Scrabble AI Project

Casey Ranft
4/23/20

Introduction:

Problem: Scrabble is a word battle between 2-4 people who wish to test their vocabulary and strategy. The problem with scrabble, like many games, it that it's only multiply player. For those who wish to play scrabble but don't have a partner, they are unable to play. An interesting solution to this is making a computer program to play against. To accomplish this, this project will use AI concepts and Collins Scrabble Words (2019) database to develop an intelligent computer scrabble opponent.

To understand how an intelligent scrabble opponent can be built, an understanding of how scrabble is played is essential. To give a general overview, it is played with multiple people, 100 letter tiles, and a scrabble board. The tiles each have a letter and a number on them. The number represents the value of a letter, more common letters (A,E) have a lower value and less common letters (Z,Q) have a higher value. The board is 15x15 with different spaces to place letters. Some of those spaces provide varying point boosters. Each player is given seven tiles from the existing 100. Words must be placed either vertically or horizontally and include an existing letter from the board. Players take turns either placing words on the board or swapping tiles from the exist 100 tile. Points are obtained by summing the value of the tiles and applying any point booster from the board. For more information on how to play visit https://www.wikihow.com/Play-Scrabble

For this project, some changes to the rule were made. For example, there are no blank tiles in the existing 100 which means the total tiles is actually 98. Also, players cannot swap tiles, they can pass or make a move. These changes occurred because of time constraints when developing.

Summary: The goal of this project is to create an intelligent computer opponent that scores highest points possible. Every turn the player tiles need to be combined somewhere on the board to produce points. More points can be earned by using rare letters and longer words but strategic placement of a single letter on the board can produce the maximum points available. That is why scrabble is though of to be not only a game of vocab but a game of strategy. For the computer to find the optimal move each turn, it needs to consider the board placement, letter value, and word length.

There are several ways to calculate an optimal move, but this project utilizes a variation on artificial intelligence concepts, specifically an Informed Search. Traditionally, Informed Search works by calculating a heuristic based off some knowledge known about the environment. The heuristic is what's used to determine which path/solution is better. Instead of the term heuristic, the term fitness will be used when referring to calculating word points. The fittest word determined by the computer will ultimately become the move on the board that will earn the highest points possible. Further sections will go more into depth on fitness algorithms.

Approach:

Resources: To develop a computer program that can intelligently play scrabble, it needs some way to come up with words. Both to create valid works and to find possible words given letters. In this case, Collins Scrabble Words (2019) text file was used. It provides about 250 words including their plurals for the computer to search through.

Python and its tools were used to create this project. The only special import was the regular expression (re) tool. Pattern matches based off an expression. A single expression is defined then

used to search a list/database and return matches. This tool was used to search the available words and match it with an expression that uses the player tiles and the board.

Fitness: To calculate the best possible fitness, the fitness algorithm is broken down into multiple steps.

1. The first thing the computer needed to do was find a single word on the board. This was done separately for word going across vs words going down. The computer traversed the board in order either going across or down and saving the tiles in a list until it reached a blank tile space. A blank space indicates no more letter in that word.

2. Determined possible words by writing a regular expression that included any combination of player tiles with the exact word found on the board. This was done by writing an expression that says any combination of player tiles in the front plus the exact word from the board plus any combination of player tiles at the end. Then the expression was used to find the matches in the words and return a list. Figure 1 shows the result of one such search for tile letters as A, A, E, B plus board letters as L. The resultant list of words is all combinations from the database that features the letters above.

```
playerLetters = ['A', 'A', 'E', 'B']
playerLetters = "".join(playerLetters)
boardRun = ['L']
boardRun = "".join(boardRun)


print("Test filterByRegex\n")
fil = Words.filterByRegex(playerLetters, boardRun, Words.words)
for f in fil:
    print(f)

print("Test filterByCount")
temp = Words.filterByCount(playerLetters, boardRun, fil)

for j in temp:
   print(j)
```

```
AAL
ABLE
AL
ALA
ALAE
ALB
ALBA
ALBE
ALE
BAAL
BAEL
BAL
BALE
BEAL
BEL
BLAE
EL
LA
```

Figure 1: Code and result for word matching using Regex

3. Filters through possible words and locates the position where the board word in the possible words. This is done to further filter the possible words by checking for validity. Meaning, it returns a list with possible words that will fit on the board and include the board word and at least one player tile.

