Variables and Purpose:

Player: Player 1 or 2

Model1: List of scores

numLetters: number of letters left for play

P1Rack,P2Rack:Racks of 2 players

P1score,P2score:Scores of 2 players

placePos: Position where computer word is to be placed

helpRack: computer player Rack

helpBoard: Letter in board which computer strings to

firstmove: checks for firstmove

replacemode: whether in replace mode or not

Trie Node Tester(details later)

Scrabble tile images(from net)

Selected: Letter currently selected

JButton Array for board(Tile)(15X15)

JButton Array for Tile(RTile)(7X1)

Added several initialisation lines in block with premade initComponents() function.

Mostly setting the look of the GUI and adding Action Listeners and filling the letters

Functions:

checkAcceptable(): checks whether a “structurally” valid word has been made on board. Rules for this taken from net and implemented meticulously. Returns Boolean. Also sets Boolean alongRow accordingly whether main word formed is vertical or horizontal.

score(String): Calculates score of word by simply adding scores of letters. Board placement taken care of in other ways.

Wordmakescore(): Takes care for finding net score of words formed. For double letter, adds same letter again to word. For word multiplier, adds entire word again. Finds all words made with main word as well as branching words. Adds all scores together and returns final score.

AcceptButton Pressing: calls checkAcceptable(). Changes GUI accordingly. Calls makeWordscore. Switches player and engages computer player via calling compPlayer() function.

PassButton Pressing: simply changes player.

ReplaceButton Pressing: Turns on replacemode. Disables other buttons. Clicking RTiles now make them blue indicating selection for replacement. On clicking again, replaces tiles with random tiles from letterset. Adds back rack tiles to letterset. Switches player.

Everytime player is switched anywhere, compPlayer is called if required.

compPlayer function always made to run in separate thread. Separate thread required else GUI updations and inputs can’t take place simultaneously. Details on compPlayer later afer Trie.

Data Structures implemented: Trienode

Trienode made in separate Java class called Trie.java. Trie.java class description:

Contains a TrieNode data structure for constant time searching. Loads word from txt file “words.txt”

This TrieNode also helps in finding words to be made by computerPlayer. Details later.

Another structure in Trie.java is a simple Linked list with attributes word,wordScore,possible,horizontal,start.

word is the word. wordScore is word Score. Possible refers to whether the word fits at position and orientation defined by start[2] and horizontal attributes.

Delete function in Trie.java removes the TrieNode from memory and replaces with new TrieNode.

compPlayer() description:

Computer takes all letters in Rack and one extra letter from any letter on board. Makes all possible English words with these letters which necessarily contain the letter taken from board. This is done via a recursion that checks in the TrieNode if next letter exists at each iteration. So not mere permutations. Much more efficient. Now for each word, horizontal and vertical fitting is tried. This is done via the fits() function. It returns an integer corresponding to vertical fit, horizontal fit or no fit.

rmRack takes the letters computer uses out of the rack.

Put(crawl,endput): puts the word denoted by crawl into board using the start[] and horizontal attributes of wNode(). If endput, this is the word computer has decided to play thus letters are removed from rack via rmRack.

putCheck(): puts word via Put() and changes wordscore attribute to actual word score by calling wordmakescore. Check acceptable is called first so wordmakescore knows horizontal or vertical. Possible attribute is also modified accordingly.

compPlayer has a sorting implemented to sort the word in descending order according to the rectified wordscores.

According to difficulty level set, bottom one third words, middle one third or top one third words are used for making words.

NetBeans feature of RadioButton group used to make sure only one difficulty level is set. Thread.sleep() makes computer slower so you can learn from computer versus computer matches.

Algorithm shortcoming: Can’t hook with multiple board letters unless all except one used board letters exist in rack too. Still, Nightmare difficulty makes quite high scoring words and all other kinds of words are taken into account like multiple branching word forming. Sprites not done due to lack of time.