A Mini Project Report

Subject

High Performance Computing

On

TIC TAC TOE using OPENMP with C

By

MceboPateguana (PRN-71700366D)

Prashant Sharma (PRN-71700476H)

ThawatchaiYango (PRN-71700619M)

MAEER'S MAHARASHTRA INSTITUTE OF ENGINEERING

DEPARTMENT OF COMPUTER ENGINEERING

*** 2020-2021 ***

Acknowledgement

The satisfaction that accompanies the successful completion of this project would be in complete without the mention of the people, who made it possible, without whose constant guidance and encouragement would have made efforts go in vain. I consider myself privileged to express gratitude and respect towards all those who guided us through the completion of this project.

Last but not the least, we wish to thank our parents for financing our studies in this college as well as for constantly encouraging us to learn engineering. Their personal sacrifice in providing this opportunity to learn engineering is gratefully acknowledged.

Abstract

OpenMp– OpenMP is an Application Program Interface (API), jointly defined by a group of major computer hardware and software vendors. OpenMP provides a portable, scalable model for developers of shared memory parallel applications. The API supports C/C++ and Fortran on a wide variety of architectures. This tutorial covers most of the major features of OpenMP 3.1, including its various constructs and directives for specifying parallel regions, work sharing, synchronization and data environment. Runtime library functions and environment variables are also covered.

OpenMP is an industry-standard, platform-independent parallel programming library built into all modern C and C++ compilers. Unlike complex parallel platforms, OpenMP is designed to make it relatively easy to add parallelism to existing sequential programs, as well as write new parallel programs from scratch. In this fun, interactive, hands-on workshop, participants will use OpenMP to learn about a variety of parallel programming concepts, including single program multiple data (SPMD) execution, fork-join threading, parallel loops, parallel blocks, atomic execution, mutual exclusion, and others.

Table of Contents

1. Introduction

Tic-tac-toe also known as noughts and crosses is a paper and pencil game for two players, who take turns marking the spaces in a 3 x 3 grid traditionally. The player who succeeds in placing three of their marks in a horizontal, vertical or diagonal row wins the game. It is a zero-sum of perfect information game. This means that it is deterministic, with fully observable environments in which two agents act alternately and the utility values at the end of the game are always equal and opposite. Because of the simplicity of tic-tac-toe, it is often used as pedagogical tool in artificial intelligence to deal with searching of game trees. The optimal move for this game can be gained by using minimax algorithm, where the opposition between the utility functions makes the situation adversarial, hence requiring adversarial search supported by minimax algorithm with alpha beta pruning concept in artificial intelligence.

1.1. Purpose

The purpose of this document is to give a detailed description of Tic Tac Toe using open mp. It will illustrate the purpose and complete declaration for the development of system. This document is primarily intended to anyone who wants to get an overview of how Tic Tac Toe using open mp works its outcomes and possible usages in the future.

1.2. System Overview

In this mini project, we are going to implement a Tic Tac Toe using open mp. In the end, we are going to build a show how Tic Tac Toe works and show some playing.

1.3. Problem Statement

- The game is to be played between two people (in this program between HUMAN and COMPUTER).
- One of the player chooses ‘O’ and the other ‘X’ to mark their respective cells.
- The game starts with one of the players and the game ends when one of the players has one whole row/ column/ diagonal filled with his/her respective character (‘O’ or ‘X’).

- If no one wins, then the game is said to be draw

1.4. Goal & Vision

In this mini project, our goal is to build the game Tic Tac Toe by using open mp for parallel programming.

2. Requirements Specification

2.1. User Characteristics

User of the Windows Operating System that wants to use the program for writing a number digit in the program by machine learning recognition.

