Joseph Slote

11/2/12

Design Document for Poggle (Python Boggle)

My plan is to create an interactive Boggle game, allowing the player to type in as many words as he can in the three-minute time limit. There will be a computer opponent with adjustable difficulty. I plan to make it graphical but will first work on the more important components of the program and only implement graphics if there is time.

	1	2	3	4
1	A	D	S	U
2	J	K	Y	N
3	D	I	N	L
4	E	E	F	G

The Rules of Boggle:

Players have three minutes to find words in a four-by-four (or five-by-five) letter grid, similar to the one on the right. Unlike a word search, the letters in the words do not have to be in a straight line. All that is required is that each letter touches the one before it and the one after it in the word, and that the same location on the board isn't used twice in the same word. For instance, "DINE", "KIN", and "KNEED" are all acceptable words on this board. During scoring, players remove any words from their lists that are not unique to their own list. Each remaining word is assigned a score based on its number of characters: score = n – 2, so only words of three letters of more are counted.

Basic Program Functionality:

1. Generate board
2. Solve board for all possible words
3. Run game with 3:00 timer
    1. Take user's input
    2. Slowly generate a word list to act as an opponent.
4. Check user words against solved board list
5. Score both user's and opponent's words

Components of the game:

- **Die Object** – for recording position and randomizing grid of letters. The original Boggle's set of dice have particular groups of six letters which allow for fairly playable boards. I will use these sequences rather than randomly generating a matrix of letters in order to avoid bad boards, for instance situations in which there are no vowels.
- **Board Solver** – This algorithm (most likely recursive) will implement a depth-first search as follows

```
1.  for startchar in boggle grid
2.      for each char touching startchar
3.          check if startchar+char is in wordtree
4.              for each nextchar touching char
```

```
                check that it hasn't been used yet in the word
                check if startchar+char+nextchar is in wordtree
                # ^ would look like wordtree[starchar][char][nextchar]
                    for each...
                    # continue this recursion until there are no
                    # possible paths in the wordtree
```

- **Word Tree** - an easily searchable collection of acceptable words. I plan to use a dictionary of dictionaries of dictionaries of …of dictionaries of words.

  E.g. This dictionary contains the words 'an', 'and', and 'ant'

```
wordtree = {
    'a':{
        'n':{
            True:'an'
            'd':{
                True:'and'
                },
            't'{
                True:'ant'
                }
            }
        }
    }
```

  To check if a word is in the dictionary, one would write something like:

```
try wordtree['a']['n']['d'][True]
```

  Advantages:

- easy to implement a search
- very speedy to search recursively.

  However, I'm not sure if this is the best way to organize the word list because of the amount of memory it would take up.

- **Word List** – This is a 1.66 Mb text file containing all the English words that have been deemed acceptable for use in Scrabble. I will remove all words less than three characters and longer than sixteen characters and then process it into the form described above, pickle it, and store it for later use.