SCRABBLE ARTIFICIAL INTELLIGENCE GAME

CS 297 Report

Presented to

Dr. Chris Pollett

Department of Computer Science

San Jose State University

In Partial Fulfillment

Of the Requirements for the Class

CS 297

By

Priyatha Joji Abraham

May 2017

TABLE OF CONTENTS

I. Introduction

Artificial Intelligence (AI) is a collection of techniques used to simulate human behaviors using machines. Since the 1950's, Artificial Intelligence (AI) has played a significant role in the game industry and games [1]. AI strives to build intelligent agents that can perceive and act rationally to accomplish one's goal. It let the system or machine think, act and solve problems like a human. 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 one such crossword board game which is widely popular for its game strategy and ability to build vocabulary.

To win this two to four player game, the player has to comprehend the whole board and tile distribution as well as apply some statistical probabilities to block the opponent's moves. In scrabble, each tile is marked with a predefined letter, ranging from A to Z, and values, ranging from zero to ten. Once the game begins, players place their square shaped tiles on a 15x15 grid of game board to build words and score points based upon the words formed [2].

The main challenge of this game 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. Another challenge faced by the current computational approaches such as DAWG (Direct Acyclic Word Graph) is the consumption of time to calculate the consequences of the exact next move [2]. The current best AI Scrabble computer player called MAVEN, created by Brian Sheppard in 2002 uses a DAWG data-structure [5]. It stores an entire lexicon in a trie in a prefix form so that the structure consumes only minimum memory or space as compared to search trees. Moreover, we need to explore efficient strategies to generate human - like judgments that can search quickly through the search space of the board of tiles using less space, time and limits the opportunities of opponents to earn a higher score.

The primary objective of this project is to build a human-AI player game that can solve the problem of generation of intelligent behaviors that makes a machine not just think like a human but even surpass human. To achieve this goal, the machine should quickly generate possible moves for an opponent and itself for a given rack and current board state. Besides, the computer machine must be modeled to apply efficient strategies wisely on each turn. An approach used in this project to resolve the problem of generation of possible moves is Monte Carlo Simulation that utilizes a Monte Carlo Tree Search (MCTS) to create quick optimal decision making using the methods of probability [3].

The following are the Deliverables we have implemented in this semester to create a human-AI scrabble board game. In Deliverable 1, a simple notation system for Scrabble has been designed. For Deliverable 2, a human Scrabble player system has been developed that accepts 2-4 players. Deliverable 3 describes the retrieval of possible playable words using a trie data structure. In Deliverable 4, an AI player with limited intelligence has been implemented.

II. Deliverable 1 - Scrabble Notation System

Deliverable 1 was to design and to develop a simple Scrabble notation system. Scrabble is a classic board game played by two to four players on a 15x15 grid game board invented by Alfred. M. Butts during 1930s [5]. 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 can be used in place of any letter. These tiles are worthless and do not earn any points. A set of seven tiles are drawn randomly form the tile bag on each turn to form a rack. In this deliverable we discuss about the player notation and an ASCII board representation used to draw the game board using characters.

A. Player Notation System:

In this project, the player notation is designed to identify a human play. A set of inputs such as the direction of placement (H/V where H denotes horizontal and V denotes vertical placement), X and Y coordinates of the game board to place the desired letter and, tiles from the rack that form the word are played on each turn. Each human player is supposed to enter the following four parameters as comma separated value to run the program.

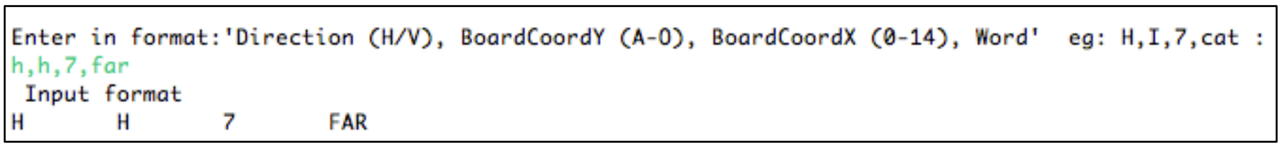
1 - Direction of play (h/H denotes horizontal direction and v/V denotes vertical direction)

2 - Board Y coordinate (A – O) or column

3 - Board X coordinate (0 – 14) or row

4 - Tiles from the rack to form a word (' _' used for blanks)

The following Fig. 1. exhibits the player input notation of a human play.

