PDC PROJECT FINAL CODE AND OUTPUT FILE

TOPIC: parallelism of minimax algorithm
Team members:

Harpal Patel	19BCE0692
Pankaj Shivnani	19BCE0821
Aditya Lal	19BCE2025

Code for serial algorithm:

```
#define __BOARD_DEF

#include <stdlib.h>

#include <omp.h>

#include <stdio.h>

#include <string.h>

#include <time.h>

#define X 1

#define EMPTY 10

#define NO_WINNER 20

#define N 3

#define M N

#define CHUNK_SIZE 1

typedef unsigned char symbol_t;

typedef struct board {

 symbol_t m[N][M];

 unsigned short n_empty;

} board_t;

typedef struct move {

 unsigned short i, j;

} move_t;

board_t* create_board();

void put_symbol(board_t*, symbol_t, move_t*);

void clear_symbol(board_t*, move_t*);

symbol_t winner(board_t*);
```

```
void print_board(board_t*);
move_t** get_all_possible_moves(board_t*, symbol_t, int*);
symbol_t other_symbol(symbol_t);
board_t* clone_board(board_t*);
board_t* create_board() {
 int i, j;
 board_t* board = (board_t*) malloc(sizeof(board_t));
 for(i = 0; i < N; i++) {
 for(j = 0; j < M; j++) {
 board->m[i][j] = EMPTY;
 }
 }
 board->n_empty = N * M;
 return board;
}
void put_symbol(board_t* board, symbol_t symbol, move_t* move) {
 board->m[move->i][move->j] = symbol;
 board->n_empty --;
}
void clear_symbol(board_t* board, move_t* move) {
 board->m[move->i][move->j] = EMPTY;
 board->n_empty ++;
}
symbol_t winner(board_t* b) {
 int i, j;
 symbol_t sym;
int equal;
 // check on rows
 for(i =0; i < N; i++) {
 equal = 1;
 sym = b->m[i][0];
```

```
if(sym != EMPTY) {
for(j = 1; j < M; j++) {
if(b->m[i][j] != sym) {
equal = 0;
break;
}
}
if(equal == 1) {
return sym;
}
}
}
// check on columns
for(i = 0; i < M; i++) {
equal = 1;
sym = b->m[0][i];
if(sym != EMPTY) {
for(j = 1; j < N; j++) {
if(b->m[j][i] != sym) {
equal= 0;
break;
}
}
if(equal == 1) {
return sym;
}
}
}
// main diagonal
equal = 1;
sym = b->m[0][0];
```

```
 if(sym != EMPTY) {
 for(i = 1; i < N; i++) {
 if(b->m[i][i] != sym) {
 equal = 0;
 break;
}
 }
 if(equal == 1) {
 return sym;
 }
 }
 // secondary diagonal
 equal = 1;
 sym = b->m[0][M-1];
 if(sym!= EMPTY) {
 for(i =1; i < N; i++) {
 if(b->m[i][M-i-1] != sym) {
 equal = 0;
 break;
 }
 }
 if(equal == 1) {
 return sym;
 }
 }
 if(b->n_empty == 0) {
 return NO_WINNER;
 }
 return EMPTY;
}
void print_board(board_t* board) {
```

```
 int i, j;
 for(i = 0; i < N; i++) {
 for(j = 0; j < M; j++) {
 if(board->m[i][j] == X) {
 printf("X ");
 } else if(board->m[i][j] == 0) {
 printf("0 ");
 }else {
 printf("- ");
 }
 }
 printf("\n");
 }
}
move_t** get_all_possible_moves(board_t* board, symbol_t symbol, int* n) {
 int i,j;
 move_t** list = (move_t**) malloc(board->n_empty * sizeof(move_t*));
 *n = 0;
 for(i = 0; i < N; i++) {
 for(j = 0; j < M; j++) {
