ADVERSARIAL SEARCH

(Documentation)

**TicTacToe**

Is a game for two players, *X* and *O*, who take turns marking the spaces in a 3×3 grid. The player who succeeds in placing three of their marks in a horizontal, vertical, or diagonal row wins the game.

The following example game is won by the first player, X:

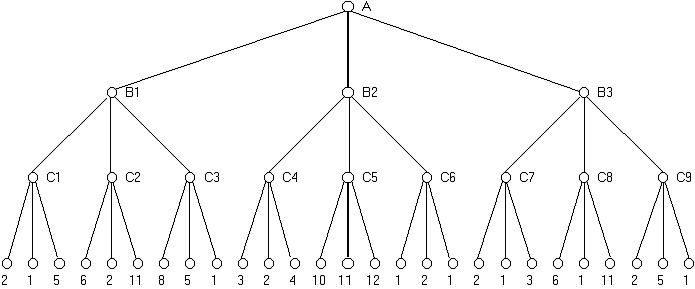


**Minimax**

the minimax algorithm is such a tactic, which uses the fact that the two players are working towards opposite goals to make predictions about which future states will be reached as the game progresses, and then proceeds accordingly to optimize its chance of victory. The theory behind minimax is that the algorithm's opponent will be trying to minimize whatever value the algorithm is trying to maximize (hence, "minimax"). Thus, the computer should make the move which leaves its opponent capable of doing the least damage.

In the ideal case (i.e., the computer has infinite time and infinite storage capacity), the computer will investigate every game outcome possible from the game's current state (as well as the paths taken to reach those states) and assign those outcomes values. For a win-or-lose game like chess or tic-tac-toe, there are only three possible values-win, lose, or draw (often assigned numeric values of 1, -1, and 0 respectively)--. Then, starting from the bottom, the computer evaluates which possible outcome is best for the computer's opponent. It then assumes that, if that game stage is reached, its opponent will make the move which leads to the outcome best for the opponent (and worse for the computer). Thus, it can "know" what its opponent will do, and have a concrete idea of what the final game state will be if that second-to-last position is in fact reached. Then, the computer can treat that position of a terminal node of that value, even though it is not actually a terminal value. This process can then be repeated a level higher, and so on. Ultimately, each option that the computer currently has available can be a assigned a value, as if it were a terminal state, and the computer simply picks the highest value and takes that action.

A relatively simple example of this is shown below:



In the above example (Fig. 1), assuming normally alternating turns, if the computer has the choice at level A, its opponent will have the choice at B, and the computer will again have the choice at C. Since the computer is trying to maximize its score, it can know ahead of time what it would choose should any given C node be reached. C1 thus effectively has a score of 5, C2, 11, C3, 8, and so on. When the opponent has a choice to make at B, however, they will choose the path that leads to the C node with the lowest effective score. Thus, the score of each B node can be thought to be the minimum of the effective scores of the C-nodes it leads to. For example, B1's score would be 5 (the minimum of 5, 11, and 8, as calculated above). B2 and B3 can be calculated in a similar fashion. Finally, we are back to the current turn. The computer now knows what will come of choosing B1, B2, or B3; and, though the actual endgame is many turns away, it will choose the maximum of those three for the best possible result. Note that, if the opponent does not behave as predicted, the calculation can simply be re-run, taking the current state as the starting node, and a result as good (or better) than what was predicted will still be achieved.

Minimax algorithm for Tictactoe

Code to solve nXn board.

**import** java.util.ArrayList;

**import** java.util.List;

**import** java.util.Random;

**import** java.util.Scanner;

**public** **class** Tictactoenxn {

**static** **int** *n* =0;

**static** **int** *count\_child* = 0;

**public** **static** **void** main(String[] args) {

**class** Point {

**int** x, y;

**public** Point(**int** x, **int** y) {

**this**.x = x;

**this**.y = y;

}

@Override

**public** String toString() {

**return** "[" + x + ", " + y + "]";

}

}

**class** PScores {

**int** score;

Point point;

PScores(**int** score, Point point) {

**this**.score = score;

**this**.point = point;

}

}

**class** Board {

List<Point> availablePoints;

Scanner scan = **new** Scanner(System.***in***);

**int**[][] board;

**public** Board() {

System.***out***.println("Enter Board dimension");

*n* = scan.nextInt();

board = **new** **int**[*n*][*n*];

}

**public** **boolean** isGameOver() {

//Game is over is someone has won, or board is full (draw)

**return** (hasXWon() || hasOWon() || getAvailableStates().isEmpty());

}

**public** **boolean** hasXWon() {

**int** Dia\_compare = 0;

**for**(**int** i = 0;i<*n*;i++)

{

**int** row\_compare = 0;

**int** col\_compare = 0;

**for**(**int** j = 0;j<*n*;j++)

{

**if**(board[i][j]==1)

{

row\_compare++;

**if**(row\_compare == *n*)

{

**return** **true**;

}

}

**if**(board[j][i]==1)

{

col\_compare++;