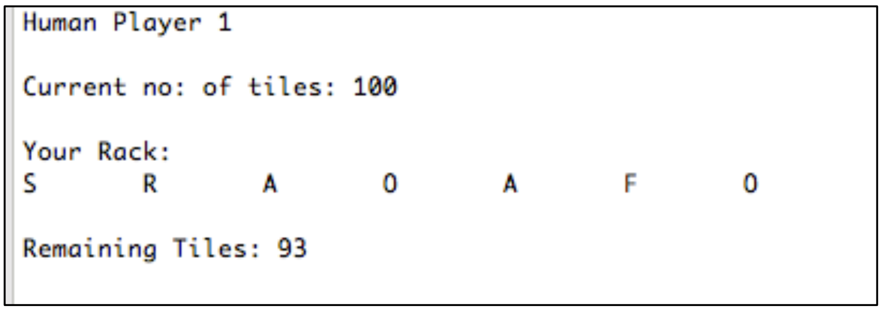


```
Enter in format:'Direction (H/V), BoardCoordY (A-O), BoardCoordX (0-14), Word'  eg: H,I,7,cat :
h,h,7,far
 Input format
H       H       7       FAR
```

Fig. 1. Player input notation.

A scrabble game does not allow diagonal plays. Hence the player can only lay the words in either horizontal or vertical direction according to the first parameter. For the second and third parameters, a player cannot exceed the row or column limits. If the limit is exceeded, then an invalid entry message is displayed to the player. In the fourth parameter, a player has to place the letter tiles from his own rack. These tiles are hooked to an existing tile of the game board in all the plays except the first play. Thus the words formed in the subsequent plays would be an extension to the words formed in the preceding play. The play gets nullified when there are no anchor letters in the given board coordinate to hook to the new word.

We also maintain the current board state, current rack state, the remaining number of tiles in the tile bag and player's rack on each turn. These are significant elements that can later turn our game of imperfect information to a perfect one. Fig 2. shows the program snippet of current state board notations. According to Fig. 2, there are 100 tiles in the tile bag initially. When seven randomly picked tiles are assigned to Player 1’s rack the remaining number of tiles become 93. The game continues until the remaining number of tiles reaches zero. This information about remaining tiles can later influence decisions made in the game such as the possible rack of tiles in opponent’s rack during end game.

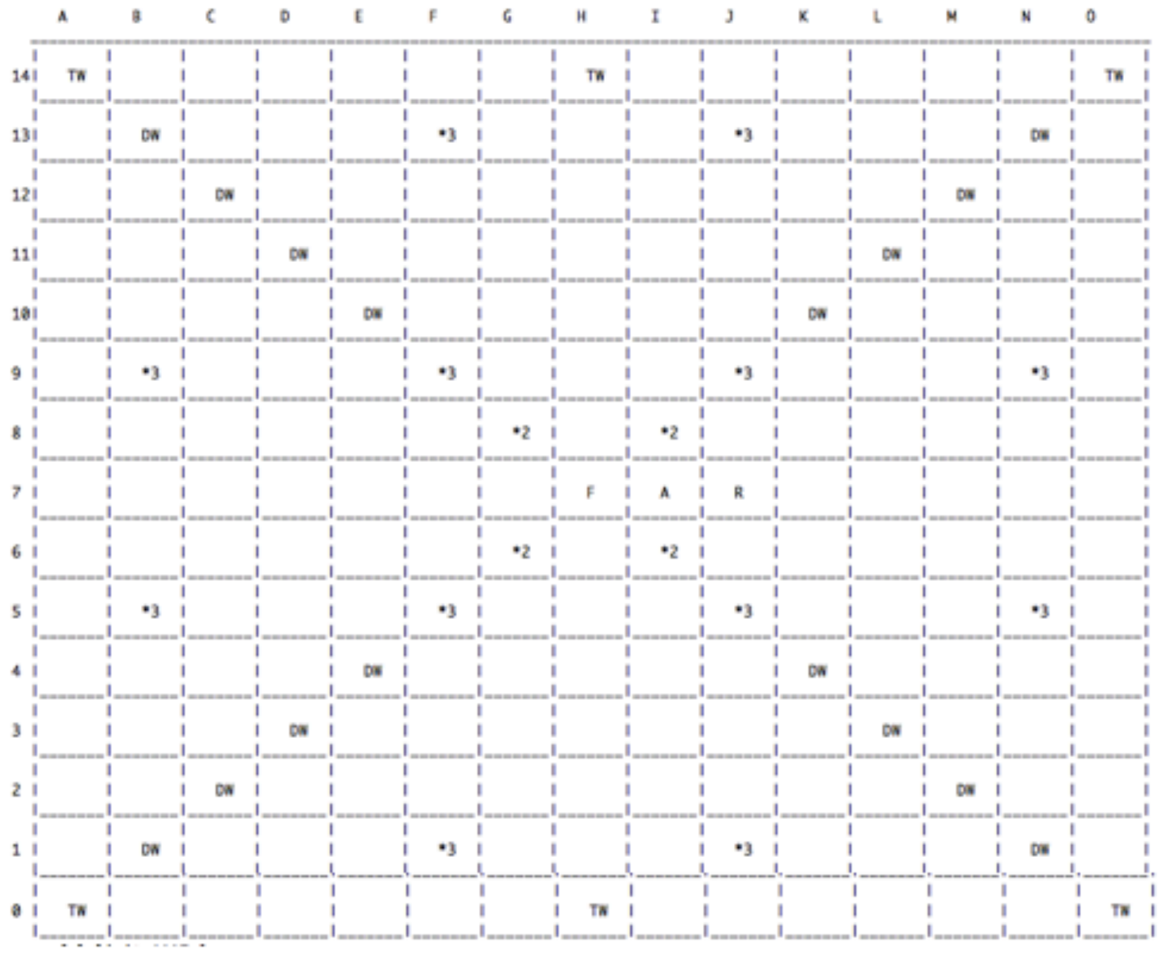


```
Human Player 1

Current no: of tiles: 100

Your Rack:
S       R       A       O       A       F       O

Remaining Tiles: 93
```