2.2. Software Requirements

2.2.1. Windows 10 Operating System
2.2.2. Code Blocks with C++
2.2.3 OpenMP Library

2.3. Hardware Requirements

2.3.1. Desktop or laptop PC
2.3.2. At least 4GB RAM
2.3.3. At least 3GB Free Space

3. Design

3.1. Architecture of Tic Tac Toe

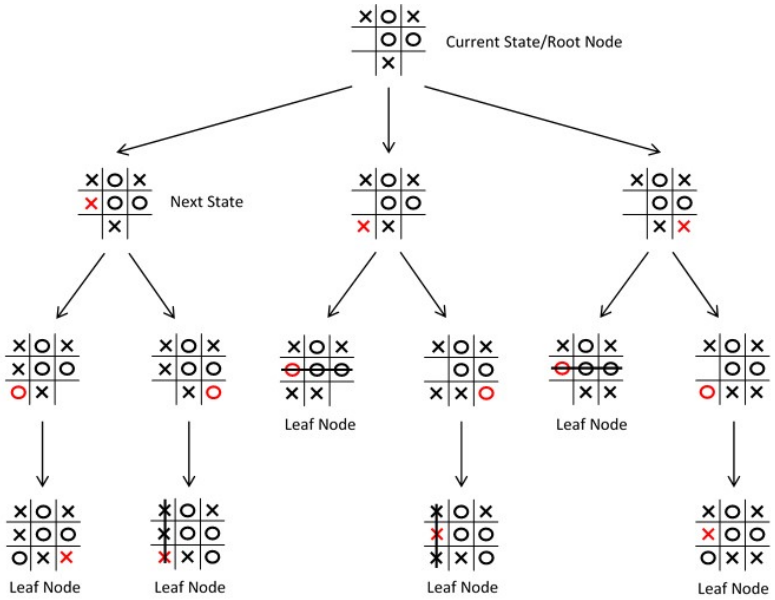


Figure 1. Architecture of Tic Tac Toe

3.2. Block Diagram for Tic Tac Toe

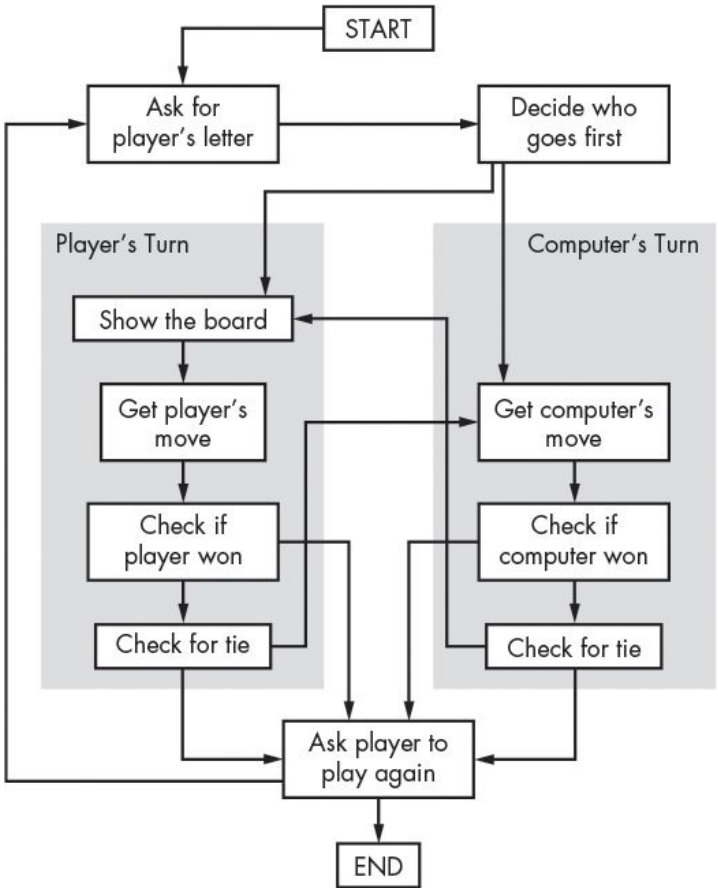


Figure 2. Block Diagram for **Tic Tac Toe**

3.3. OpenMp Architecture

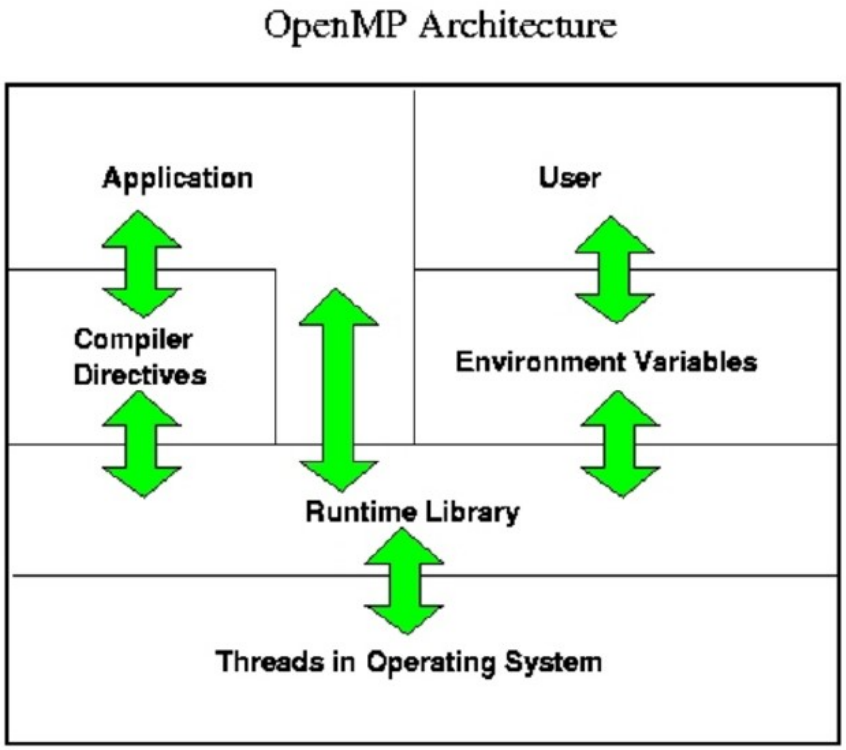


Figure 3. **OpenMp Architecture**

4. Coding for the program


```
#include "stdio.h"
#include "stdlib.h"
#include "string.h"
#include<comp.>

/* text colour code declarations */
#define KNRM  "\x1B[0m"
#define KRED  "\x1B[31m"
#define KGRN  "\x1B[32m"
#define KYEL  "\x1B[33m"
#define KBLU  "\x1B[34m"
#define KMAG  "\x1B[35m"
#define KCYN  "\x1B[36m"
#define KWHT "\x1B[37m"

/* enum int const chars */
enum { NOUGHTS, CROSSES, BORDER, EMPTY };
enum { HUMANWIN, COMPWIN, DRAW };

/* var definitions */
const int Directions[4] = {1, 5, 4,  6};
// times by -1 to
//go opposite direction
const int ConvertTo25[9] = { /* positions in 25 array*/
6,7,8,
    11,12,13,
    16,17,18,
};
const int InMiddle = 4;
const int Corners[4] = { 0, 2, 6, 8 };

int ply = 0; // how many moves deep into tree
int positions = 0; // no of pos searched
int maxPly = 0; // how deep we have went in tree

int GetNumForDir (int startSq, const int dir, const int *board, const int us) {
    int found = 0;
    while (board[startSq] != BORDER) { // while start sq not border sq
        if(board[startSq] != us) {
    break;
    }
    found++;
startSq += dir; 44
}
    return found;
}

int FindThreeInARow(const int *board, const int ourindex, const int us){

    int DirIndex = 0;
    int Dir = 0;
    int threeCount = 1;

for(DirIndex = 0; DirIndex <4; ++DirIndex) {
        Dir = Directions[DirIndex];