 if(board->m[i][j] == EMPTY) {
 list[(*n)] = (move_t*) malloc(sizeof(move_t));
 list[(*n)]->i = i;
 list[(*n)]->j = j;
 (*n) ++;
 }
 }
 }
 return list;
}
symbol_t other_symbol(symbol_t symbol) {
```

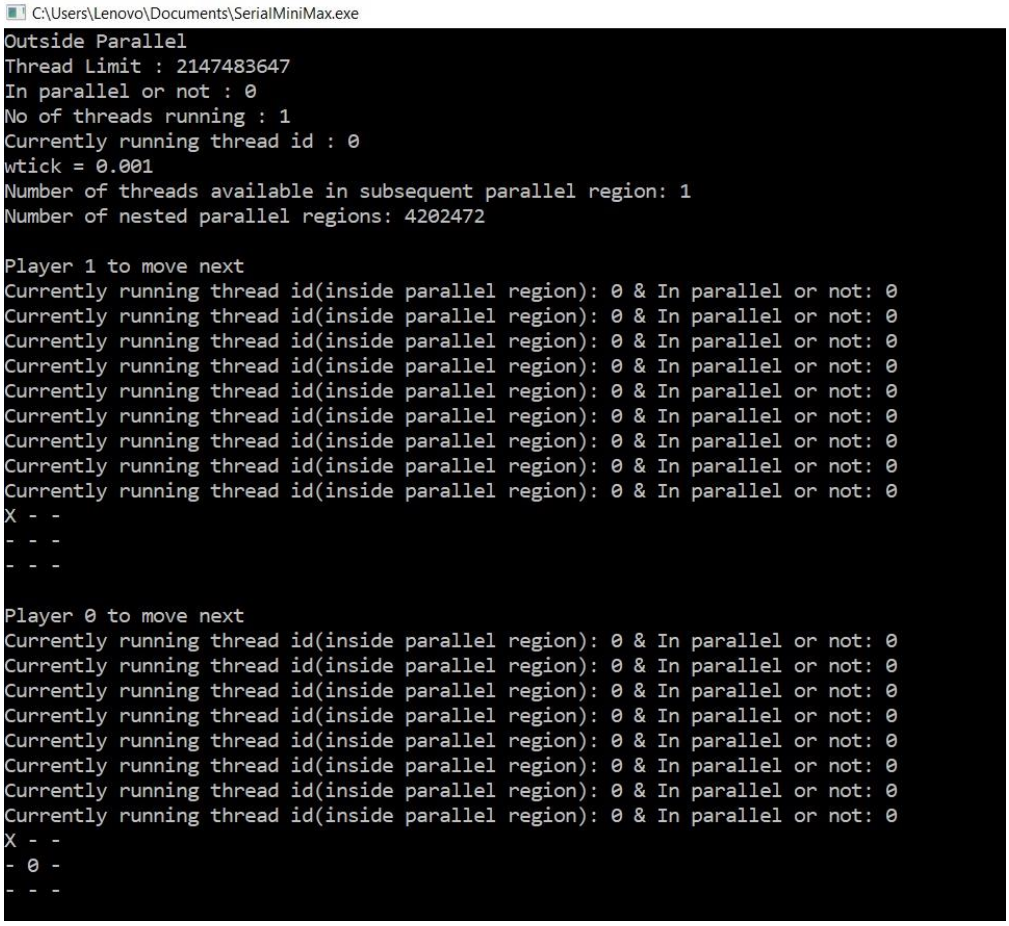
```
 return 1 - symbol;
}
board_t* clone_board(board_t* b) {
 int i, j;
 board_t* board = (board_t*) malloc(sizeof(board_t));
 board->n_empty = b->n_empty;
 for(i= 0; i < N; i++) {
 for(j = 0; j < M; j++) {
 board->m[i][j] = b->m[i][j];
 }
 }
 return board;
}
int get_score(board_t* board, int depth, symbol_t symbol) {
 symbol_t result = winner(board);
 if(result == symbol) {
 return N * M + 10 - depth;
 } else if(result != EMPTY && result != NO_WINNER) {
 return -(N * M) - 10 + depth;
 } else if(result == NO_WINNER) {
 return 1;
 } return 0;
}
 //SERIAL CODE PART:
int move(board_t* board, symbol_t symbol, int depth, int alpha, int beta) {
 int n, i;
 move_t* max_move;
 int score = get_score(board, depth, symbol);
 if(score != 0) {
 return score;
 }
```

```
 move_t** moves = get_all_possible_moves(board, symbol, &n);
 if(depth == 0) {
 int max_score = -9999;
 int flag;
 int nthreads;
 int thread_num;
 board_t* b;
 for(i = 0; i < n; i++) {
 int thread_num= omp_get_thread_num();
 int inp= omp_in_parallel();
 printf("Currently running thread id(inside parallel region): %d & In
parallel or not: %d\n",thread_num,inp);
 b = clone_board(board);
 put_symbol(b, symbol, moves[i]);
 score = -move(b, other_symbol(symbol), depth + 1, -beta, -
max_score);{
 if(score > max_score) {
 max_score = score;
 max_move = moves[i];
 }
 }
 free(b);
 }
 alpha = max_score;
 } else {
 for(i = 0; i < n; i++) {
 put_symbol(board, symbol, moves[i]);
 score = -move(board, other_symbol(symbol), depth + 1, -beta, -alpha);
 clear_symbol(board, moves[i]);
 if(score > alpha) {
 alpha = score; max_move = moves[i];
```

```
 }
 if(alpha >= beta) {
 if(depth == 0) {
printf("%i %i %i\n", i, beta, alpha);
 }
 break;
 }
 }
 }
 if(depth == 0) {
 put_symbol(board, symbol, max_move);
 }
 for(i = 0; i < n; i++) {
 free(moves[i]);
 }
 free(moves);
 return alpha;
 }
int main(int argc, char* argv[]) {
 clock_t start_clock = clock();
 int limit = omp_get_thread_limit();
 printf("Outside Parallel\n");
 printf("Thread Limit : %d\n",limit);
 int in = omp_in_parallel();
 printf("In parallel or not : %d\n",in);
 int thread = omp_get_num_threads();
 printf("No of threads running : %d\n",thread);
 int current_thread = omp_get_thread_num();
 printf("Currently running thread id : %d\n",current_thread);
 double wtick = omp_get_wtick();
 printf("wtick = %.4g\n",wtick);
```

```
 omp_set_dynamic(9);
 printf("Number of threads available in subsequent parallel region: %d\n",
omp_get_dynamic());
