we created Maven and Quackle AIs in two separate classes and made function calls to *firstmove()* of game open AI and *bestmove()* of both AIs for subsequent plays alternatively until a winner is detected. We considered Maven for first turn open in these experiments

.

In our hypothesis, we predicted that Maven Q-Sticking AI will perform better than Quackle as Maven AI can block the hot spots of Quackle when Quackle is stuck with Q and play with a greedy approach. The results of the experiment are shown in Figure 11.

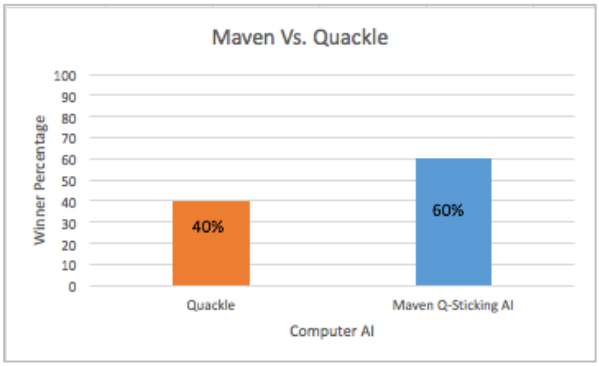



**Figure 11. Experiments between Maven Q-Sticking AI and Quackle AI**

To determine the average mean score and standard deviation (SD), we noted down the distribution of game scores of both the players on each iteration and calculated its mean after 10 iterations. Standard deviation was then calculated to measure the point spread.

**Table 1. Average game scores of Maven Q-Sticking AI and Quackle**

| Player AI | Mean Score + Standard Deviation |
|---|---|
| Maven Q-Sticking AI | 473.2 ± 70 |
| Quackle | 450.4 ± 59.2 |

### 7.1.1 Observations:

From the test results, we conclude that Maven only wins 60% of the time and Quackle wins 40%, and the standard deviation of Maven is quiet higher than Quackle. At the beginning of the end-game situation, Quackle leads the game. However, after calculating the opponent's rack, we found that Quackle is stuck with an unplayable tile. So the Maven AI applied its Q-Sticking strategy and scored by blocking the opponent hot spots using its high scoring words and became the winner 60% of the times. Thus, we realized that a leading Quackle could be defeated by Maven AI even if it is behind the points.

We also observed that a Maven win is influenced by factors like the score difference, points on the rack tile, number of rack tiles, and the type of unplayable tile of the opponent. In the Iterations, 1, 2, and 5 Maven had high scoring tiles and could place those tiles by blocking the premium bonus spots of Quackle. Thus it was easily able to defeat Quackle because whenever Quackle gave a pass, Maven received extra 20 points

for each pass. In the Iterations 3, 6 ,7 the score difference during the start of endgame phase was less and Maven could easily outscore Quackle due to less point differential. Thus above-experiment confirmed our hypothesis that even if Quackle out-scores Maven, Maven has still room to win the game.

Consider a real end-game scenario where the current score of Quackle and Maven is 480 and 360 respectively. Maven has seven tiles on the rack. However, the competitor Quackle is stuck with a Q. In this case, Maven blocks and beat the opponent as it has suitable tiles that can form a right legal word with the tiles in the hot spot. Most of the times, this scenario turns true however if Maven has less number of low scoring tiles, it cannot beat the opponent even if the blocking occurs. That is because Maven Q-Sticking AI places the highest scoring word on the hot spot and if the highest scoring word constitutes all the tiles in the rack and the total score is not greater than Quackle score, Maven may fail. Conversely, the highest scoring word does not mean the longest word in the rack. A high scored two letter word on premium squared board squares is one such instance. Figure 10 referred below shows that candidate word-set in the format <word, score, row, col>. In this case, word ‘IT’ has score 25 and ‘GHAT’ only scored 24 as IT is placed on a premium word score.

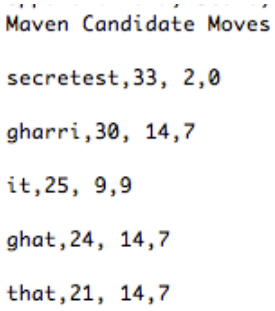



**Figure 12. Candidate moves of Maven**

## 7.2. Experiment 2: Maven slow-endgame AI versus Quackle for Endgame:

This experiment was conducted on ten randomly generated Scrabble endgames between Maven-slow-endgame AI and Quackle. To do this experiment, we created these two AIs in two separate classes and called the functions alternatively until a winner is detected. In this strategy, Maven slows down the endgame by playing low scoring one tile at the moment and takes away an opponent's opportunity to increase the total reward.