4. Scoring. All invalid words are given a score of 0. Possible words must be scored by the possibility of them going across or them going down. The score it calculated by testing a possible words at a row column index on the board and summing its letter values then applying any board boosters. For example, AAA at row 0 and column 0 (0,0) would produce a score of 9. The value of letter A is 1 with a sums to 3, and the board booster at (0,0) is triple word score. With the booster applied, the score jumps to 9.

5. The methods to calculate the across fitness and down fitness only do so for a single word on the board. This means it find the best possible move for that specific row and column on the board but somewhere else on the board could still get a higher score. To get the fittest move possible, the getFittestMove method will iterate through every row and column combinations, calculating the best across move and best down move then comparing them. Figure 2 shows a demonstration of multiple words across the board but the move "LIMBY" is chosen because it produced the highest number of points.

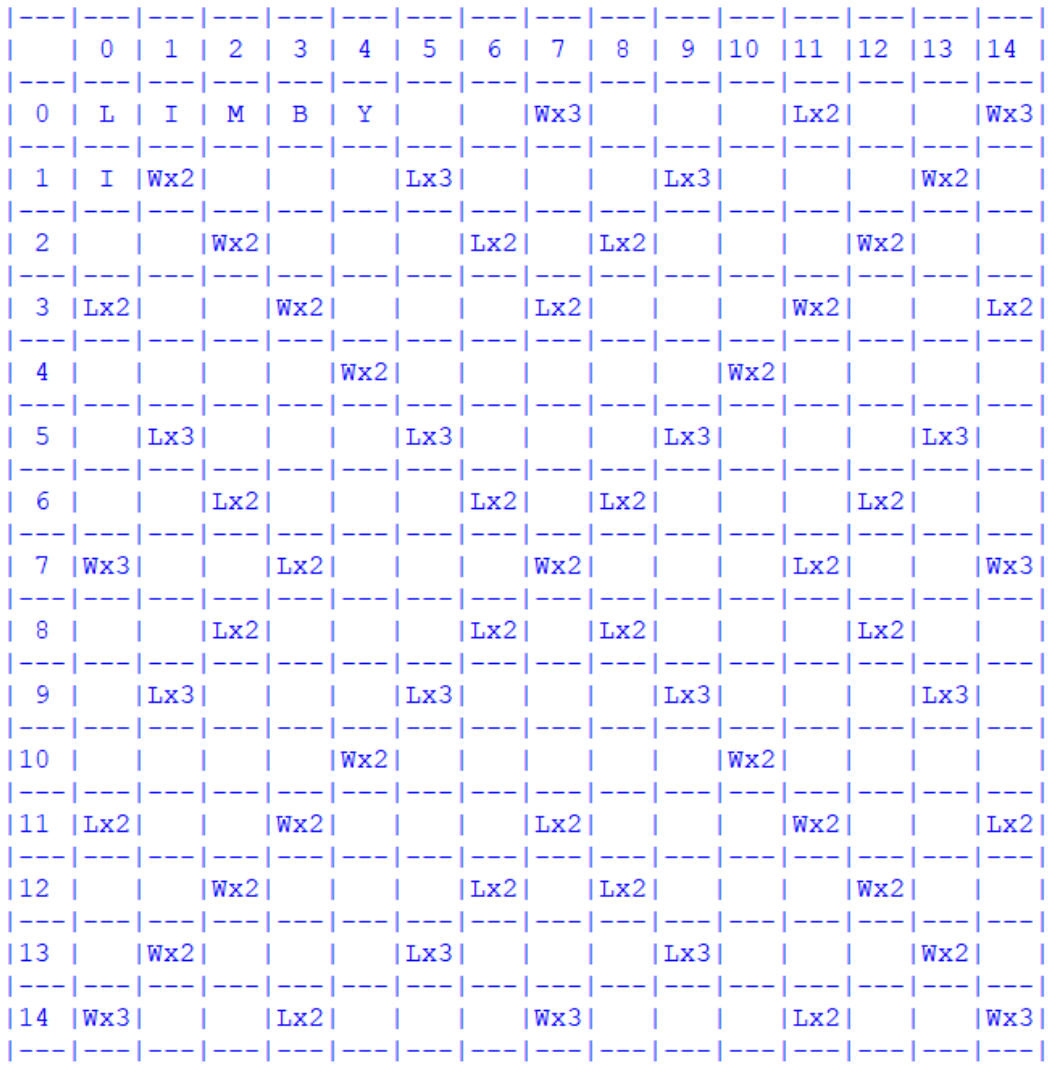

Figure 2: LI was places on the board in the top left corner and from available options, "LIMBY" was chosen