printf("Number of nested parallel regions: %d\n",omp_get_level);
board_t* board = create_board();
symbol_t result;
symbol_t current_symbol = X;
move_t m;
m.i = 1;
m.j = 1;
while(1) {
 printf("\nPlayer %i to move next\n", (int) current_symbol);
move(board, current_symbol, 0, -9999, 9999);
 print_board(board);
 result = winner(board);
 if(result != EMPTY) {
 break;
 }
 current_symbol = 1 - current_symbol;
 }
 printf("\nWinner: %i", (int) result);
 clock_t end_clock = clock();
 printf("\nProgram Execution Time : %ld ms",(end_clock-start_clock));
 return 0;
}
```

Output of serial algorithm: EXECUTION TIME FOR THIS TEST IS 93ms



```
Player 1 to move next
Currently running thread id(inside parallel region): 0 & In parallel or not: 0
Currently running thread id(inside parallel region): 0 & In parallel or not: 0
Currently running thread id(inside parallel region): 0 & In parallel or not: 0
Currently running thread id(inside parallel region): 0 & In parallel or not: 0
Currently running thread id(inside parallel region): 0 & In parallel or not: 0
Currently running thread id(inside parallel region): 0 & In parallel or not: 0
Currently running thread id(inside parallel region): 0 & In parallel or not: 0
X X -
- 0 -
- - -

Player 0 to move next
Currently running thread id(inside parallel region): 0 & In parallel or not: 0
Currently running thread id(inside parallel region): 0 & In parallel or not: 0
Currently running thread id(inside parallel region): 0 & In parallel or not: 0
Currently running thread id(inside parallel region): 0 & In parallel or not: 0
Currently running thread id(inside parallel region): 0 & In parallel or not: 0
Currently running thread id(inside parallel region): 0 & In parallel or not: 0
X X 0
- 0 -
- - -

Player 1 to move next
Currently running thread id(inside parallel region): 0 & In parallel or not: 0
Currently running thread id(inside parallel region): 0 & In parallel or not: 0
Currently running thread id(inside parallel region): 0 & In parallel or not: 0
Currently running thread id(inside parallel region): 0 & In parallel or not: 0
Currently running thread id(inside parallel region): 0 & In parallel or not: 0
X X 0
- 0 -
X - -
```

```
Player 0 to move next
Currently running thread id(inside parallel region): 0 & In parallel or not: 0
Currently running thread id(inside parallel region): 0 & In parallel or not: 0
Currently running thread id(inside parallel region): 0 & In parallel or not: 0
Currently running thread id(inside parallel region): 0 & In parallel or not: 0
X X 0
0 0 -
X - -

Player 1 to move next
Currently running thread id(inside parallel region): 0 & In parallel or not: 0
Currently running thread id(inside parallel region): 0 & In parallel or not: 0