In our hypothesis, we predicted that Maven will outscore its Quackle opponent because it draws the tiles one-by-one at a time to gain high reward and maximum point spread through penalty score. Figure 13 referred below shows the experiment results.

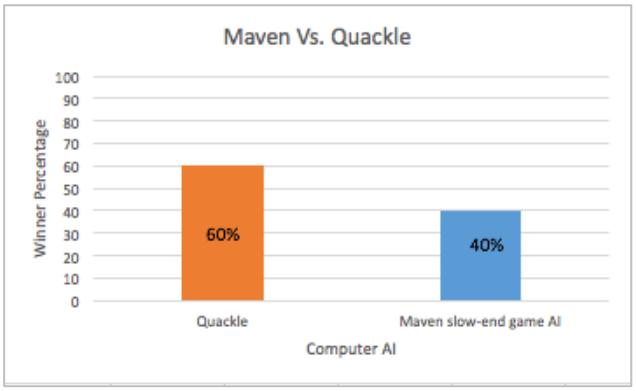



**Figure 13. Experiments between Maven slow-endgame AI and Quackle**

The same procedure explained in section 7.1 was continued to determine the mean score and standard deviation of the game.

**Table 2. Average game scores of Maven slow-endgame AI and Quackle**

| Player AI | Mean Score + Standard Deviation |
|---|---|
| Maven slow-endgame AI | 461.9 ± 80 |
| Quackle | 442.6 ± 60 |

### 7.2.1 Observations:

From the above results, we learned that Quackle can still defeat Maven even if Maven plays out first or lead in the beginning of an endgame. Quackle won 60% of the time where Maven only won 40%. Table 2 shown above illustrates that the average scores of Maven is higher as compared to Quackle. Thus Maven's data points are widely spread around the mean. Quackle win-influence factors seen in the game are a high-score lead for Quackle, Maven's less value tile rack, Quackle's rack of high point unplayable tiles when Maven tiles finished soon (e.g.: An unplayable tile V in the Quackle rack could only earn 4 points whereas Q in the rack earned 10 points and Quackle won).

For instance, in Iteration 2, Maven's score was 214, and Quackle's score was 343. Quackle was stuck with a Q. Maven played out all his two remaining tiles, and at the end of the game, as a penalty, Maven added the maximum tile value '10' (value of Q) to his score as well as subtracted the same score from Quackle. Still, Quackle won the game. The reason we analyzed is that there was a 129-point difference between the players during the beginning of the endgame situation. After play out, Maven's score was only 230 whereas Quackle's score was greater than that. Moreover, there were only two tiles for Maven to play and one tile for Quackle. If there were a full rack tiles in Maven's player rack, it could still win the game. However, 40% of the time, Maven had a high score full rack of tiles, and it won the game. In Iterations 7 and 8, the scoring lead for Quackle was few points (<10). Thus Maven could quickly take over Quackle. Hence, we realized that this AI is not good for Maven and this experiment denied our hypotheses.

## 7.3. Experiment 3: Maven Q-Sticking slow-endgame AI versus Quackle for Endgame:

This experiment was conducted on randomly generated Scrabble endgames between Maven Q-Sticking slow-endgame AI and Quackle. Maven Q-Sticking slow-endgame AI will always win the game as it can block any single opportunity of the opponent even if it is a hot spot or not and at the same time drag the game by placing tiles one by one with a goal of increasing the score. The above mentioned is a good strategy to trap the opponent with unplayable tiles and earn an extra score of 20 for each pass. Instead of blocking with the highest valued tile as seen in Q-Sticking, this strategy blocks the hot spots with low-scoring tiles.

In our hypothesis, we predicted that Q-Sticking slow-endgame AI will outscore its Quackle opponent because it blocks all the opponent moves and draws the tiles one-by-one at a time to gain high reward and maximum point spread through penalty score. To perform this experiment, we built Maven and Quackle AIs in two separate classes and made function calls as discussed in Experiment 7.1. We considered Maven for first turn open 70% of the times in the experiments. Figure 14 shows the experiment results.