Fig. 2. Current game state and remaining tiles.

B. ASCII Board Representation:

An ASCII board representation of the scrabble board was created and is illustrated in Fig.3. As the game board is a 15x15 square matrix, rows are numbered from “0 to 14” and columns are labeled with alphabet letters “A to O”. Blank tiles in the rack are represented as ‘_’ in the program. When a player draws blank tile from the rack, the program asks to enter a replacement letter. In Scrabble, there is a concept of bonus points for game squares on the board. According to the board notation system, “*2” indicates Double Letter Bonus and “*3” indicates Triple Letter Bonus. Similarly, “DW” and “TW” represents Double Word and Triple Word Bonus points. Again, Fig. 3. shows the scrabble board notation used in this game.



```
      A      B      C      D      E      F      G      H      I      J      K      L      M      N      O
   ---------------------------------------------------------------------------------------------------------
   |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |
14 |  TW  |      |      |      |      |      |      |  TW  |      |      |      |      |      |      |  TW  |
   |______|______|______|______|______|______|______|______|______|______|______|______|______|______|______|
   |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |
13 |      |  DW  |      |      |      |  *3  |      |      |      |  *3  |      |      |      |  DW  |      |
   |______|______|______|______|______|______|______|______|______|______|______|______|______|______|______|
   |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |
12 |      |      |  DW  |      |      |      |      |      |      |      |      |      |  DW  |      |      |
   |______|______|______|______|______|______|______|______|______|______|______|______|______|______|______|
   |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |
11 |      |      |      |  DW  |      |      |      |      |      |      |      |  DW  |      |      |      |
   |______|______|______|______|______|______|______|______|______|______|______|______|______|______|______|
   |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |
10 |      |      |      |      |  DW  |      |      |      |      |      |  DW  |      |      |      |      |
   |______|______|______|______|______|______|______|______|______|______|______|______|______|______|______|
   |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |
9  |      |  *3  |      |      |      |  *3  |      |      |      |  *3  |      |      |      |  *3  |      |
   |______|______|______|______|______|______|______|______|______|______|______|______|______|______|______|
   |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |
8  |      |      |      |      |      |      |  *2  |      |  *2  |      |      |      |      |      |      |
   |______|______|______|______|______|______|______|______|______|______|______|______|______|______|______|
   |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |
7  |      |      |      |      |      |      |      |  F   |  A   |  R   |      |      |      |      |      |
   |______|______|______|______|______|______|______|______|______|______|______|______|______|______|______|
   |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |
6  |      |      |      |      |      |      |  *2  |      |  *2  |      |      |      |      |      |      |
   |______|______|______|______|______|______|______|______|______|______|______|______|______|______|______|
   |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |
5  |      |  *3  |      |      |      |  *3  |      |      |      |  *3  |      |      |      |  *3  |      |
   |______|______|______|______|______|______|______|______|______|______|______|______|______|______|______|
   |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |
4  |      |      |      |      |  DW  |      |      |      |      |      |  DW  |      |      |      |      |
   |______|______|______|______|______|______|______|______|______|______|______|______|______|______|______|
   |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |
3  |      |      |      |  DW  |      |      |      |      |      |      |      |  DW  |      |      |      |
   |______|______|______|______|______|______|______|______|______|______|______|______|______|______|______|
   |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |
2  |      |      |  DW  |      |      |      |      |      |      |      |      |      |  DW  |      |      |
   |______|______|______|______|______|______|______|______|______|______|______|______|______|______|______|
   |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |
1  |      |  DW  |      |      |      |  *3  |      |      |      |  *3  |      |      |      |  DW  |      |
   |______|______|______|______|______|______|______|______|______|______|______|______|______|______|______|
   |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |
0  |  TW  |      |      |      |      |      |      |  TW  |      |      |      |      |      |      |  TW  |
   |______|______|______|______|______|______|______|______|______|______|______|______|______|______|______|
```

