Automated Scrabble Environment and Intelligent Agents

Eeshan Malhotra, 14305R001

Vivek Poonia, 140359005

ABSTRACT

This report describes our work on an automated interface for playing Scrabble, and various flavours of intelligent agents designed to compete in this environment. The agents are based on a mixture of manually derived rules and learned characteristics. For comparison, a popular Scrabble program, *Quackle* is also adapted for playing in the environment.

The interface is designed to provide researchers a standardized environment to learn and test their agents in, and the results we obtained using our own intelligent agents will, hopefully, motivate the direction for future work in Scrabble agents.

1. INTRODUCTION

Scrabble is a popular, language-based board game, where players take turns to play alphabet tiles from their individual racks of seven tiles to the common board, to form valid words. While Scrabble plays two to four players, formal tournament games are always one-on-one [5]. The moves are constrained by the previous plays made by both players. Each move scores certain points based on the values indicated on each tile used, and special bonus squares marked as such on the board. Tiles used in each turn are replaced by replenishing from a bag with a finite number of tiles. A special move called a 'bingo' involves using all seven of the tiles on one's rack and earns a bonus of 50 points. The game continues till the tile bag is empty, and one of the players uses up all of his/her tiles. The aim is to maximize the cumulative score difference between yourself and your opponent.

From an artificial intelligence perspective, Scrabble is an adversarial, zero-sum game, and can be modelled in a POMDP or an MDP framework with an inherently non-deterministic environment. Players can only see the tiles on the board and on their own racks. So it is a game of imperfect information. There is also an element of randomness, since drawing tiles from the bag is a stochastic process. Long-term strategy is quite important in making decisions about which tiles to leave on one's rack for future moves, which areas of the board to open up, and how to provide limited avenues for your opponent to score.

The game has a rich history, and is actively played competitively in a standardized format in numerous tournaments worldwide.
All of these factors combined make Scrabble a novel and challenging pursuit.

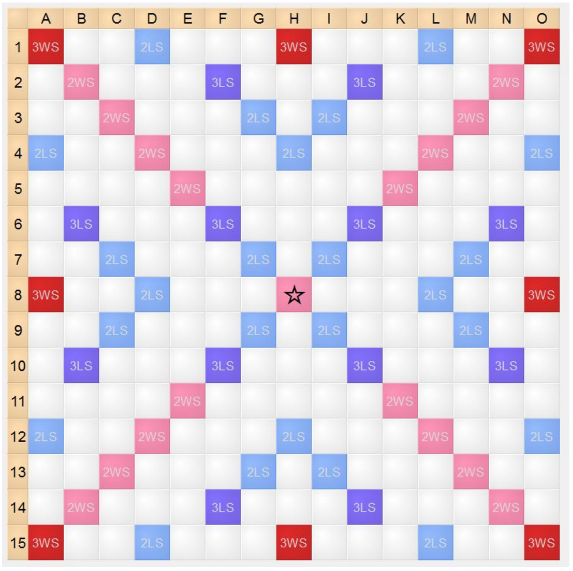

Figure 1: Schematic of a typical scrabble board

The current best Scrabble agents rival the expertise of the best human players in the world. But the game has not been treated with the rigour anywhere close to what its peers (like chess, backgammon, and go) have received. Especially with recent advances in AI, it is the authors' opinion that a revisit to the game is overdue, and will result in great output, both, in terms of tangible, high-performing agents, as well as intangible learning about the nature of intelligent agents, and AI as a whole. Our results show that this direction of research in intelligent Scrabble agents indeed shows promise.

2. RELATED WORK

While only minimal literature has been published in the field in recent times, there are two existing Scrabble agents which are widely regarded as the current best Scrabble playing programs. Both of these rival the proficiency of the world's best human players, and have competed with human experts in tests. However, little improvement has been made in these in recent times, despite some major advances in our understanding of reinforcement learning and game-playing.

The first is Maven, which is developed by Brian Sheppard [4], and is now owned by Hasbro Inc. (which also licenses the game Scrabble in North America). It uses a tree-search algorithm, with techniques which are predecessors of what is today referred to as reinforcement learning. Unfortunately, this proprietary agent is available to public only in packaged games as released by Hasbro.