Currently running thread id(inside parallel region): 0 & In parallel or not: 0
X X 0
0 0 X
X - -

Player 0 to move next
Currently running thread id(inside parallel region): 0 & In parallel or not: 0
Currently running thread id(inside parallel region): 0 & In parallel or not: 0
X X 0
0 0 X
X 0 -

Player 1 to move next
Currently running thread id(inside parallel region): 0 & In parallel or not: 0
X X 0
0 0 X
X 0 X

Winner: 20
Program Execution Time : 93 ms
```

Code for parallel algorithm:

```
#define __BOARD_DEF
#include <stdlib.h>
#include <omp.h>
#include <stdio.h>
#include <string.h>
#include <time.h>
#define X 1
#define EMPTY 10
#define NO_WINNER 20
#define N 3
#define M N
#define CHUNK_SIZE 1
typedef unsigned char symbol_t;
typedef struct board {
 symbol_t m[N][M];
 unsigned short n_empty;
} board_t;
typedef struct move {
 unsigned short i, j;
} move_t;
board_t* create_board();
void put_symbol(board_t*, symbol_t, move_t*);
void clear_symbol(board_t*, move_t*);
symbol_t winner(board_t*);
void print_board(board_t*);
```

```
move_t** get_all_possible_moves(board_t*, symbol_t, int*);
symbol_t other_symbol(symbol_t);
board_t* clone_board(board_t*);
board_t* create_board() {
 int i, j;
 board_t* board = (board_t*) malloc(sizeof(board_t));
 for(i = 0; i < N; i++) {
 for(j = 0; j < M; j++) {
 board->m[i][j] = EMPTY;
 }
 }
 board->n_empty = N * M;
 return board;
}
void put_symbol(board_t* board, symbol_t symbol, move_t* move) {
 board->m[move->i][move->j] = symbol;
 board->n_empty --;
}
void clear_symbol(board_t* board, move_t* move) {
 board->m[move->i][move->j] = EMPTY;
 board->n_empty ++;
}
symbol_t winner(board_t* b) {
 int i, j;
 symbol_t sym;
 int equal;
 // check on rows
 for(i =0; i < N; i++) {
 equal = 1;
```

```
sym = b->m[i][0];
if(sym != EMPTY) {
for(j = 1; j < M; j++) {
if(b->m[i][j] != sym) {
equal = 0;
break;
}
}
if(equal == 1) {
return sym;
}
}
}
// check on columns
for(i = 0; i < M; i++) {
equal = 1;
sym = b->m[0][i];
if(sym != EMPTY) {
for(j = 1; j < N; j++) {
if(b->m[j][i] != sym) {
equal= 0;
break;
}
}
if(equal == 1) {
return sym;
}
}
}
```

```
// main diagonal
equal = 1;
sym = b->m[0][0];
if(sym != EMPTY) {
for(i = 1; i < N; i++) {
if(b->m[i][i] != sym) {
equal = 0;
break;
}
}
if(equal == 1) {
return sym;
}
}
// secondary diagonal
equal = 1;
sym = b->m[0][M-1];
if(sym!= EMPTY) {
for(i =1; i < N; i++) {
if(b->m[i][M-i-1] != sym) {
equal = 0;
break;
}
}
if(equal == 1) {
return sym;
}
}
if(b->n_empty == 0) {
```

```
 return NO_WINNER;
 }
 return EMPTY;
}
void print_board(board_t* board) {
 int i, j;
 for(i = 0; i < N; i++) {
 for(j = 0; j < M; j++) {
 if(board->m[i][j] == X) {
 printf("X ");
 } else if(board->m[i][j] == 0) {
 printf("0 ");
 }else {
 printf("- ");
 }
 }