Code Structure:

Section	Description
Collins Scrabble Words (2019).txt	Text file contain about 250 words that needs to be read in to be used. To read the file, the absolute path needs to be inputted for current user
Words Dictionary	Handles all text file I/O and word filtering methods. Also contains a dictionary of all letter values, e.g. A : 1, Z: 10
Letter Tiles	Initializes 98 tiles to the correct amount using a tile quantities dictionary, e.g. A : 9 means nine A's were created. It then randomizes the tiles. It also contains a pull tile method that pops a single tile from the 98 tiles.
Board of Squares	Initialize the board (15x15) with an object called Square in each row and column. This object sets the tile to none and the score multiplier to its specified booster. The score multiplier signifies if that space is a double letter boost, triple word boost, normal, etc. It also contains a method to return a word on the board given a row and column going across. And another for down.
Player	Initializes the players tiles to seven and by removing them from the tiles inventory (98 tiles). It set the player type to computer or person. It also has methods to retrieve or remove a player's tile by letter. The lower part has a class called Move which defines the inputs for a player move, e.g. row, column, word, direction, etc.
Model	Holds all the information containing the board breakdown such as which players turn it is and initializing tiles and board.
Controller	Contains all the fitness logic, tile placement, validity checks, etc. It has all the fitness methods for the computer both across and down. Has generic place tiles methods for across and down to physically put the tiles on the board. It has score calculators for words going down and across. It also advances which players turn it is
View	Implements the view aspect of the games. Shows when where the letters are on the board. The row and column for defining a move to the human player. The score multipliers that are seen as 2xW which is double word. It prints out the game state and the scores.

Results

This project was success on some fronts and not others. The computer intelligence aspect of calculating the highest score wasn't the hardest part. The hardest was validating whether a move was allowed or not. The computer matching process to find valid words to play created words that would place letters off the board. For example, let's say the player tiles are A, A, B, C, C, D, D and the board had AD written in the top left corner going down. This means, AD is right along the edge. When my regular expression returns matches, it includes words like DAD or BAD which if placed on the board would be over the edge.

The problem is looking at one of the possible words and knowing how many letters are to the left or right. Knowing how many letter to the left the word goes means a check for validating left side which would check for placing a word over other tiles, off the board, and touching another tile. A check would also have to be performed for right side, up, and down.

A success with this project was the search time. Having to traverse through 250 words for every space on the board was about 60,000 searches. Keeping the computational time low makes the scrabble intelligence even more impressive.

Conclusion:

Summary: To optimize scrabble to it's fullest, there an extensive amount of computational logic required. The computer must try every possible combination of words and word booster and compare to a database in a timely manner. There is also an extensive amount of checks throughout the entire program just make sure the rules of scrabble are followed. Overall, not a bad attempt for intelligence scrabble opponent.

Lessons: To create a program that optimizes completely and 100% of the time for artificial intelligence takes a deep understand of algorithms and optimal cohesion. This project has many methods that call other methods and reference other classes.

Next Steps: In order to finish this project, someone would have to expand upon the game function of this project and more validity checks. Currently, there are some validity checks missing and optimizations in word searches that would greatly reduce the search time.

References

"Scrabble Tools." *Scrabble Tools - Collins Dictionaries*, www.collinsdictionary.com/scrabble/scrabble-tools/.

wikiHow. "How to Play Scrabble." *WikiHow*, WikiHow, 2 Apr. 2020, www.wikihow.com/Play-Scrabble.

Appendix:

Lesson Learned: artificial intelligence algorithms are only a 3rd of the project. The majority of the programming was formatting and game design. Something I tragically underestimated.