The second is Quackle [2], developed by Jason Katz-Brown and John O'Laughlin. While Quackle has been used in fewer computer vs human trials, it is widely regarded as on par with Maven [3], and is used by some of the best experts in the world to analyze their games. Though Quackle's learning techniques are unpublished, the agent itself is made partially open-source under a GNU General Public License. Quackle utilizes a series of heuristics to assign a value to each possible move from a given state. This is similar to what is common practice in reinforcement learning literature, but the value estimation is not as principled.

From the authors' personal experience, Quackle is quite an aggressive player and often lacks the ability to play defensively when ahead in a game. On a related point, it lays more emphasis on winning by scoring more points, while an equally good (or better) strategy might be by restricting the opponent to a lower score.

In this work, we use Quackle as not only a benchmark, but also an active tool in the learning of one of our agents. The agent learns and adapts by repeated play against Quackle.

3. PROBLEM STATEMENT

Our aim with this work was two-fold:

1. **Learning and Testing Environment**
   Create an environment that mediates matches between two automated Scrabble agents, and serves as the generation and communication interface. This can be used for testing and/or learning. Any agents that can successfully send and receive information using the standardized protocol should be able to utilize
2. **Intelligent Agents**
   Create intelligent scrabble agent(s) that calculate all possible moves given a board and rack state, and then smartly adjudicate between these to suggest a single move to maximize the relative Scrabble score.

4. LEARNING ENVIRONMENT

The learning environment was developed with the aim of allowing any agent that follows the protocol to be able to interact and compete. A high degree of customisation is provided, and assumptions are kept to the minimum possible. It was also designed to allow users to plug in any settings without requiring any knowledge of the programming of the environment itself.

4.1 Parameters

While the primary objective has been to develop this environment for Scrabble, the interface can be configured to work just as well for the related family of games, such as Words With Friends. Customizable parameters include:

1. Board dimensions
2. Locations and values of bonus squares
3. Tile distribution
4. Tile values
5. Starting state of the board
6. Total time provided to each agent
7. Dictionary used to check validity of words
8. Settings to control verbosity of output

In addition, users can provide generic executables or even scripts as input which will be called by the interface.

4.2 Message Passing

Every communication from the environment to the is in the form of a JSON string. The string has the following structure:

```
{
    "status": {
        "moverequired": <>,
        "endofgame": <>
    },
    "secondstogo": <>,
    "errcode": <>,
    "score": {
        "me": <>,
        "opponent": <>
    },
    "board":<>,
    "rack": <>
}
```

Listing 1: Message Passing from the Interface

Where each ⟨⟩ is replaced by appropriate information. Messages requesting moves use the the `moverequired` flag set to `true`, whereas acknowledgement messages have it set to `false`.

Agents are required respond to the environment with a single move each time, indicating the location of the move, and the word formed. A detailed description of the format can be found in the ReadMe file in the software bundle, but as an example, placing a `P` and an `R` around an existing horizontal word `APE` may be input as `C5 P(APE)R`, or `C5 P(A)(P)(E)R`, or `C5 P###R`.

5. AGENTS AND ALGORITHMS

5.1 Listing Moves

Unlike a number of other games, finding all valid moves given a board and rack of letters is not a trivial task for Scrabble. Multiple kinds of plays and a large dictionary (~270,000 words) make this task difficult. The average number of possible moves per turn is about 300, but varies from none (requiring a 'Pass') to well over a 1000.

Our initial approach was to store the dictionary as a Directed Acyclic Word Graph (DAWG) [1], with an extension to a bi-directional DAWG. While this was quite fast when it came to actually listing moves, we noticed that loading an arbitrary data structure into memory required more time than reading a larger map (python dictionary) stored as plain-text. This trade-off required more storage space, but improved the move finding speed significantly.

The final structure we found to be optimal was storing the dictionary in two formats. First, a simple plaintext list. Second, 14 hashmaps, one each for lengths ranging from 2 to 15 letters. Every one of these one maps each unique set of letters (which permute to give at least one valid English word), to this list of words. For example, one entry in the dictionary of 7-letter words would read:
`AENORST: ATONERS, SANTERO, SENATOR, TREASON`
This is useful when calculating the odds of a bonus play in a potential rack.

The actual move finding algorithm proceeds thus: First, the problem of finding all possible moves on a board is reduced to finding only horizontal moves - since the same process can then be repeated on a transposed board to find the vertical moves (This symmetry is also exploited to save time when finding the opening move - only horizontal moves are considered).