Fig. 3. Scrabble Board Notation using ASCII Characters.

III. Deliverable 2 – Human Scrabble Player

In this deliverable, we developed a human Scrabble shell for players that accepts two to four players who compete against each other. Each player is provided with a shuffled rack of seven tiles. The notion of legal play, valid word and a trie data structure that stores the entire Scrabble dictionary has been implemented in this deliverable.

In a legal play, a player must extend the existing board word by placing set of tiles from his rack. These letters can be placed parallel or perpendicular to the board word. For instance, if the board contains a word called RUN, a player can extend it by using letters ING. Thus a new word called RUNNING is built. Also, the new words formed across or down the board must be a complete valid word as per Official Scrabble Players Dictionary (OSPD). Hence, to guarantee a legal play, we check if a letter extends on its left, right, top or bottom whenever a tile is placed on a given board-coordinate. Also the words once played, should not be played again.

To ensure the validity of played words, we load the entire OSPD into a trie data structure as it consumes least memory compared to other data structures such as hash-set or array-list. Trie is also known as prefix tree where each node stores alphabet characters and the path of the trie defines actual word. The following Fig 4. is an example of a trie data-structure where it stores words such as AND, BE, BEAR, BEND, CAT. The root note of trie is empty and serves as starting node. Each Trie Node can store up to 26 child nodes (one node for each alphabet).

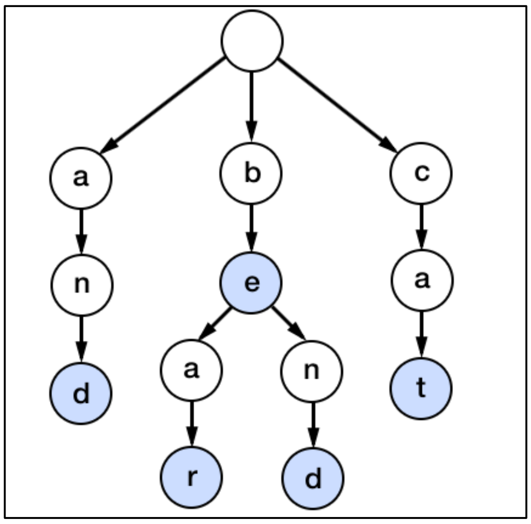
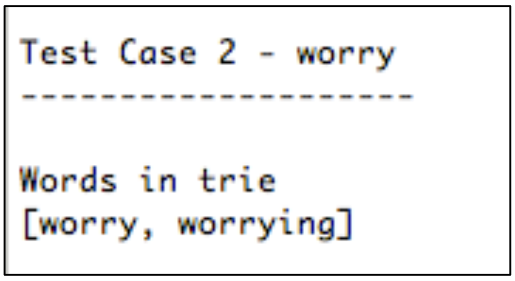


Fig. 4. Trie data-structure example [7].