        threeCount += GetNumForDir(ourindex + Dir, Dir, board, us);
        threeCount += GetNumForDir(ourindex + Dir * -1, Dir * -1, board, us);
        if (threeCount == 3) {
            break;
        }
        threeCount = 1;
        }
        return threeCount;
}

int FindThreeInARowAllBoard(const int *board, const int us) {
// after move made
int threeFound = 0;
    int index;
    for(index = 0; index < 9; ++index) { // for all 9 squares
        if(board[ConvertTo25[index]] == us) { // if player move
            if(FindThreeInARow(board, ConvertTo25[index], us) == 3) {
                threeFound = 1; // if move results 3 in row,confirm
                break;
            }
        }
    }
    return threeFound;
}

int EvalForWin(const int *board, const int us) {
// eval if move is win draw or loss
        return 1; // player win confirmed
```

```
85        if(FindThreeInARowAllBoard(board, us ^ 1) != 0) // opponent win?
86            return -1; // opp win confirmed
87    return 0; 88
      }
89
90    int MinMax (int *board, int side) {
91    // recursive function calling -   min max will call again  and again
92    // through tree - to maximise score
93    // check if there is a win
94    // generate tree for all move for side (ply or opp)
95    // loop moves , make move, min max on move to get score
96    // assess best score
97    // end moves return bestscore
98
99    // definitions
100       int MoveList[9]; // 9 pos egs on board
101       int MoveCount = 0; // count of move
102       int bestScore = -2;
103       int score = -2; // current score of move
104       int bestMove = -1; // best move with score
105       int Move; // current move
106       int index; // indexing for loop
107
108       if(ply > maxPly) // if current pos depper than max dep
109            maxPly = ply; // max ply set to current pos
110       positions++; // increment positions, as visited new position
