Brief Observations - scrabble

Main Components

Board:

- The board is 13 `squares` in order to minimize complexity; the rules state the first move must cover the middle square and with a max word size of 7 `tiles` the board only needs to be 13 elements
- Included is a `set_cell` function to place the word, and `display` to show the state of the board

Agent:

- The Agent is responsible for arriving at the goal. It instantiates the `board` the `rack` and also contains a `display` function for displaying the rack.
- The `solve` functions runs through the `successor` module and solves the board:
    - `generate_anagrams` instantiates the `TrieGuy` which is a prefix-tree that holds the official scrabble dictionary. In this data structure all words with common roots share common nodes. This significantly reduces the search time to find a legal word.
    - A blank tile (`_`) can be any letter but some letters are better than others. The order of letters chosen is weighted as 60% score and 40% commonality in the english language
    - As anagrams are generated the function uses `estimate_score` to compare the estimated maximum score of the rack to the estimated score of a given anagram; if the word is at least 40% of `max_score` it is added to the `anagram_list`. This eliminates the need to `set_cell` and actually calculate the score for each potential word & significantly reduces the `anagrams` list

  - The final `anagrams` list is then iterated over and the `find_best_move` decides where to place it using these rules:
    - If the word is less than 4 characters, it must start on the center tile
    - Put the highest scoring letter on the double letter tile such that a legal move is produced
  - The scores resulting from `find_best_move` are compared and the highest scoring move is chosen
  - Results are displayed to the user

Main

- `main` kicks off the application by reading in a `racks` file that is a set of 7 scrabble letters. Each set is separated by a new line.
- Each `rack` is accompanied by an agent who is instantiated and the `best_move` is found.

Observations

At no point did it really seem like my machine was struggling on this one. The previous application - the sudoku solver - seemed to work the cpu a bit harder. I think this is because of my `TrieGuy` . I never tried to iterate over the dictionary, which was easily the biggest data structure involved. My first iteration of the application reads the dictionary into the prefix-tree.

With that data structure set up I don't think any other part of my application really needed to be that efficient to produce a result in just a few seconds.

My other observation is how I choose the `_` letter. At first I thought I should iterate over the alphabet by commonality, and then I thought I should iterate by score. Then I thought of the super clever idea of giving each letter a weighted score based a 60/40 split of score/commonality respectively. After finishing the app I realized… I don't think it matters. We iterate over the whole alphabet anyway, it's not like by doing this we are eliminating letters to check.

The real heurstic is reducing the anagrams list. By setting a 80% threshold on the estimated points of the word we significantly reduce the number of anagrams, and therefore reduce the time to a solution. However after more testing I needed to lower than number significantly, to 40%. The high threshold was occasionally causing no anagramst to return, mostly with racks with Z's because of the high potential score.

As described above there is a chance this will produce the wrong answer. If a rack with a blank tile can produce a higher score with all 7 letters than it can with 6 letters its possible the wrong highest score will be returned.

This would be essentially flawless if you removed the bonus score altogether. We could also adjust the application to math that out but it would lose a good chunk of efficiency as we would need to set and calculate every anagram.

So for sure room for improvement here but we have a `Trie` tree and good heuristics for the `anagrams` and I think this is a good solution.

https://github.com/godfreypj/intro-to-ai