Next, we find 'anchor points' for each row. An anchor point is an empty square, with an adjacent occupied square. Every valid move must touch at least one anchor point (The only exception is the very first move, which is handled separately as a special case). Further, it is the anchor points only which are constrained in terms of the tiles that can be played. Thus, for each anchor point, we find the set of all tiles that can be played there. Taking an intersection of the anchor constraints with the tiles available on the rack gives us a very small set of playable tiles on each square.

Further, as long as these constraints are obeyed, all rows can be treated *independently*. Now, for each row, it is only a matter of performing a regular expression search to find words that fit all the constraints, row by row. RegEx searches are typically quite optimized in terms of speed in most languages, including Python.

This generates all the possible moves given a board state and a rack of seven tiles.

5.2 Greedy Agent

A naive strategy, once all the possible moves are known, would be to pick the highest scoring move (breaking ties arbitrarily). This would maximize the immediate score, but not take into account two things - the potential future score from the 'leave' on one's rack; and the opportunities provided to the opponent by the move.

Using a large set of human expert games [1], we used Quackle to generate the move that it believed to be optimal at each juncture in the game. This was found to be the same as a greedy move between 32-38% of the time.

A full greedy agent was programmed and tested against Quackle, and obtained a win rate of 29.5%, which we perceived to be surprisingly high. More complete results are included in the results section.

5.3 Clabbers1: Handcrafted, Heuristic Agent

Clabbers1 was the second agent we developed. While the model is hand-crafted, this is the point where the process of move selection in Scrabble is posed as an MDP, and it sows the seeds for many of the strategies used in the learning agent.

We defined our state to corresponds to a combination of tile positions on the board, tiles on ones rack and the scores of both players. An action corresponds to playing a specific word at a specific location on the board. The objective that we seek to maximise is the score difference between the final score of the agent minus the final score of the opponent.

Since we are modeling the problem as an MDP, it is worth clarifying that we use episodic tasks with a γ value of 1.

The actual number of state-action pairs in Scrabble is astronomical. To propose to learn any smart strategy in this space would be a foolish task. Instead, we mapped the state-action space a lower dimensional feature space.

Each board state and move combination is mapped to a single vector, containing values of the following features.

Certain features are dependent only on the board (state):

- currentScoreDifference
- numTilesLeftInBag
- numUnseenConsonantsMinusnumUnseenVowels
- numUnseenBlanks

The following are dependent on the board, the rack, and the proposed move (state-action)

- Immediate reward maximization:
  - moveScore
- Potential for future scoring:
  - leave_length
  - leave_ConsonantsMinusVowels
  - leave_blanks
  - leave_bingoProbOnNextDraw
  - leave_expectedNumberOfBingosOnNextDraw
- Preventing opponent from scoring highly:
  - numNewHotspotsAccessible
  - numNewBingoLanes

Using these features, and intuitive knowledge as Scrabble players, we handcrafted a rule based agent, that tries to simulate how a human player thinks during a game of Scrabble. This gave a huge advantage over a greedy agent. Clabbers1 won 42% of its games against Quackle.

5.4 Clabbers2: A Learning Agent

With the foundation of a feature space in place, we developed an agent that would learn to weight these features by repeated play against a better agent. We formulated the task as follows: Given a state-action pair (as defined in the previous subsection), estimate the (signed) difference in scores after three more moves by each player. The logic behind using a horizon of six moves (3 moves $*$ 2 players) was motivated by the experience that predicting a score too

[1]Obtained courtesy http://www.cross-tables.com

far in the future may be impossible because of the highly random nature of the environment.

A linear model was used, essentially formulating the problem as given the current state, predict the Q-value of each state-action pair as a linear combination of the features. The weights of this linear model are what we want to learn. Note that defining a weight vector also effectively defines a policy - for every move, calculate a Q-value using the current weight vector, and select the move with the highest estimated Q-value.

