Learning a Reward Function for Scrabble Using Q-Learning

Bihan Jiang
Department of Computer Science
Stanford University
bihan@stanford.edu

Michael Du
Department of Computer Science
Stanford University
mdu7@stanford.edu

Sam Masling
Department of Computer Science
Stanford University
smasling@stanford.edu

Abstract

Computing the optimal move in a scrabble game is a long-studied problem in reinforcement learning. Monte Carlo Tree Search (MCTS) methods have proven powerful in planning for large scale sequential decision-making problems, such as Scrabble in the past. However, in order to address the delayed reward issue, MCTS methods build look-ahead trees to estimate the action values, which is usually time-consuming and memory demanding. Additionally, MCTS methods suffer when the planning depth and sampling trajectories are limited compared to the delay of rewards or when the rewards are sparse. The current cutting edge Scrabble agents depend on Monte Carlo simulations and thus depend on the time available for running the simulations. In this paper, we present an agent that can perform at a similar level to the existing agents, but with a much faster performance time, by using the valuation function to speed up Q-learning and simulate on fewer moves due to selecting better plays. We use linear regression to come up with a valuation function to create a high scoring Scrabble agent. The valuation function presented in our paper is based on the score of the played word, the tiles left on the player's rack, and the state of the board.

1 Introduction

1.1 Scrabble

Scrabble is a board game where 2-4 players take turns forming words on a 15 $\times$ 15 board by placing single letters in the form of tiles onto the board. These tiles must form words defined in a standard dictionary that all players should agree upon before the game begins. All words labeled as a part of speech (including those listed of foreign origin, and as archaic obsolete, colloquial, slang, etc.) are permitted with the exception of words that are always capitalized, abbreviations, prefixes and suffixes standing alone, and words requiring a hyphen or an apostrophe. The words must be formed in a way in which they may be read from left to right in rows or downwards in columns (diagonal words are not allowed) and at least one tile in the word must be placed next to an existing tile on the board. Additionally, the board contains “Premium" squares which adds a multiplier to the tile or word, depending on the color of the square.
The game begins with 2-4 racks and 100 letter tiles in a letter bag, which players draw from for the first play. The player with the letter closest to “A" plays first. Then, the players draw seven new letters and place them on their racks. For every tile placed on the board, the player draws a replacement tile

from the bag. The game ends when all letters have been drawn and one player uses his or her last letter, or when all possible plays have been made.

Figure 1: Example Scrabble board with write squares as normal squares and colored squares as premium squares.

There are several factors that make the game of Scrabble difficult to model. One factor is having imperfect information about the racks of other players, which makes it difficult to predict opponents' moves. Additionally, there is randomness introduced through the bag of letters, as random letters are drawn into each player's rack during each round. It is also difficult to model the state space of Scrabble, since the board is marked with specific letters as opposed to a binary marking.
In response to the challenge presented in creating an AI agent for Scrabble, our project aims to build such an AI agent that is comparable to existing agents using the model-free reinforcement learning algorithm Q-Learning. In particular, we attempt to improve the valuation function that determines the strength of different moves during the game.

1.2 Related Work

1.2.1 Maven

Maven is an AI Scrabble player that is used in official licensed Hasbro Scrabble games. The algorithms divide the game play into three phases. [4]

1 Mid-Game: The mid-game phase lasts from the beginning of the game until there are nine or fewer tiles remaining in the letter bag. The program then finds all possible play from the given rack and uses heuristics to sort them in order or quality. The top promising moves are then simulated in game-play thousands of times, where the program simulates the random drawing of tiles, plays forward a set number of plays (2-ply deep), and compares the points spread of the moves' outcomes.

2 Pre-Endgame: The pre-endgame phase works in a similar way to the mid-game phase, except that it is designed to attempt to yield a good end-game situation.

3 Endgame: The endgame phase begins when there are no tiles left in the letter bag. During this phase, the players now have perfect information, since the players can now deduce from the initial letter distribution the exact tiles on each other's racks. During the endgame phase, Maven uses the B-star search algorithm to analyze the game tree.

1.2.2 Quackle

Quackle [2] is another AI Scrabble player that rivals the best players in the world. Quackle includes a move generator, move evaulator, simulator, and user interface that can be used with any board layout, alphabet, lexicon, and tile distribution. Quackle uses a similar approach as Maven in the mid-game phase. The heuristic function used is a sum of the move score and the value of the rack leave, which is computed favoring both the chance of playing a strong word on the next turn as well as leaves that garnered high scores when they appeared on a player's rack in a sample of Quackle vs Quackle games.

The win-percentage of each move is then estimated and the move with the highest win-percentage is played.

