**Problem input/command line interface:**

Your program should:

* Accept three (3) command line arguments, so your code could be executed with
  + ALGO specifies which algorithm the computer player will use:
    - 1 – MiniMax,
    - 2 – MiniMax with alpha-beta pruning,
  + FIRST specifies who begins the game:
    - X
    - O
  + MODE is mode in which your program should operate:
    - 1 – human (X) versus computer (O),
    - 2 – computer (X) versus computer (O),

**Program details:**

Specific program details:

* The Tic-Tac-Toe game board is represented by 3 x 3 grid with cells numbered as follows

|  |  |  |
| --- | --- | --- |
| 1 | 2 | 3 |
| 4 | 5 | 6 |
| 7 | 8 | 9 |

* Possible moves/actions for both players match cell numbers (if a player wants to place an ‘X’ in the middle of the board, the move/action is ‘5’,
  + First is the information who makes the first move as specified by a command line argument,
  + Mode is the game mode as specified by a command line argument,
* If the game mode is human versus computer display an empty board first and prompt the user to pick the move (see below)

| |

---+---+---

| |

---+---+---

| |

* When it is human player turn, your program should display the following prompt:

X’s move. What is your move (possible moves at the moment are: <list of possible moves> | enter 0 to exit the game)?

where: <list of possible moves> is a sorted list of all available moves at the moment, for example, if the board arrangement is:

|  |  |  |
| --- | --- | --- |
| X |  |  |
| O | O | X |
|  | O |  |

and it is X’s move, the prompt should be:

What is your move (possible moves at the moment are: 2, 3,7,9) | enter 0 to exit the game)??

If the user enters anything other than 0 / valid move number (0 should terminate the game) your program should repeat the prompt above.

Once the user enters a valid move, display the updated game board on screen.

* + X WON or O WON
  + TIE
  + X LOST or O LOST

**Analysis:**

Play nine (9) human versus computer (**using both algorithms**) games, each starting with a different move. Count the total number of expanded nodes (sum of expanded nodes for every computer move) and report them in the table below.

|  |  |  |
| --- | --- | --- |
| Your (X) First move | Computer (0) with MiniMax algorithm. Total (for every move) number of generated nodes | Computer (0) with MiniMax with alpha beta pruning algorithm. Total (for every move) number of generated nodes |
| 1 | 59696+920+56+2=60674 | 4081+257+44+2=4384 |
| 2 | 63896+1048+48+2=64994 | 6053+367+34+1=6455 |
| 3 | 59696+920+56+2=60674 | 4941+362+51+2=5356 |
| 4 | 63896+892+38+1=64827 | 7633+301+30+1=7965 |
| 5 | 55496+926+46+2=56470 | 6130+206+38+2=6376 |
| 6 | 63896+892+38+1=64827 | 9777+231+25+1=10034 |
| 7 | 59696+928+39=60663 | 5177+493+48+2=5720 |
| 8 | 63896+1012+44+2=64954 | 11522+822+37+2=12383 |
| 9 | 59696+920+56+2=60674 | 6192+362+51+2=6607 |

What are your conclusions? Which algorithm performed better? Write a short summary below.

|  |
| --- |
| **Conclusions** |
| The min max algorithm with pruning performs better than the regular min max algorithm.  The alpha-beeta reduces the number of nodes explored. On an average it just explores only 10 to 17 percent of nodes explored by the regular minimax algorithm. The alpha beeta prunes the irrelavant branches of the search tree. This becomes a significant number especially in a complex search trees like chess, tic tac toe etc. This reduce in the node exploration results in preserving the computations.  The alpha beeta results in early cutoffs in the search tree. If it encounters an undesirable branch it stops exploring that particular branch tree and moves to the next. |