111
112       if(ply > 0) {
113           score = EvalForWin(board, side); // is current pos a win
114           if(score != 0) { // if draw
115               return score; // return score, stop searching, game won
116           }
117       }
118
119       // if no win, fill Move List
120       for(index = 0; index < 9; ++index) {
121           if( board[ConvertTo25[index]] == EMPTY) {
122               MoveList[MoveCount++] = ConvertTo25[index]; // current pos on loop
123           }
124       }
125
126       // loop all moves - put on board
127       for(index = 0; index < MoveCount; ++index) {
128           Move = MoveList[index];
129           board[Move] = side;
130           ply++; // increment ply
131           score = -MinMax(board, side^1); // for opposing side
132   if(score > bestScore) { // if score is best score (will be for first move)
133               bestScore = score;
134               bestMove = Move;
135   }
136
137   /* OMP parallel section segment - each section in the parallel sections
138   section is executed in parallel */
139
140   #pragma omp parallel sections
141   {
142    #pragma omp section
143      {
144       // undo moves
145           board[Move] = EMPTY; // else clear board
146           ply--; // decrement ply
147       } // end this parallel section
148
149    #pragma omp section
150      {
151       // tackle move count is 0 as board is full
152       if(MoveCount==0) {
153           bestScore = FindThreeInARowAllBoard(board, side);
154       }
155         } // end this parallel section
156        } // end parallel sections segment
157
158       // if not at top at tree, we return score
159       if(ply!=0)
160           return bestScore;
161       else
162           return bestMove;
163       }
164          }
165
166   void InitialiseBoard (int *board) { /* pointer to our board array */
167       int index = 0; /* index for looping */
```