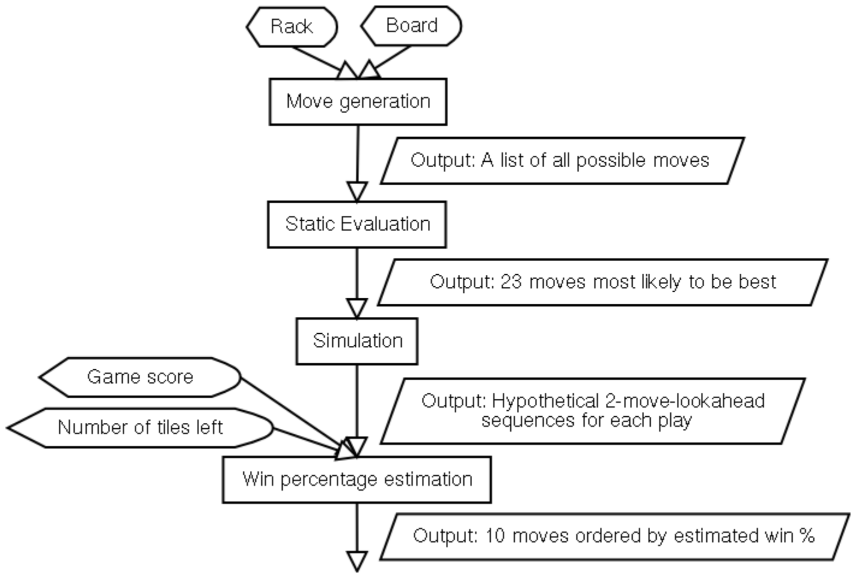

Rack
Board
Move generation
Output: A list of all possible moves
Static Evaluation
Output: 23 moves most likely to be best
Simulation
Game score
Number of tiles left
Output: Hypothetical 2-move-lookahead sequences for each play
Win percentage estimation
Output: 10 moves ordered by estimated win %

Figure 2: Diagram of Quackle's AI.

2 Methods

Before implementing each model fully, we implemented a baseline and an oracle. These low-intelligence models are intended to provide benchmarks for the performance of our more complex models. All of our code can be accessed at the following Github repository:

`https://github.com/smasling/ScrabbleAI`

2.1 Baseline

The purpose of the baseline is to create an extremely simplistic model that provides a reasonable lower bound on the performance of a classification process. For our baseline algorithm, we implemented a self-playing Scrabble AI where our reward function is simply the score of the word played. This is a simplistic heuristic estimate of the reward function since there are many subtleties that affect the quality of a move, such as the letters remaining in the bag, the state of the board, the letters the other players possess, etc. No reinforcement learning was used for the baseline.
We determine the success of our model by calculating the win rate out of 1000 games against our improved model, and the average score of the games played.

2.2 Oracle

The purpose of the oracle is to determine a upper bound, best-case result. Our oracle was human performance. One of the contributors, Sam Masling, is a professional Scrabble player (most recently ranked 190th in the country). While we correctly expected our algorithm would eventually surpass our oracle, as computer agents have a significant advantage over human players, our oracle was easily able to best our baseline algorithm, and was helpful as a rule of thumb when tuning hyper parameters when training our improved algorithm.

2.3 GADDAG Background

A GADDAG is a data structure used in generating moves for Scrabble and other word-generation games. It is often compared to move-generation algorithms using a directed acyclic word graph

(DAWG) such as the one used by Maven. It is generally twice as fast as the traditional DAWG algorithms, but take about 5 times as much space for regular Scrabble dictionaries.[1]

A GADDAG is a specialization of a Trie, and contains states and branches to other GADDAGs. It works by storage every reversed prefix of every word in a dictionary, meaning every word has as many representations as it does letters. Since the average word in most Scrabble dictionaries is 5 letters long, the GADDAG will be about 5 times as big as a simple DAWG.

2.4 Move Generation

We use GADDAG's to construct our move-generation algorithm.
There are three constraints that any move-generation algorithm must follow:

1 Board constraints: a word may only be built onto existing letters of the board, and tiles may only be played on empty squares.

2 Rack constraints: Tiles may only be placed from one's rack.

3 Dictionary constraint: All words formed must exist in the game's dictionary.

Maven uses a DAWG-based algorithm, which meets the second and third constraints since the DAWG is built around the dictionary and is traversed using tiles in the rack. However, the DAWG failts to address the first constraint. For example, suppose one would like to form the word "OPT" using the 'P' from the pre-existing word "CAP" on the board. In this case, one must search through the dictionary for all words containing letters from the rack where the third letter is 'P'. This is an inefficient method, since when searching through the DAWG, many searches through the trie will be fruitless.

To combat this inefficiency, we use the GADDAG's storage of prefixes. By traversing the 'P' branch of a GADDAG, one can determine all words that contain a 'P' in the composition, and can form the word with tiles in the rack. For example, searching for 'P' could yield "PO+T". The letters between 'P' and '+' can be placed above the 'P' on the board, and the letters after the '+' can be placed below the 'P'.

2.5 Valuation Function

If our group had a more powerful computer or a longer time period, we would have used Markov Decision Processes and value iteration to determine our value function, but due to the massive size of the state space of Scrabble, we found it more feasible to train our weights using linear regression.

