CS221 Fall 2016 Project Proposal: Scrabble AI

Authors: Colleen Josephson {cajoseph} and Rebecca Greene {greenest}

Introduction

We propose a Scrabble AI to play against human or alternate AI agents. Scrabble is a turn-based zero-sum game where players spell valid English words on a 15x15 game board. Like a crossword puzzle, besides the initial move, all words must intersect with one or more words already on the board.

Each turn consists of the player putting a word onto the board and then tallying up the letters scores against any multipliers to get a positive integer score for that move. In general, long words and words with uncommon letters score highly. Once a player has run out of tiles or cannot make a move, the game ends. Whichever player has the highest cumulative score wins.

For each turn, a set of game state will be input, and a valid move will be output. See next page for a concrete example.

Metrics

Since this is a zero-sum game, the utility is formally $+\infty$ if the AI wins, $-\infty$ if the AI loses. The value function $V_{agent,opp}(s)$, the expected utility for the game at state s.

We will first explore using a minimax value function where we assume the opponent is trying to minimize our score.

As a first-pass, the AI will try to output the highest scoring word possible for that board during a turn. Eventually we would like to incorporate estimates of the opponents state and techniques to play adversarially that will minimize the opponent's score while maximizing our own. For example, if there are two potential locations for high-scoring words, we place the word in the location that will maximally impede the opponent in creating future high-scoring words. In summary:

Local metric: score for a single move; **Global metric:** end-game score

Baseline and Oracle

We created an oracle and a baseline to get a better sense of the upper and lower bounds of possible performance on the project.

The oracle searches through the scrabble dictionary to make a list of moves that are possible with any seven letters. This is the upper bound of any move possible for any scrabble player, human or otherwise, because the likelihood that a player would possesses the correct tiles for that move is quite low (and sometimes impossible).

Our baseline is given 7 random tiles and and makes the first possible move that it finds. Occasionally this method might produce an optimal move, but that would be highly unlikely.

Examples and Preliminary Data

Example Input:

```
00:[’ ’, ’ ’, ’ ’, ’ ’, ’ ’, ’ ’, ’ ’, ’ ’, ’ ’, ’ ’, ’ ’, ’ ’, ’ ’, ’ ’, ’ ’]
01:[’ ’, ’ ’, ’ ’, ’ ’, ’ ’, ’ ’, ’ ’, ’ ’, ’ ’, ’ ’, ’ ’, ’ ’, ’ ’, ’ ’, ’ ’]
02:[’ ’, ’ ’, ’ ’, ’ ’, ’ ’, ’ ’, ’ ’, ’ ’, ’ ’, ’ ’, ’ ’, ’ ’, ’ ’, ’ ’, ’ ’]
03:[’ ’, ’ ’, ’ ’, ’ ’, ’ ’, ’ ’, ’ ’, ’ ’, ’ ’, ’ ’, ’ ’, ’ ’, ’ ’, ’ ’, ’ ’]
04:[’ ’, ’ ’, ’ ’, ’ ’, ’S’, ’ ’, ’ ’, ’ ’, ’ ’, ’ ’, ’ ’, ’ ’, ’ ’, ’ ’, ’ ’]
05:[’ ’, ’ ’, ’ ’, ’ ’, ’T’, ’ ’, ’ ’, ’ ’, ’ ’, ’ ’, ’ ’, ’ ’, ’ ’, ’ ’, ’ ’]
06:[’ ’, ’ ’, ’ ’, ’ ’, ’A’, ’ ’, ’ ’, ’ ’, ’ ’, ’ ’, ’ ’, ’ ’, ’ ’, ’ ’, ’ ’]
07:[’ ’, ’ ’, ’ ’, ’U’, ’N’, ’I’, ’V’, ’E’, ’R’, ’S’, ’I’, ’T’, ’Y’, ’ ’, ’ ’]
08:[’ ’, ’ ’, ’ ’, ’ ’, ’F’, ’ ’, ’ ’, ’ ’, ’ ’, ’ ’, ’ ’, ’ ’, ’ ’, ’ ’, ’ ’]
09:[’ ’, ’ ’, ’ ’, ’ ’, ’O’, ’ ’, ’ ’, ’ ’, ’ ’, ’ ’, ’ ’, ’ ’, ’ ’, ’ ’, ’ ’]
10:[’ ’, ’ ’, ’ ’, ’ ’, ’R’, ’ ’, ’ ’, ’ ’, ’ ’, ’ ’, ’ ’, ’ ’, ’ ’, ’ ’, ’ ’]
11:[’ ’, ’ ’, ’ ’, ’ ’, ’D’, ’ ’, ’ ’, ’ ’, ’ ’, ’ ’, ’ ’, ’ ’, ’ ’, ’ ’, ’ ’]
12:[’ ’, ’ ’, ’ ’, ’ ’, ’ ’, ’ ’, ’ ’, ’ ’, ’ ’, ’ ’, ’ ’, ’ ’, ’ ’, ’ ’, ’ ’]
13:[’ ’, ’ ’, ’ ’, ’ ’, ’ ’, ’ ’, ’ ’, ’ ’, ’ ’, ’ ’, ’ ’, ’ ’, ’ ’, ’ ’, ’ ’]
14:[’ ’, ’ ’, ’ ’, ’ ’, ’ ’, ’ ’, ’ ’, ’ ’, ’ ’, ’ ’, ’ ’, ’ ’, ’ ’, ’ ’, ’ ’]
----|00| |01| |02| |03| |04| |05| |06| |07| |08| |09| |10| |11| |12| |13| |14|
```

Example output: {word:’PIZZAZZY’, score: 49, startPoint: (10, 6), orientation: ’v’}

We created 10 test scenarios for a simplified version of the game with no multiplier tiles, and asked the oracle and baseline to place a word. The oracle performed about 10x better than the baseline[1]: **Average oracle score:** 36.3, **Average baseline score:** 3.3

Approach

Next we plan to implement the turn-based framework of the game and add score multipliers. This should be straightforward.

Our approach at a brute-force search Oracle was relatively successful at optimizing for a local move. However, it was quite slow, and didn’t have constraints on the letters available like a real player. To speed things up we will investigate using alpha-beta pruning and search heuristics like A*. We’d like to see the average move take 10s or less.

Additionally, as mentioned earlier, we’d like to incorporate strategizing against the opponent into our AI. This is a bit vague, we expect we’ll have a better idea of what to do after completing the PacMan assignment.

Prior Work and Conclusion

By this point, most popular board games have AIs, and Scrabble is no exception. In 2002, Brian Sheppard created ’Maven’, which has won games against several world champions. Another is the 1988 Appel-Jacobsen Scrabble AI, optimized for speed rather than score.

We realize that a Scrabble AI is not groundbreaking, but this project should satisfy our goal of creating a substantial body of code that demonstrates our understanding of introductory AI.

- Maven: `http://www.sciencedirect.com/science/article/pii/S0004370201001667`
- Appel-Jacobsen `http://repository.cmu.edu/cgi/viewcontent.cgi?article=2580&context=compsci`
- A Recent (2012) Scrabble AI: `http://ceur-ws.org/Vol-860/paper16.pdf`
- An open-source amateur AI: `https://www.scotthyoung.com/blog/2013/02/21/wordsmith/`

[1]Note: the oracle is slow, it takes about 90 seconds to find a move. Because crafting example boards by hand and running the algorithm are time consuming, we kept the test set limited to 10.