The learning strategy, then, is this:
We start with an initial set of weights (educated guesses), w, and update these weights after each complete play-through of a game. Once the game is over, for every move in the game, generate a record containing a feature vector, and the target (score difference after six moves). Using this data, we train a supervised linear regression model, and obtain a new set of weights w_{game}. Now, we update w to move in the direction of w_{game}, as $w = w + \alpha(w - w_{game})$

In pseudocode,

Algorithm 1 Clabbers2 Learning Algorithm

```
1:  $w \leftarrow$ initial weight vector
2:  for $game = 1$ to $n$ do
3:      Play game as:
4:      while asked for move do
5:          Map each move to feature vector $\phi$
6:          Play argmax $w^T\phi$
7:      Generate training data $D$ as:
8:      $D \leftarrow$ null
9:      for each played move in game do
10:         $y \leftarrow$ Score difference after 3 moves of each player
11:         Append tuple $\langle \phi, y \rangle$ to $D$
12:     $w_{game} \leftarrow$ coefficients of lin-reg model $y \sim \phi$
13:     $w \leftarrow w + \alpha(w - w_{game})$
14:     Write $w$ to disk
```

($y \sim \phi$ is notation for a model trained using the column y as the target, and the columns containing ϕ as the predictors)

In this learning approach, an α value of 0.1 was used, as a somewhat arbitrary value.

6. EVALUATION AND RESULTS

We used four agents in our evaluation - Greedy, Clabbers1, Clabbers2, and Quackle. The most direct metric for evaluation is the fraction of games won by the agent against each of the other agents.

In this respect Quackle clearly stood out. But the Clabbers agents proved to be surprisingly effective. The win fraction of the agents against each other is summarized in Table 1.

Table 1: Fraction of games won by the row player against the column player

	Greedy	Clabbers1	Clabbers2	Quackle
Greedy	-	44%	46.4%	29.5%
Clabbers1	56%	-	x	42%
Clabbers2	53.2%	x	-	38%
Quackle	70.5%	58%	62%	-

7. DISCUSSION AND FUTURE WORK

We successfully created a learning and testing environment which can be utilized by other researchers working to create a smarter scrabble agent. Moreover, since a standardized format and syntax often goes a long way in making research accessible, it is hoped that this will ease collaboration and comparison among different groups, and lead to progress a problem which has not received due attention.

While none of our agents could beat the benchmark of Quackle, we certainly obtained some very encouraging results. Clabbers1, while hand-crafted, utilized a more complex model (in terms of variable interactions). However, it did prove the power of the feature space we mapped to.

Clabbers2 had the advantage of learning automatically, but was perhaps not complex enough to capture the dynamics of the problem space. A combination the two strategies seems like a very promising approach to follow. It would seem that there are potential avenues to combine the merits of each apporach and improve the model.

Concretely, we propose three directions for future work:

1. **Stronger feature space**
   While the features proved to be quite useful, these failed to capture the intricacies of letter synergies. For example, a `Q` is very high scoring if held with a `U`, but quite a nuisance otherwise.
   Apart from letter synergies, features based on modeling the opponent's rack and playing style may prove to be useful.

2. **Adding exploration**
   In the current framework, once the weights were know, the policy was deterministically known. All decisions were taken as per the policy. However, occasionally going off-policy and exploring might add the randomness that's needed to more effectively scan the policy space.

3. **More complex models**
   a decision tree or random forest model could be developed with a similar pipeline as used in this work. The difficulty lies in merging the existing tree (or forest) with the observed game tree (or forest). However, this is not an insurmountable task.
   Further, as a stretch goal, we had hoped to be able to develop a neural network base model during the project. Unfortunately the scope we covered proved quite immense in itself. But the idea has all the makings of a good approach - once again, it combines a more complex model with continuous learning through repeated play.

In conclusion, we want to re-emphasize the value of a mathematically principled, reinforcement-learning based approach to the problem of move selection in Scrabble. We strongly believe the with some more focus on the problem, it has the potential to become another one of AI's great game-playing successes.

8. REFERENCES

[1] A. W. Appel and G. J. Jacobson. The world's fastest scrabble program. *Communications of the ACM*, 31(5):572–578, 1988.

[2] J. Katz-Brown and J. O'Laughlin. Quackle 0.9 beta release offers leading-edge artificial intelligence and novel analysis tools, 2006.

[3] M. Richards and E. Amir. Opponent modeling in scrabble. In *IJCAI*, pages 1482–1487, 2007.

[4] B. Sheppard. World-championship-caliber scrabble. *Artificial Intelligence*, 134(1):241–275, 2002.

[5] WESPA_Rules_Committee. World english-language scrabble players association game rules, October 2015.