Each Trie Node can store up to 26 child nodes (one node for each alphabet).
Acceptable English words are checked by passing the prefixes of the new words found on board as parameters. A method called `getWords()` defined in Trie class returns all the words emanating from the given prefix.
The following Fig. 5. is an example of output words retrieved from trie for the word "WORRY".



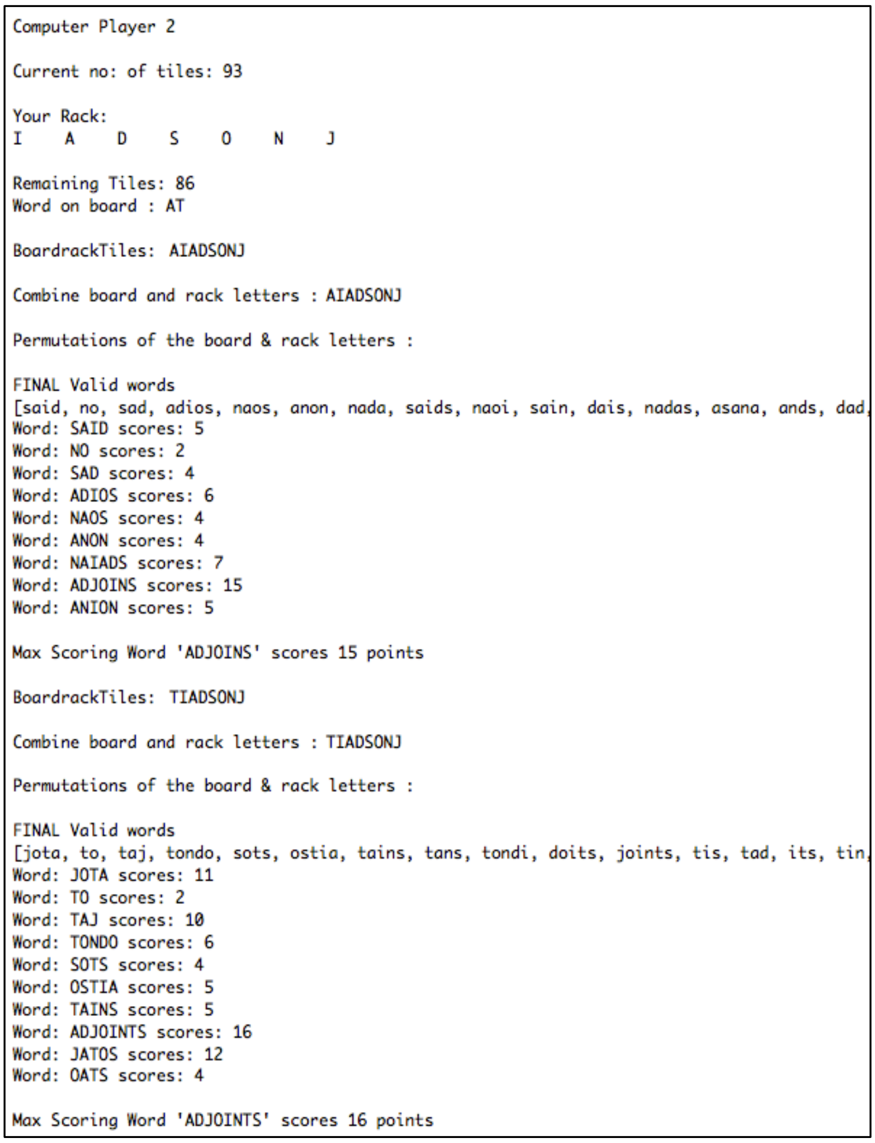
```
Test Case 2 - worry
-------------------

Words in trie
[worry, worrying]
```

Fig. 5. Trie retrieval for word WORRY.

IV. Deliverable 3 – Find the Possible Set of Words using Trie Data–Structure

Deliverable 3 was to develop a trie data structure useful for the discovery of the most promising words that could be played on each turn. To do that we accumulate the letters from the board and rack to form a combination of possible letters that could be placed on the board. Each letter from the board is fetched to combine with player's rack of tiles. These letters are then passed to a trie data structure as a parameter to explore the permutations of potential words that can be formed from the combination of board and rack letters. The score is also computed for each word in the collection. Once the set of all promising words is created, we play with the highest scoring word from the given set.

