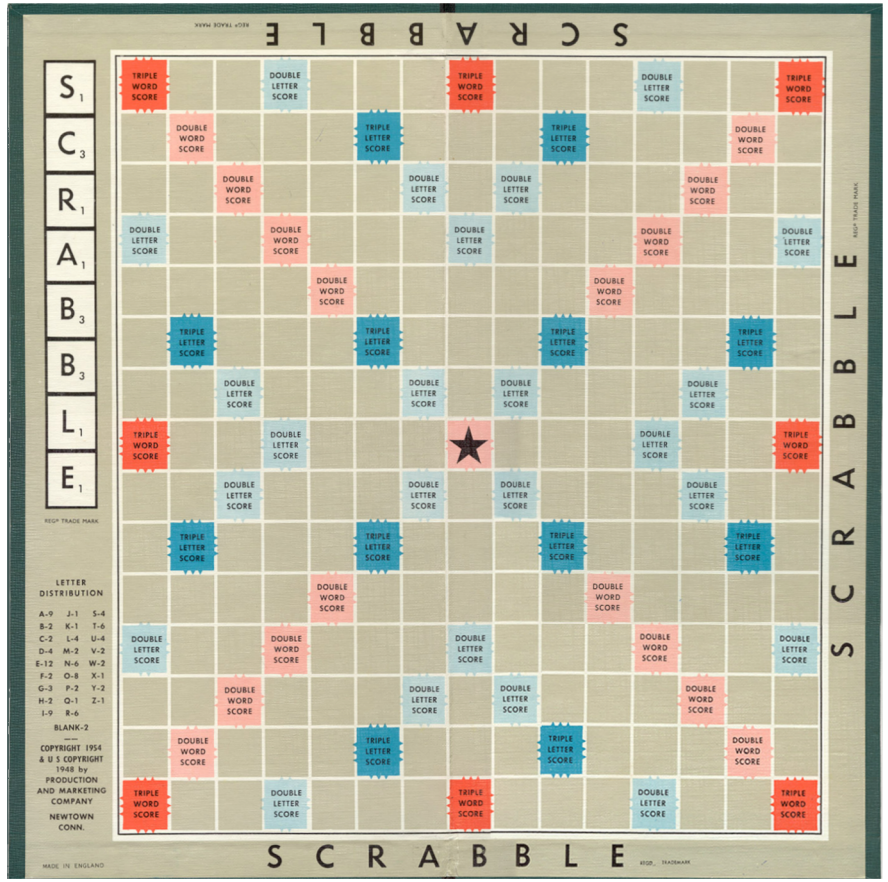


Scrabble

Group 14.B

Cameron Haupt

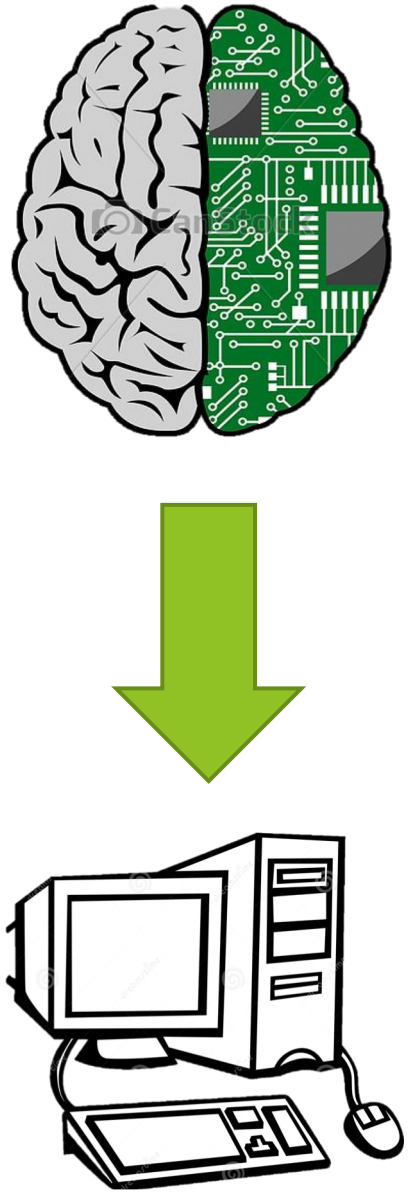
John Linn

Andrea Swanson

4/24/2018

Agenda

- Goals and Motivation
- Initial Approaches
- Final Approaches
- Score Evaluation
- Analysis



Goals and Motivation

- Design and build an ‘AI’ that will play Scrabble
- Provided 7 letters and and initial word on the Scrabble board the program finds and plays the best word
- Efficiency Constraints
  - Preprocessing time: << 5 Minutes
  - Time to find word: << 1 Second
  - Memory Used: ~150MB

Initial Approaches

Supporting Structures

- The Dictionary is stored in a trie to decrease searching time when validating words
  - The root is an empty node and the first level of children is a node containing a character and string
  - Each level consists of a node where the character is the next letter in a dictionary word and the string is null until the characters up the tree form a word
- The possible enumerations were stored in a HashSet to avoid storage of duplicate words
- Arrays and ArrayLists used for the storage of words and available letters

Initial Approaches

Supporting Algorithms

- The possible enumerations from the 7 available letters and a letter from the initial word was the main algorithm required for success
  - The first attempt had a Big-O time complexity of at least O(n*n!). It used large amounts of time and space
  - Later attempts utilize recursive methods and the java.util.HashSet import to decrease space and stop the storage of duplicate words
- Tree traversal to check validity of word once word is enumerated
- The traversal uses Breadth First Traversal where it checks a level by level going from left to right
- The algorithm checks for the first level then drops down to its decedents and checks left to right once again
- If the arrangement of letters is an available path return true