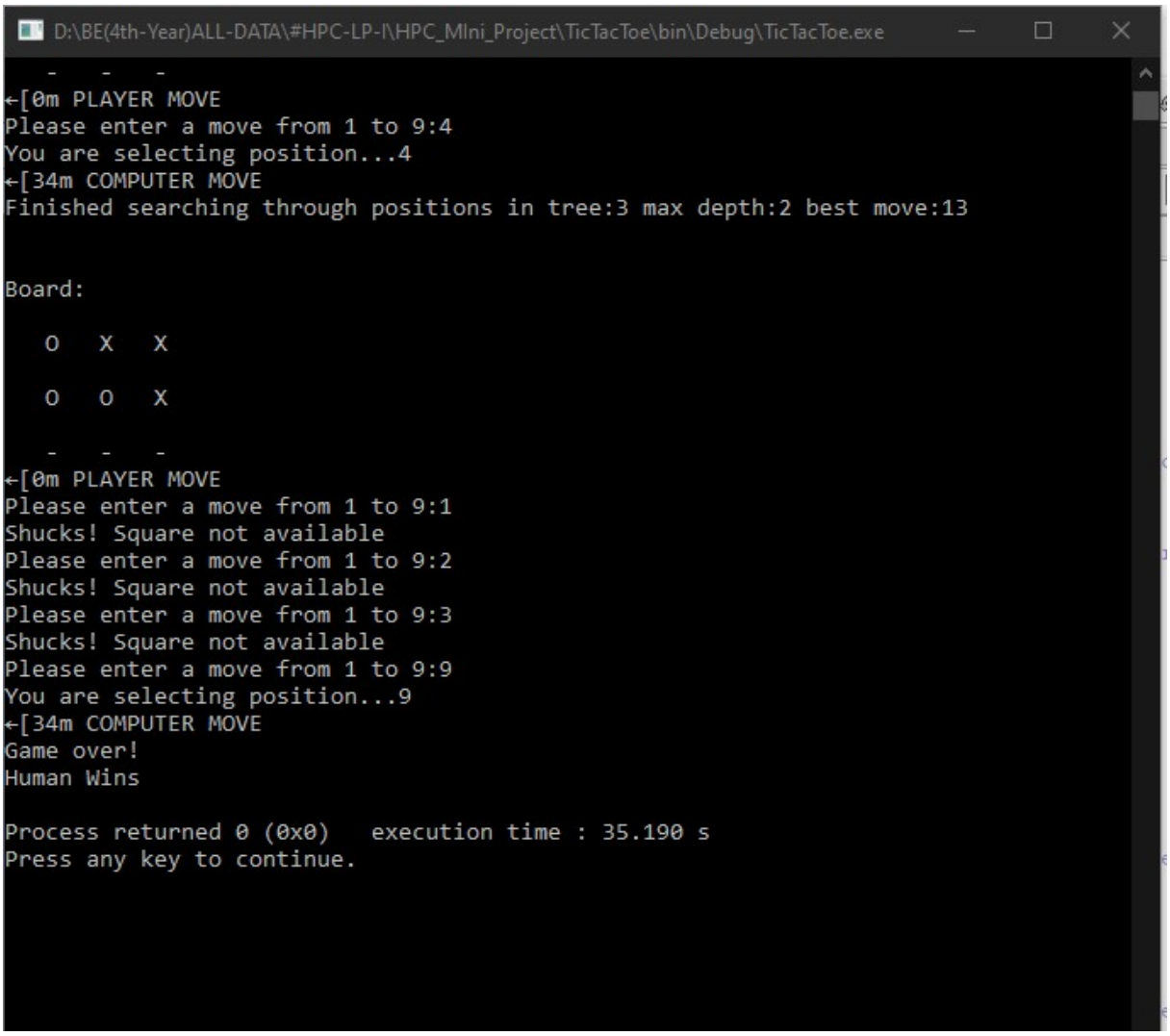
```
169         for (index = 0; index < 25; ++index)  {
170             board[index] = BORDER; /* all squares to border square */
171         }
172
173         for (index = 0; index < 9; ++index)  {
174             board[ConvertTo25[index]] = EMPTY /* all squares to empty */;
175         }
176            }
177
178     void PrintBoard(const int *board) {
179
180         int index = 0;
181         char pceChars[] = "OX|-";/* board chars */
182
183         printf("\n\nBoard:\n\n");
184         for(index = 0; index < 9; ++index) { /* for the 9 pos on board */
185             if(index!=0 && index%3==0) { /* if 3 pos on each line */
186                 printf("\n\n");
187             }
188             printf("%4c",pceChars[board[ConvertTo25[index]]]);
189         }
190         printf("\n");
191     }
192
193     int GetNextBest(const int *board)  {
194     /* if comp didn't find winning move, place priority for move in middle */
195     /* if middle not available, then */
196     /* place priority on corners, if corners not available */
197     /* then make random move */
198
199         int ourMove = ConvertTo25[InMiddle]; // set move to middle
200         if(board[ourMove] == EMPTY)  {
201             return ourMove; // if board empty place in middle
202         }
203
204         int index = 0; // indexing for looping
205         ourMove = -1; // next best not found
206
207         for(index = 0; index < 4; index++) { // loop for no of corners
208             ourMove = ConvertTo25[Corners[index]];
209             if(board[ourMove] == EMPTY)  {
210                 break;
211             }
212             ourMove = -1;
213         }
214
215         return ourMove;
216     }
217
218     int GetWinningMove(int *board, const int side) {
219
220         int ourMove = -1;
221         int winFound = 0;
222         int index = 0;
223
224         for(index = 0; index < 9; ++index)  {
225             if( board[ConvertTo25[index]] == EMPTY)  {
226                 ourMove = ConvertTo25[index];
227                 board[ourMove] = side;
228
229                 if(FindThreeInARow(board, ourMove, side) == 3)  {
230                     winFound = 1;
231                 }
232                 board[ourMove] = EMPTY;
233                 if(winFound == 1)  {
234                     break;
235                 }
236                 ourMove = -1;
237             };
238         }
239         return ourMove;
240     }
241
242
243     int GetComputerMove(int *board, const int side)  {
244         ply=0;
245         positions=0;
246         maxPly=0;
247         int best = MinMax(board, side);
248         printf("Finished searching through positions in tree:%d max depth:%d best
    move:%d\n",positions,maxPly,best);
249         return best;
250     }
```

```
252  int GetHumanMove(const int *board) {
253
254      char userInput[4];
255
256      int moveOk = 0;
257      int move = -1;
258
259      while (moveOk == 0) {
260
261          printf("Please enter a move from 1 to 9:");
