Team: Random2

Our bot strategy is comprised of two parts:

- Generate list of possible legal words
- Evaluate each legal word to determine the best move

To generate a list of the possible legal words to play we have used the GADDAG data structure described in the paper the prof has put online. Aforementioned data structure contains the possible ways to create words in a fashion similar to the Trie used by the prof to implement the dictionary. Building a semi-minimised GADDAG using all the possible words lead to a memory footprint of over 1.5 Gb and a substantial initialisation-time requirement. Since our tests showed that plays of length greater than 7 were extremely rare we have halved the GADDAG size and reduced its initialisation time by a factor of ten by using only the words of length less than a certain threshold.

To allow for a fast retrieval of such words, each time is called on a specific square, it starts to build the word first going backward from the start position and then recursively follows pointers that lead it both backward and forward, saving all the valid words it encounters. While traversing the GADDAG the bot's rack and board are checked to ensure that the necessary words to build the words are available. Furthermore the words created by parallel/multiple intersecting placements should be checked to be valid words, but since that would have meant to reimplement the same code we already have to check words (using professor's code from inside the bot turned out to be too messy) we have simplified the process only allowing for words that do not run parallel or intersect with other words.

The evaluation of the best move is comprised of two possible alternatives:

- If there are legal words to play the heuristic used has been highest score / longest word
- If there are no legal words to play then a random number of tiles, between 2 and 5, are exchanged.

Possible improvements to our strategy would be:

- to improve the minimisation of the GADDAG, thus allowing us to use the whole dictionary in the legal words generation part
- use our previous code to correctly check for parallel/multiple intersecting words.
- fix the unexpected bias towards the plays made in the bottom-right sector of the board