 printf("\n");
 }
}
move_t** get_all_possible_moves(board_t* board, symbol_t symbol, int* n) {
 int i,j;
 move_t** list = (move_t**) malloc(board->n_empty * sizeof(move_t*));
 *n = 0;
 for(i = 0; i < N; i++) {
 for(j = 0; j < M; j++) {
 if(board->m[i][j] == EMPTY) {
 list[(*n)] = (move_t*) malloc(sizeof(move_t));
 list[(*n)]->i = i;
 list[(*n)]->j = j;
```

```
 (*n) ++;
 }
 }
 }
 return list;
}
symbol_t other_symbol(symbol_t symbol) {
return 1 - symbol;
}
board_t* clone_board(board_t* b) {
 int i, j;
 board_t* board = (board_t*) malloc(sizeof(board_t));
 board->n_empty = b->n_empty;
 for(i= 0; i < N; i++) {
 for(j = 0; j < M; j++) {
 board->m[i][j] = b->m[i][j];
 }
 }
 return board;
}
int get_score(board_t* board, int depth, symbol_t symbol) {
 symbol_t result = winner(board);
 if(result == symbol) {
 return N * M + 10 - depth;
 } else if(result != EMPTY && result != NO_WINNER) {
 return -(N * M) - 10 + depth;
 } else if(result == NO_WINNER) {
 return 1;
 } return 0;
```

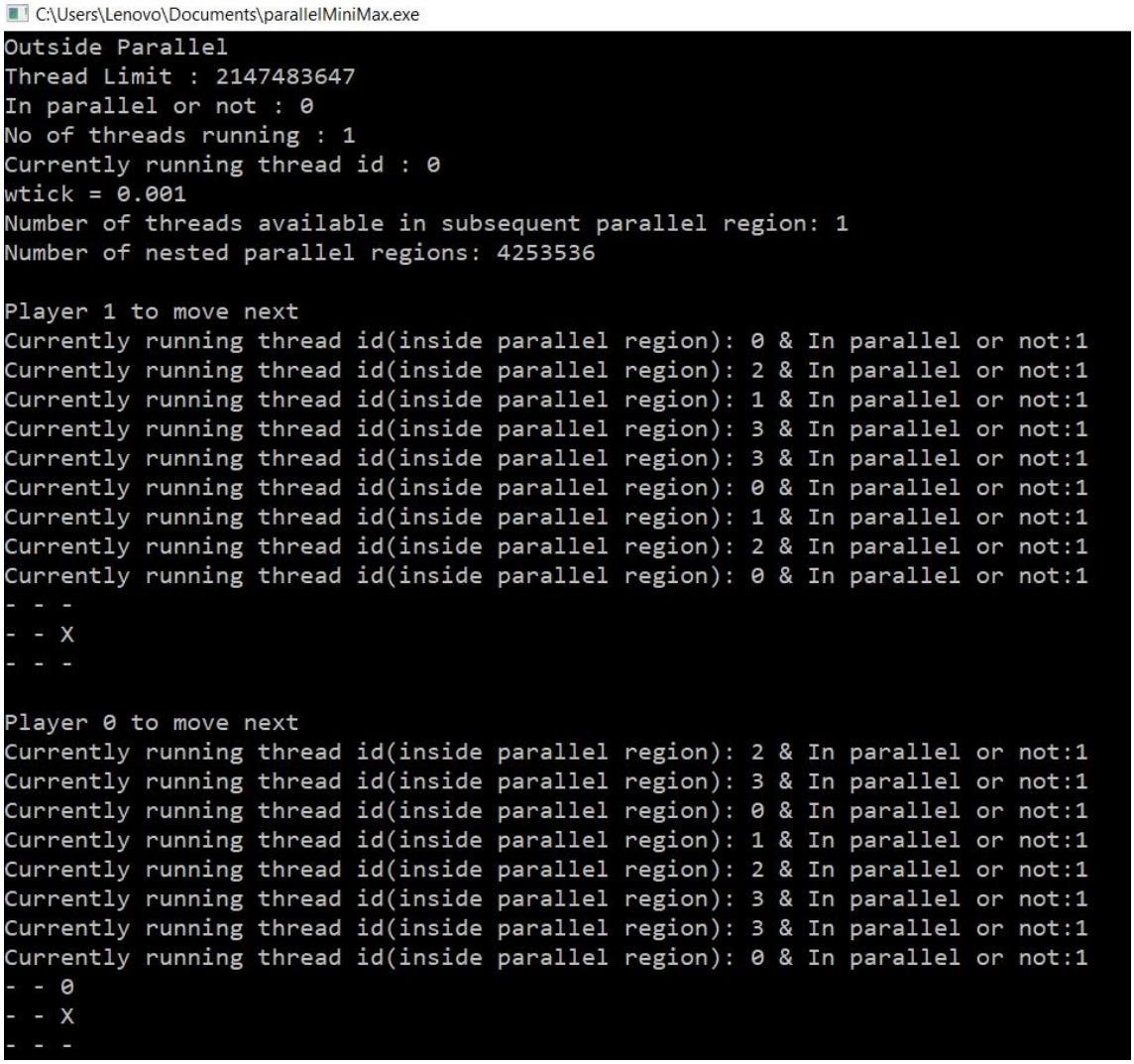
```
}
int move(board_t* board, symbol_t symbol, int depth, int alpha, int beta) {
 int n, i;
 move_t* max_move;
 int score = get_score(board, depth, symbol);
 if(score != 0) {
 return score;
 }
 move_t** moves = get_all_possible_moves(board, symbol, &n);
 if(depth == 0) {
 int max_score = -9999;
 int flag;
 int nthreads;
 int thread_num;
 board_t* b;
 //PARALLEL CODE PART
 #pragma atomic write
 #pragma omp flush flag = 1;
#pragma omp flush (flag)
 #pragma omp master
 {
 thread_num= omp_get_thread_num(); nthreads =
omp_get_num_threads();
 }
 #pragma omp barrier
 #pragma omp parallel for private(i, score, b, thread_num) shared(board,
alpha, beta, moves, depth, symbol, max_score) schedule(guided, CHUNK_SIZE)
 for(i = 0; i < n; i++) {
 int thread_num= omp_get_thread_num();
```

```
int inp= omp_in_parallel();
printf("Currently running thread id(inside parallel region): %d & In parallel
or not:%d\n",thread_num,inp);
b = clone_board(board);
put_symbol(b, symbol, moves[i]);
score = -move(b, other_symbol(symbol), depth + 1, -beta, -max_score);
#pragma omp critical
{
if(score > max_score) {
max_score = score;
max_move= moves[i];
}
}
free(b);
}
alpha = max_score;
} else {
for(i = 0; i < n; i++) {
put_symbol(board, symbol, moves[i]);
score = -move(board, other_symbol(symbol), depth + 1, -beta, -alpha);
clear_symbol(board, moves[i]);
if(score > alpha) {
alpha = score;
max_move = moves[i];
}
if(alpha >= beta) {
if(depth == 0) {
printf("%i %i %i\n", i, beta, alpha);
}
```

```
 break;
}
 }
 }
 if(depth == 0) {
 put_symbol(board, symbol, max_move);
 }
 for(i = 0; i < n; i++) {
 free(moves[i]);
 }
 free(moves);
 return alpha;
}
int main(int argc, char* argv[]) {
 clock_t start_clock = clock();