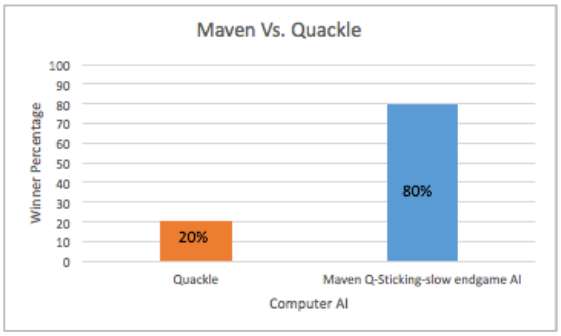



**Figure 14. Experiments between Maven Q-Sticking slow-end game AI and Quackle**

Again we applied same procedure explained in section 7.1 was to determine the mean score and standard deviation of the game.

**Table 3. Average game scores of Maven Q-Sticking slow-end game AI and Quackle**

| Player AI | Mean Score + Standard Deviation |
|---|---|
| Maven Q-Sticking slow-end game AI | 472.2 ±86.79 |
| Quackle | 417.9 ± 54.63 |

### 7.3.1 Observations:

After running the experiments, we realized that 80% of times Maven defeated Quackle. In Iterations 1,2,3,4 even if the score variation was in the range 65 to 100 points, Maven blocked all the spots, hot spots or not, and played the low-scoring tiles one by one. Dragging the game allowed Maven to earn a bonus of 20 points on each pass of the opponent. Also, Iterations 6,7,8 and 10 Maven had also got the chance to place the majority of the low-scoring tiles in premium squares, gaining extra bonuses. Maven lose in 10% of the times as its low-scoring tiles played out fast due to less quantity and Quackle had a high score lead. Thus above-experiment confirmed our hypothesis that even if Quackle out-scores Maven, Maven can win the game.

## 7.4. Experiment 4: Maven Q-Sticking slow-endgame AI versus No-Q-Sticking AI:

This experiment was conducted on randomly generated Scrabble endgames between Maven Q-Sticking slow-endgame AI and No-Q-Sticking AI that plays a straightforward strategy. The later AI tries to score maximum points by placing the highest scoring word on board. Maven applies this strategy mostly in mid-game phase,

but we included it in endgame scenarios too.

In our hypothesis, we predicted that Q-Sticking slow-endgame AI will outscore No-Q-Sticking AI opponent because it blocks all the opponent moves and draws the tiles one-by-one at a time to gain high reward and maximum point spread through penalty score. Figure 15 shows the experiment results of Maven Q-Sticking slow-endgame AI versus No Q-Sticking AI.

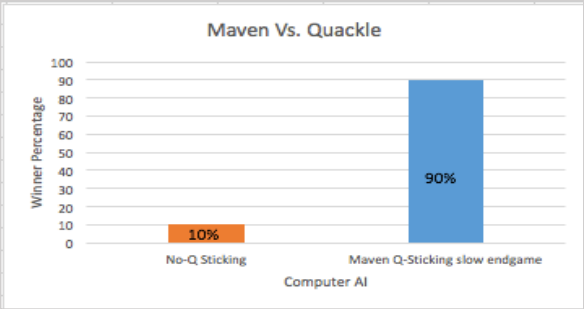



**Figure 15. Experiments between Maven Q-Sticking slow-end game AI and No-Q-Sticking**

We calculated the average scores of both the AIs to find the point spread over the mean over 10 games and found that Maven has higher mean score and deviation than the other one.

**Table 4. Average game scores of Maven Q-Sticking slow-end game AI and No Q-Sticking**

| Player AI | Mean Score + Standard Deviation |
|---|---|
| Maven Q-Sticking slow-end game AI | 479.9 ± 75 |
| No-Q Sticking AI Score | 415.1 ± 45 |

### 7.4.1 Observations:

After running the experiments on a Q stuck opponent leading scenarios, we realized that 90% of times Maven Q-Sticking defeated No-Q-Sticking AI because No-Q-Sticking does not block any opponent moves rather it plays the highest scoring move. This time Maven-Q-Sticking places its unplayable tiles in few moves, and thus it plays out all the tiles and wins the game. We realized that blocking is necessary to trap the opponent in an endgame scenario. Thus above-experiment confirmed our hypothesis that even if Quackle out-scores Maven, Maven can win the game.

# 8. CONCLUSIONS AND FUTURE WORK

In this project, we developed five different AIs having different move generation and simulation techniques. Maven AI with Q-Sticking slow-end game strategy was expected to outscore the Quackle AI, and it proved to be the best computer player AI in our project. It can block the hot spots as well as earn bonus points by trapping an opponent using the slow- endgame strategy. Maven Q-Sticking AI also had a relevant move generation heuristic and outscored the Quackle opponent. Still, the heuristics were not as favorable as Maven Q-Sticking slow-endgame AI. Maven slow-end game AI did not perform well for endgames, and the AI lost to Quackle many times. Quackle AI could only outscore Maven slow-end game AI. Finally, No Q-Sticking AI was found to be a promising AI during mid-games. However, this AI did not perform well during endgame scenarios.