262          fgets(userInput, 3, stdin);
263          fflush(stdin); /* fgets take first 3 chars and flush rest */
264
265          if(strlen(userInput) != 2) {
266              printf("Shucks! You entered an invalid strlen()! \n");
267              continue;
268          }
269
270          if( sscanf(userInput, "%d", &move) != 1) {
271              move = -1;
272              printf("Shucks! You entered an invalid sscanf()! \n");
273              continue;
274          }
275
276          if( move < 1 || move > 9) {
277              move = -1;
278              printf("Shucks! You entered an invalid range! \n");
279              continue;
280          }
281
282          move--; // Zero indexing
283
284          if( board[ConvertTo25[move]]!=EMPTY) {
285              move=-1;
286              printf("Shucks! Square not available\n");
287              continue;
288          }
289          moveOk = 1;
290      }
291      printf("You are selecting position...%d\n",(move+1));
292      return ConvertTo25[move];
293  }
294
295  int HasEmpty(const int *board) { /* Has board got empty sq */
296      int index = 0;
297      for (index = 0; index < 9; ++index) {
298          if( board[ConvertTo25[index]] == EMPTY) return 1;
299      }
300      return 0;
301  }
302
303  void MakeMove (int *board, const int sq, const side) {
304      board[sq] = side; /* pos of square equal the side (either x or o) */
305  }
306
307  void RunGame() {
308  printf("%s TIC TAC TOE \n", KRED);
309      int GameOver = 0;
310      int Side = NOUGHTS;
311      int LastMoveMade = 0;
312      int board[25];
313
314      InitialiseBoard(&board[0]);
315      PrintBoard(&board[0]);
316
317      while (!GameOver) { // while game is not over
318      if (Side==NOUGHTS) {
319          LastMoveMade = GetHumanMove (&board[0]);
320          MakeMove(&board[0], LastMoveMade,Side);
321          Side=CROSSES;
322  printf("%s COMPUTER MOVE \n", KBLU);
323  }
324      else {
325      LastMoveMade = GetComputerMove(&board[0], Side);
326      MakeMove(&board[0], LastMoveMade, Side);
327      Side=NOUGHTS;
328      PrintBoard(&board[0]);
329  printf("%s PLAYER MOVE \n", KNRM);
330
331      }
332
333  // if three in a row exists Game is over
334          if( FindThreeInARow(board, LastMoveMade, Side ^ 1) == 3) {
335              printf("Game over!\n");
```

```
336                 GameOver = 1;
337                 if(Side==NOUGHTS) {
338                     printf("Computer Wins\n");
339                 } else {
340                     printf("Human Wins\n");
341                 }
342             }
343
344         if(!HasEmpty(board)) {
345         printf("Game Over! I know, it's a shame it can't last forever! \n");
346         GameOver = 1;
347         printf("It's a draw! Come on, try harder for the win next time!");
348     }
349         }
350         }
351
352     int main() {
353         srand(time(NULL)); /* seed random no generator - moves on board randomly */
354         RunGame();
355         return 0;
356         }
357
```