 int limit = omp_get_thread_limit();
 printf("Outside Parallel\n");
 printf("Thread Limit : %d\n",limit);
 int in = omp_in_parallel();
 printf("In parallel or not : %d\n",in);
 int thread = omp_get_num_threads();
 printf("No of threads running : %d\n",thread);
 int current_thread = omp_get_thread_num();
 printf("Currently running thread id : %d\n",current_thread);
 double wtick = omp_get_wtick();
 printf("wtick = %.4g\n",wtick);
 omp_set_dynamic(9);
 printf("Number of threads available in subsequent parallel region: %d\n",
omp_get_dynamic());
```

```
printf("Number of nested parallel regions: %d\n",omp_get_level);
board_t* board = create_board();
symbol_t result;
symbol_t current_symbol = X;
move_t m;
m.i = 1;
m.j = 1;
while(1) {
 printf("\nPlayer %i to move next\n", (int) current_symbol);
 move(board, current_symbol, 0, -9999, 9999);
 print_board(board);
 result = winner(board);
 if(result != EMPTY) {
break;
 }
 current_symbol = 1 - current_symbol;
 }
 printf("\nWinner: %i", (int) result);
 clock_t end_clock = clock();
 printf("\nProgram Execution Time : %ld ms",(end_clock-start_clock));
 return 0;
}
```

Output for parallel algorithm: EXECUTION TIME FOR THIS TEST IS 83ms


C:\Users\Lenovo\Documents\parallelMiniMax.exe
Outside Parallel
Thread Limit : 2147483647
In parallel or not : 0
No of threads running : 1
Currently running thread id : 0
wtick = 0.001
Number of threads available in subsequent parallel region: 1
Number of nested parallel regions: 4253536
Player 1 to move next
Currently running thread id(inside parallel region): 0 & In parallel or not:1
Currently running thread id(inside parallel region): 2 & In parallel or not:1
Currently running thread id(inside parallel region): 1 & In parallel or not:1
Currently running thread id(inside parallel region): 3 & In parallel or not:1
Currently running thread id(inside parallel region): 3 & In parallel or not:1
Currently running thread id(inside parallel region): 0 & In parallel or not:1
Currently running thread id(inside parallel region): 1 & In parallel or not:1
Currently running thread id(inside parallel region): 2 & In parallel or not:1
Currently running thread id(inside parallel region): 0 & In parallel or not:1
- - -
- - X
- - -
Player 0 to move next
Currently running thread id(inside parallel region): 2 & In parallel or not:1
Currently running thread id(inside parallel region): 3 & In parallel or not:1
Currently running thread id(inside parallel region): 0 & In parallel or not:1
Currently running thread id(inside parallel region): 1 & In parallel or not:1
Currently running thread id(inside parallel region): 2 & In parallel or not:1
Currently running thread id(inside parallel region): 3 & In parallel or not:1