The major drawback of our project was the execution time taken by three-ply look-ahead simulation. For example, our project consumes 100-255 seconds to finish 100 simulations of Quackle whereas originally the Quackle championship player only takes about 19-50 seconds for a move generation. In case of Maven AI, the execution time of a single move generation was about 0-2000 milliseconds, however, originally Maven AI consumed only 0-30 milliseconds. The time consumption of original Maven AI was computed using a stopwatch.

As a future enhancement, we will reduce this execution time of the simulation engine and will implement an interactive graphical UI for the project. Furthermore, we

realized that Quackle is not just a Scrabble play engine, it is also an analysis tool that can simulate up to 1000 times and gives an option to the player to look up to 6 plies deep which could be time-consuming. So we will try to incorporate these features into our AI in the future.

# 9. BIBLIOGRAPHY


[1] S. Russell and P. Norvig, "Adversarial search," in *Artificial Intelligence: A Modern Approach*, 3rd ed. New Jersey: Pearson, 2010, Ch. 5, pp. 161-189.

[2] F. Di Maria and A. Strade, "An artificial intelligence that plays for competitive scrabble," in *Proc. AI*IA Workshop*, Bologna, Italy, 2012, Vol. 860, pp. 98-103. [Online]. Available: http://ceurws.org/Vol-860/paper16.pdf

[3] B. Sheppard, "World-championship-caliber Scrabble," *Artificial Intelligence*, Vol. 134, pp. 241-275, Jan. 2002. [Online]. Available: http://ac.elscdn.com/S0004370201001667/1-s2.0-S0004370201001667-main.pdf?_tid=490191b6-2168-11e7-91f6-00000aacb35e&acdnat=1492211924_4ed8a733c08936feace8b1ad2aab9c37

[4] J.K. Brown and J. O'Laughlin, "How Quackle Plays Scrabble," 2007. [Online]. Available: http://people.csail.mit.edu/jasonkb/quackle/doc/how_quackle_plays_scrabble.html

[5] A. W. Appel and G. J. Jacobson, "The world's fastest Scrabble program," *Communications of the ACM*, pp. 572-576, May. 1988. [Online]. Available: http://www.gtoal.com/wordgames/jacobson+appel/aj.pdf

[6] C. Josephson and R. Greene, "CS221 Fall 2016 project final report: Scrabble AI," 2016. [Online]. Available: https://web.stanford.edu/class/cs221/2017/restricted/p-final/cajoseph/final.pdf

[7] N. Gupta *et. al*, "AI Agent to play Scrabble," 2017, [Online]. Available: https://web.stanford.edu/class/cs221/2017/restricted/p-final/rsbauer/final.pdf

[8] S.A. Gordon, "A faster Scrabble move generation algorithm," *Software: Practice and Experience*, vol. 24, no. 2, pp. 219- 232, Feb. 1994. [Online]. Available: http://ericsink.com/downloads/faster-scrabble-gordon.pdf

[9] "Endgame finesse," Australian Scrabble® Players Association, Mar. 2005. [Online]. Available: http://www.scrabble.org.au/strategy/endgame.htm

[10] J. Mandziuk, "Maven and Quackle-The top level machine scrabble players," in *Knowledge-Free and Learning-Based Methods in Intelligent Game Playing*, Chennai, India: Springer, 2010, Ch. 4, pp. 45 - 47.

[11] "Breaking the game," 2014. [Online]. Available: http://www.breakingthegame.net/strategy

[12] J. Edley and J.Williams, "Everything Scrabble: Third Edition," 2009, pp. 255- 315. [Online]. Available:https://books.google.com/books?id=Uml_kpOO64gC&lpg=PP1&pg=PR5#v=onepage&q=end%20game&f=false

[13] J. Meyers, "How To Master SCRABBLE & Win Every Game," 2015. [Online]. Available: https://scrabble.wonderhowto.com/how-to/master-scrabble-win-every-game-0115054/

[14] M. Pekker, “Scrabble: Brain medicine to fight Alzheimer’s,” February, 2013. [Online]. Available: http://alzheimers-review.blogspot.com/2013/02/scrabble-word-game-helps-alzheimers.html