5. Testing

```
D:\BE(4th-Year)ALL-DATA\#HPC-LP-I\HPC_MIni_Project\TicTacToe\bin\Debug\TicTacToe.exe
←[31m TIC TAC TOE


Board:

   -   -   -

   -   -   -

   -   -   -
Please enter a move from 1 to 9:1
You are selecting position...1
←[34m COMPUTER MOVE
Finished searching through positions in tree:7 max depth:6 best move:7


Board:

   O   X   -

   -   -   -

   -   -   -
←[0m PLAYER MOVE
Please enter a move from 1 to 9:5
You are selecting position...5
←[34m COMPUTER MOVE
Finished searching through positions in tree:5 max depth:4 best move:8


Board:

   O   X   X

   -   O   -

   -   -   -
←[0m PLAYER MOVE
Please enter a move from 1 to 9:4
You are selecting position...4
←[34m COMPUTER MOVE
Finished searching through positions in tree:3 max depth:2 best move:13
```


D:\BE(4th-Year)ALL-DATA\#HPC-LP-I\HPC_MIni_Project\TicTacToe\bin\Debug\TicTacToe.exe
- - -
←[0m PLAYER MOVE
Please enter a move from 1 to 9:4
You are selecting position...4
←[34m COMPUTER MOVE
Finished searching through positions in tree:3 max depth:2 best move:13
Board:
O X X
O O X
- - -
←[0m PLAYER MOVE
Please enter a move from 1 to 9:1
Shucks! Square not available
Please enter a move from 1 to 9:2
Shucks! Square not available
Please enter a move from 1 to 9:3
Shucks! Square not available
Please enter a move from 1 to 9:9
You are selecting position...9
←[34m COMPUTER MOVE
Game over!
Human Wins
Process returned 0 (0x0) execution time : 35.190 s
Press any key to continue.

6. Installation Instructions

6.1. Prerequisites

The interesting This project requires you to have basic knowledge of C programming, OpenMp library and The parallel programming.

6.2. Install CodeBlocks or Any IDE.

6.3. Install OpenMp library.

7. Summary

In this mini project, we have successfully built a OpenMP with C project on Tic Tac Toe game. We have built and we have build by using parallel programming to implement the program.

8. References

1] K. Kask. [Online]. Available: https://www.ics.uci.edu/~kkask/Fall2016%20CS271/slides/04-games.pdf. [Accessed 02 01 2020].

[2] G. Surma. [Online]. Available: https://towardsdatascience.com/tic-tac-toe-creatingunbeatable-ai-with-minimax-algorithm-8af9e52c1e7d. [Accessed 20 12 2019].

[3] P. G. ,. P. S. P. Sunil Karamchandani, "A Simple Algorithm For Designing An Artificial Intelligence Based Tic Tac Toe Game".

[4] 12 09 2019. [Online]. Available: https://www.edureka.co/blog/alpha-beta-pruning-in-ai.