In our final implementation, we use a stronger valuation function based on the following features:

1 Score of the played word

2 Tiles left on the player's rack

3 State of the board- which multipliers are exposed as the player makes the play.

compared to the previous valuation function of just the score of the played word. We trained on a dataset that for the features contain the score of the word played, the state of the board, and the tiles left. The label is calculated by the difference in the scores four plays (2 turns) from the starting point minus the difference in the scores at the starting point. For example, if we have a training example $x^{(i)}$ at turn 3 for player 1 where the score is $50 - 30$, and after 4 plays (turn 5) the score is then $90 - 60$, our label for $x^{(i)}$ is 10 because $\nabla_1 = 20$ and $\nabla_2 = 30$.
We created a simulation class to simulate games between two computer players to come up with a better valuation function than simply looking at the top score. At each time step (move), the listed features above were tracked, then a linear regression was used to find the optimal weights for the features. The new regression score was then used as our valuation function.

2.6 Q-Learning

Q-learning is a model-free reinforcement learning algorithm whose goal is to learn a policy, which tells an agent what action to take under what circumstances.
We model the game of Scrabble as a Markov Decision Process (MDP) with a given state containing the following information:

1. state['board'] = a representation of the communal board
2. state['players'] = a list of tuples (score, letters) where letters are the letters on the current player's rack
3. state['letters remaining'] = a list of the letters left in the bag that have not been drawn or played.

This inherently The main methods of our MDP class run according to the rules of Scrabble described in the introduction. We then run Q-learning on our MDP according to algorithm 5.3 from Kochenderfer [3].

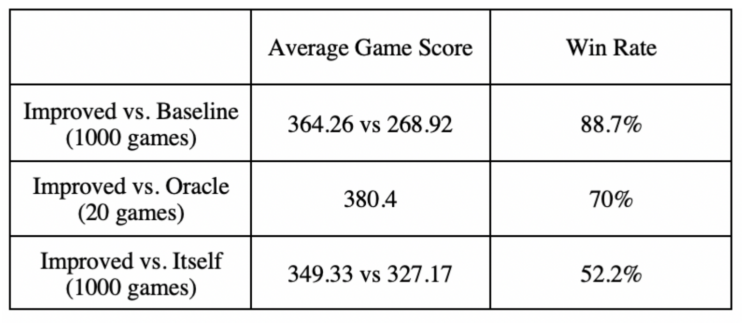
3 Results

After running our simulations we found the following results:

We found that when our improved algorithm was played against the baseline algorithm, the average score per game for our improved algorithm was 364.26 points versus 268.92 points for the baseline, and the win rate was for the improved algorithm was 88.7%.

We found that when our improved algorithm was played against the oracle, the average score per game for our improved algorithm was 380.4 points, and the win rate was 70% (it won 14 out of 20 games against our oracle). While our improved algorithm had a lower win rate against the oracle versus the baseline, we see that it's average score went up. This is likely because the baseline algorithm would close up the board rapidly, greedily searching for the highest scoring play, which also hampered the improved algorithm's ability to score well.

We found that when our improved algorithm was played against itself, the average score per game for our improved algorithm was 349.33 points versus 327.17 points, and the win rate was 52.17%.



	Average Game Score	Win Rate
Improved vs. Baseline (1000 games)	364.26 vs 268.92	88.7%
Improved vs. Oracle (20 games)	380.4	70%
Improved vs. Itself (1000 games)	349.33 vs 327.17	52.2%

Figure 3: Average Game Score and Win Rate for Improved Algorithm.

4 Conclusions

Setting up the architecture of the game ended up taking significantly more hours than expected since we ended up creating the game and simulations from scratch.

Our investigation suggests that Q-learning is a successful model. Compared to Maven and

Quackle, our use of Q-learning leads to a Scrabble AI that does not rely on real-time simulation to play against another agent. Given enough time, the agent learned to play more strategically, taking into account defense, future actions, and letter conservation. One primary limiting factor of the project was the need time needed to tune our move generation functions and simulation ability. This caused iteration on the valuation function to be slow.

We believe that with improvement on the move generation and simulation ability, as well as a longer period to train our valuation function, the model could learn to play a more robust strategy and result in a higher win rate and average game score. Additionally, while our model is able to play our baseline, oracle, and itself, the model has not been tested against other open sourced Scrabble AI.

Our group plans to extend the project by both advancing the agent to play against other open-sourced Scrabble AI, as well as extending the model's move generation speed by extending the GADDAG implementation.

5 Contributions

Bihan Jiang contributed to preliminary research, analysis of different implementations, prepared data visualizations, ran simulations, and contributed to this paper.
Sam Masling programmed much of the Scrabble game-play, the Q-learning, acted as the oracle, and contributed to this paper.
Michael Du programmed much of the Scrabble simulation, the Q-learning, ran simulations, and contributed to this paper.

References

[1] S. Gordon. A faster scrabble move generation algorithm. In *Software: Practice and Experience*, volume 24, pages 219–232, 1994.

[2] J. Katz-Brown and J. O'Laughlin. How quackle plays scrabble. 2005.

[3] M. Kochenderfer. Decision making under uncertainty: Theory and application. MIT press, 2015.

[4] B. Sheppard. World-championship-caliber scrabble. In *Artificial Intelligence*, volume 134, pages 241–275, 2002.