The strategy used here is similar to the straightforward strategy used in current best Scrabble computer player called Maven. In the straightforward strategy, human players place the longest word that earns the highest score. A drawback with this method is that it can generate a lot of hot spots for the opponent to place their own tiles [4]. These hot spots on the game board let the opponents to place a high score move in the next turn. This move could be a bingo move that lets the opponents earn 50 bonus points in one turn. Thus, we must remember that the highest scoring move is not always the best one. An intelligent AI player must efficiently apply different strategies at each turn depending upon the board state. The following Fig. 6. shown below is an example of an AI player that generates potential words.



```
Computer Player 2

Current no: of tiles: 93

Your Rack:
I    A    D    S    O    N    J

Remaining Tiles: 86
Word on board : AT

BoardrackTiles:  AIADSONJ

Combine board and rack letters : AIADSONJ

Permutations of the board & rack letters :

FINAL Valid words
[said, no, sad, adios, naos, anon, nada, saids, naoi, sain, dais, nadas, asana, ands, dad,
Word: SAID scores: 5
Word: NO scores: 2
Word: SAD scores: 4
Word: ADIOS scores: 6
Word: NAOS scores: 4
Word: ANON scores: 4
Word: NAIADS scores: 7
Word: ADJOINS scores: 15
Word: ANION scores: 5

Max Scoring Word 'ADJOINS' scores 15 points

BoardrackTiles:  TIADSONJ

Combine board and rack letters : TIADSONJ

Permutations of the board & rack letters :

FINAL Valid words
[jota, to, taj, tondo, sots, ostia, tains, tans, tondi, doits, joints, tis, tad, its, tin,
Word: JOTA scores: 11
Word: TO scores: 2
Word: TAJ scores: 10
Word: TONDO scores: 6
Word: SOTS scores: 4
Word: OSTIA scores: 5
Word: TAINS scores: 5
Word: ADJOINTS scores: 16
Word: JATOS scores: 12
Word: OATS scores: 4

Max Scoring Word 'ADJOINTS' scores 16 points
```

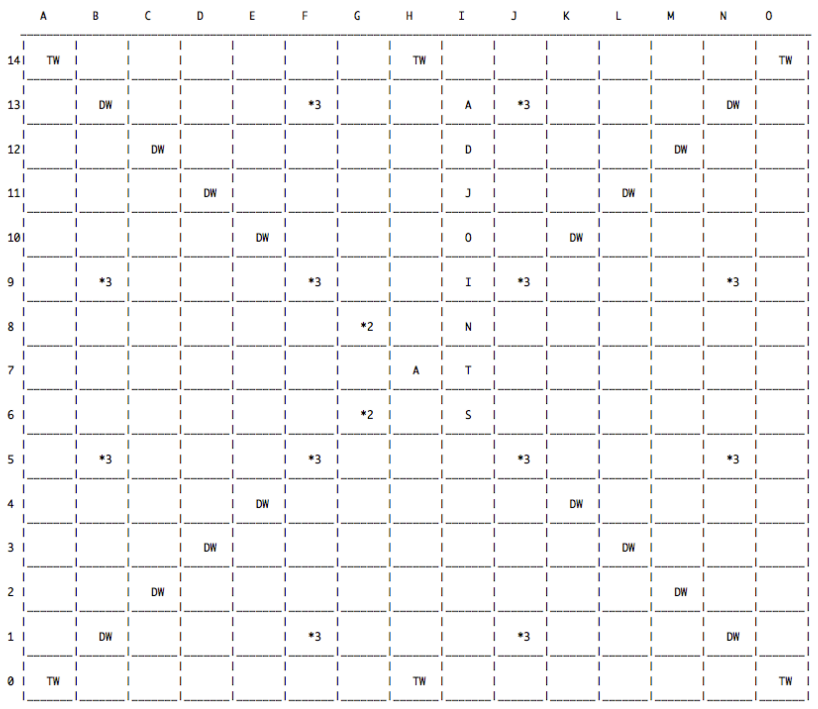
Fig. 6. Permutations of possible valid words and the maximum scoring word from that set.

Here we iterate all the letters from the board and generate a valid permutation of the letters on the board and the rack. In the above result displayed in fig. 6., two blends of board and rack tiles are formed such as AIADSONJ and TIADSONJ. From these letter combination, we can state that the first letters of the BoardrackTiles variable, A and T, are the letters from the board and the letters IADSONJ are the rack tiles. ADJOINTS is the maximum scoring word among the possible set of words formed from the combination of board and rack letters.