Currently running thread id(inside parallel region): 3 & In parallel or not:1
Currently running thread id(inside parallel region): 0 & In parallel or not:1
- - 0
- - X
- - -

```
Player 1 to move next
Currently running thread id(inside parallel region): 1 & In parallel or not:1
Currently running thread id(inside parallel region): 3 & In parallel or not:1
Currently running thread id(inside parallel region): 2 & In parallel or not:1
Currently running thread id(inside parallel region): 0 & In parallel or not:1
Currently running thread id(inside parallel region): 1 & In parallel or not:1
Currently running thread id(inside parallel region): 3 & In parallel or not:1
Currently running thread id(inside parallel region): 2 & In parallel or not:1
X - 0
- - X
- - -

Player 0 to move next
Currently running thread id(inside parallel region): 3 & In parallel or not:1
Currently running thread id(inside parallel region): 2 & In parallel or not:1
Currently running thread id(inside parallel region): 1 & In parallel or not:1
Currently running thread id(inside parallel region): 0 & In parallel or not:1
Currently running thread id(inside parallel region): 3 & In parallel or not:1
Currently running thread id(inside parallel region): 2 & In parallel or not:1
X - 0
- 0 X
- - -

Player 1 to move next
Currently running thread id(inside parallel region): 1 & In parallel or not:1
Currently running thread id(inside parallel region): 2 & In parallel or not:1
Currently running thread id(inside parallel region): 3 & In parallel or not:1
Currently running thread id(inside parallel region): 0 & In parallel or not:1
Currently running thread id(inside parallel region): 1 & In parallel or not:1
X - 0
- 0 X
X - -
```

```
Player 0 to move next
Currently running thread id(inside parallel region): 2 & In parallel or not:1
Currently running thread id(inside parallel region): 3 & In parallel or not:1
Currently running thread id(inside parallel region): 1 & In parallel or not:1
Currently running thread id(inside parallel region): 0 & In parallel or not:1
X - 0
0 0 X
X - -

Player 1 to move next
Currently running thread id(inside parallel region): 2 & In parallel or not:1
Currently running thread id(inside parallel region): 3 & In parallel or not:1
Currently running thread id(inside parallel region): 1 & In parallel or not:1
X X 0
0 0 X
X - -

Player 0 to move next
Currently running thread id(inside parallel region): 3 & In parallel or not:1
Currently running thread id(inside parallel region): 2 & In parallel or not:1
X X 0
0 0 X
X 0 -

Player 1 to move next
Currently running thread id(inside parallel region): 1 & In parallel or not:1
X X 0
0 0 X
X 0 X

Winner: 20
Program Execution Time : 83 ms
--------------------------------
```