Initial Approaches

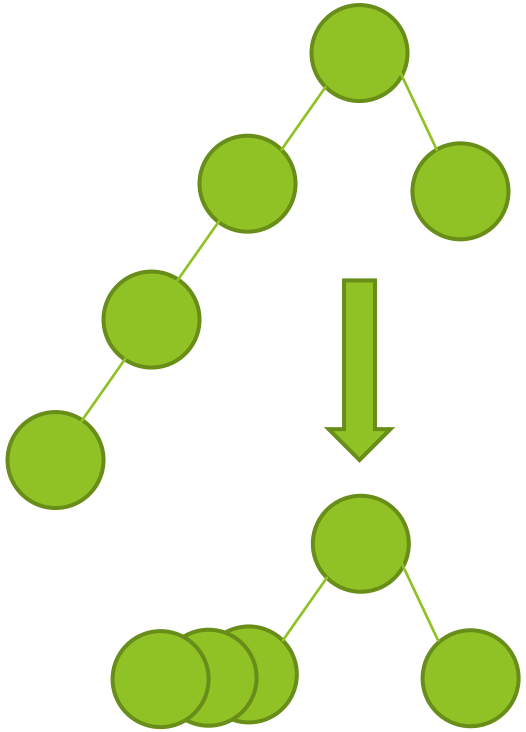
Ideas Generated and Borrowed

- Dictionary
  - First approaches for storing the dictionary included: Linear lists, Trie, Compressed, Trie, regular tree,
  - Some research lead to the idea of sorting the letters in each word and using anagrams to to store the words - not implemented
    - Trie implementation adapted from text section 13.3
- Enumerating words
  - First thoughts were to use 8 nested for-loops to iterate through the letters
  - Time taken was ~180 seconds so new ideas were generated
  - The implemented is more efficient by checking validity before saving and using a set

Final Approaches

Changes to the Structures

- Compressing the Trie
  - The compressed trie would use less space and enable faster searching
  - Compression occurs when a node only has one child
  - The child node would then be concatenated with the parent node
- Problems with implementation
  - Finding valid words after compression would be difficult if one word was an extension of another
  - Entire node implementation would need to be changed for success



Final Approaches

Changes to Algorithms

- Enumeration Algorithm
  - Find more valuable enumerations
  - Find less invalid and low point words
- Score calculation
  - Now finds individual letter scores and stores a max word mid-enumeration
  - Also accounts for multipliers on the board when determining a max word
- Other changes in algorithms ensure that a valid word is always played in the boundary of the 15x15 board

Final Approaches

Ideas Generated and Borrowed

- The final approaches were generated based on feedback from the initial submission and further review or program performance
- Trie
  - When an invalid word is generated then remove that path from the trie so it can not be used again, speed up word validation
- Points
  - Evaluate based on overall word score with board multipliers
  - Find and store multiple words with the best scores incase the top word is not inbounds

Evaluation

Initial

- Seeds 1 through 15 used and data was recorded for trials
- Averages
  - Time: .487 sec
  - Memory: 547 MB
  - Points: 10.3
  - Performance: .0199

Initial Submission				
Seed	Time	Memory [Mb]	Points	Performance
1	2.76E-01	551	16	0.0207
2	1.62E-01	467	10	0.0115
3	2.99E-01	563	30	0.0693
4	2.28E-01	569	20	0.0355
5	2.00E+00	600	0	0
6	2.00E+00	600	0	0
7	2.08E-01	569	19	0.0336
8	3.20E-01	563	0	0
9	2.90E-01	540	0	0
10	2.45E-01	519	34	0.1023
11	2.10E-01	488	0	0
12	2.08E-01	491	0	0
13	2.44E-01	569	0	0
14	3.00E-01	563	12	0.011
15	3.09E-01	563	14	0.0148
Average	4.87E-01	547.67	10.3	0.0199

Evaluation

Final

- Seeds 1 through 15 used and data was recorded for trials
- Averages
  - Time: .289 sec
  - Memory: 430MB
  - Points: 23.3
  - Performance: .0590

Final Submission				
Seed	Time	Memory [MB]	Points	Performance
1	2.81E-01	438	18	0.0292
2	2.81E-01	426	16	0.0265
3	3.12E-01	426	26	0.0585
4	2.34E-01	427	20	0.04
5	2.96E-01	426	15	0.0198
6	3.43E-01	435	13	0.0138
7	2.81E-01	438	19	0.033
8	3.59E-01	426	28	0.0267
9	3.12E-01	427	22	0.0419
10	2.81E-01	426	60	0.3284
11	2.18E-01	427	36	0.1341
12	2.18E-01	426	22	0.0494
13	2.50E-01	438	16	0.0245
14	3.43E-01	438	24	0.0469
15	3.28E-01	438	15	0.0118
Average	2.89E-01	430.8	23.3	0.0590

Analysis

Initial

- Points
  - Error in calculation or points, only works sometimes
- Time
  - Slower – would find every enumeration and then find max word
- Memory
  - More memory – more nodes stored in the trie and more enumerations were accounted for

Final

- Points
  - Finds valid words and chooses based on a combination of word length and letter values
- Time
  - Faster – no longer finds and stores all of the possible words
- Memory
  - Less memory – Unused nodes were removed and less possible enumerations were made

Further Improvements

- Improve the enumeration algorithm to be more efficient and while also finding more words
- Improve the algorithm for finding the best possible word
  - Account for all multipliers on the board
  - Find more possible orientations and placements of letters
- Use memory more efficiently
  - Less unnecessary generation of duplicates
  - Use dynamic programing to save chunks that get used multiple times

Questions?

John Linn