V. Deliverable 4 – A Simple AI Player

Deliverable 4 was to implement a simple AI player with limited intelligence. In this Deliverable, we developed a computer player that can compete with a human player built in Deliverable 2. Rather than providing inputs such as direction, board coordinates or letters as a comma separated value, the program automatically fetches the letters from the board and rack to form a set of promising words that can be placed on the board.

As we have already completed the retrieval of the most promising words using a trie data structure in Deliverable 3, we emphasis on the placement of the most potential words in the legal direction. To implement this, we retrieved the current direction of placement of board words along with the board tile positions. If the word is initially placed in a horizontal direction, we first check if the new word formed is an extension of the existing word on board. If yes, then we have to lay the new word in the same direction of the current board word. However, if the new word is not an extension, then we need to place it perpendicular to the initial board word. In both the cases, to place a new word, we first have to be aware of the X-Y board coordinates of the first letter of the new word. This is calculated from the existing board word coordinate and getting the length of the new word. For instance, the final board state after placing the word ADJOINTS is shown below in Fig. 7.



```
     A     B     C     D     E     F     G     H     I     J     K     L     M     N     O
14 | TW  |     |     |     |     |     |     | TW  |     |     |     |     |     |     | TW  |
13 |     | DW  |     |     |     | *3  |     |     | A   | *3  |     |     |     | DW  |     |
12 |     |     | DW  |     |     |     |     |     | D   |     |     |     | DW  |     |     |
11 |     |     |     | DW  |     |     |     |     | J   |     |     | DW  |     |     |     |
10 |     |     |     |     | DW  |     |     |     | O   |     | DW  |     |     |     |     |
 9 |     | *3  |     |     |     | *3  |     |     | I   | *3  |     |     |     | *3  |     |
 8 |     |     |     |     |     |     | *2  |     | N   |     |     |     |     |     |     |
 7 |     |     |     |     |     |     |     | A   | T   |     |     |     |     |     |     |
 6 |     |     |     |     |     |     | *2  |     | S   |     |     |     |     |     |     |
 5 |     | *3  |     |     |     | *3  |     |     |     | *3  |     |     |     | *3  |     |
 4 |     |     |     |     | DW  |     |     |     |     |     | DW  |     |     |     |     |
 3 |     |     |     | DW  |     |     |     |     |     |     |     | DW  |     |     |     |
 2 |     |     | DW  |     |     |     |     |     |     |     |     |     | DW  |     |     |
 1 |     | DW  |     |     |     | *3  |     |     |     | *3  |     |     |     | DW  |     |
 0 | TW  |     |     |     |     |     |     | TW  |     |     |     |     |     |     | TW  |
```

Fig. 7. Final board state after placing word ADJOINTS.

VI. Conclusion

By the end of CS 297, the basic implementation of human and AI Scrabble player engine that can accept two to four players has been completed. We have also explored the various parts of Scrabble such as the basic notation system, human player mode that can play legal moves and valid words, an AI computer player that can generate possible word list and finally play a word that earns highest score in each turn. Presently, the AI player plays against human and have only partial intelligence.

In CS 298, we will build a complete intelligent AI player that can play against the current best computer player called Maven that can beat human. We will also look into other computer player variants that can be played against our new AI player. We are also planning to model efficient opponent strategies that let the system defeat the strongest opponents. Thus, the two AI players would be competing against each other to discover the best Scrabble strategies used in the game by turning the game of imperfect information into a perfect one. To achieve this goal, we may need a simulation process for exploring the search space of possible future outcomes in limited time and for optimal decision making.

References

[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] C. Browne *et al.*, "A survey of Monte Carlo Tree Search methods," in *IEEE Trans. Comp. Intelligence and AI in Games*, Vol. 4, no. 1, pp. 1-43, Mar. 2012. [Online]. Available: http://www.cameronius.com/cv/mcts-survey-master.pdf

[4] M. Richards and E. Amir, "Opponent modeling in Scrabble," in *Proc. 20th International Joint Conf. Artificial Intelligence,* Hyderabad, India, 2007, pp. 1482-1487. [Online]. Available: http://reason.cs.uiuc.edu/mdrichar/my_papers/IJCAI07-239.pdf

[5] 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

[6] 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

[7] "Lab 9: Trie". [Online]. Available: http://www.cs.bu.edu/courses/cs112/Lab09.html