**if**(col\_compare==*n*)

{

**return** **true**;

}

}

**if**(i==j)

{

**if**(board[i][j]==1)

{

Dia\_compare++;

**if**(Dia\_compare == *n*)

{

**return** **true**;

}

}

}

}

}

**return** **false**;

}

**public** **boolean** hasOWon() {

**int** Dia\_compare = 0;

**for**(**int** i = 0;i<*n*;i++)

{

**int** row\_compare = 0;

**int** col\_compare = 0;

**for**(**int** j = 0;j<*n*;j++)

{

**if**(board[i][j]==2)

{

row\_compare++;

**if**(row\_compare == *n*)

{

**return** **true**;

}

}

**if**(board[j][i]==2)

{

col\_compare++;

**if**(col\_compare==*n*)

{

**return** **true**;

}

}

**if**(i==j)

{

**if**(board[i][j]==2)

{

Dia\_compare++;

**if**(Dia\_compare == *n*)

{

**return** **true**;

}

}

}

}

}

**return** **false**;

}

**public** List<Point> getAvailableStates() {

availablePoints = **new** ArrayList<>();

**for** (**int** i = 0; i < *n*; ++i) {

**for** (**int** j = 0; j < *n*; ++j) {

**if** (board[i][j] == 0) {

availablePoints.add(**new** Point(i, j));

}

}

}

**return** availablePoints;

}

**public** **void** placeAMove(Point point, **int** player) {

board[point.x][point.y] = player; //player = 1 for X, 2 for O

}

**public** Point returnBestMove() {

**int** MAX = -100000;

**int** best = -1;

**for** (**int** i = 0; i < rootsChildrenScores.size(); ++i) {

**if** (MAX < rootsChildrenScores.get(i).score) {

MAX = rootsChildrenScores.get(i).score;

best = i;

}

}

**return** rootsChildrenScores.get(best).point;

}

**void** takeHumanInput() {

System.***out***.println("Your move: ");

**int** x = scan.nextInt();

**int** y = scan.nextInt();

Point point = **new** Point(x, y);

placeAMove(point, 2);

}

**public** **void** displayBoard() {

System.***out***.println();

**for** (**int** i = 0; i < *n*; ++i)

{

**for** (**int** j = 0; j < *n*; ++j)

{

**if**(board[i][j] == 1)

{

System.***out***.print(" X |");

}

**else** **if**(board[i][j] == 2)

{

System.***out***.print(" O |");

}

**else**

{

System.***out***.print(" |");

}

}

System.***out***.println();

}

}

**public** **int** returnMin(List<Integer> list) {

**int** min = Integer.***MAX\_VALUE***;

**int** index = -1;

**for** (**int** i = 0; i < list.size(); ++i) {

**if** (list.get(i) < min) {

min = list.get(i);

index = i;

}

}

**return** list.get(index);

}

**public** **int** returnMax(List<Integer> list) {

**int** max = Integer.***MIN\_VALUE***;

**int** index = -1;

**for** (**int** i = 0; i < list.size(); ++i) {

**if** (list.get(i) > max) {

max = list.get(i);

index = i;

}

}

**return** list.get(index);

}

List<PScores> rootsChildrenScores;

**public** **void** callMinimax(**int** depth, **int** turn){

rootsChildrenScores = **new** ArrayList<>();

minimax(depth, turn);

}

**public** **int** minimax(**int** depth, **int** turn) {

**if** (hasXWon()) **return** +1;

**if** (hasOWon()) **return** -1;

List<Point> pointsAvailable = getAvailableStates();

**if** (pointsAvailable.isEmpty()) **return** 0;

List<Integer> scores = **new** ArrayList<>();

**for** (**int** i = 0; i < pointsAvailable.size(); ++i) {

Point point = pointsAvailable.get(i);

**if** (turn == 1) { //X's turn select the highest from below minimax() call

placeAMove(point, 1);

*count\_child*++;

displayBoard();

**int** currentScore = minimax(depth + 1, 2);

scores.add(currentScore);

**if** (depth == 0)

rootsChildrenScores.add(**new** PScores(currentScore, point));

} **else** **if** (turn == 2) {//O's turn select the lowest from below minimax() call

placeAMove(point, 2);

*count\_child*++;

displayBoard();

scores.add(minimax(depth + 1, 1));

}

board[point.x][point.y] = 0; //Reset this point

}

**return** turn == 1 ? returnMax(scores) : returnMin(scores);

}

}

Board board = **new** Board();

Random rand = **new** Random();

board.displayBoard();

System.***out***.println("Choose the player who start's the game?\n 1. Computer\n 2. User ");

**int** choice = board.scan.nextInt();

**if**(choice == 1){

// Point p = new Point(rand.nextInt(n), rand.nextInt(n));

// board.placeAMove(p, 1);

// board.displayBoard();

System.***out***.println("Computer move");

board.callMinimax(0, 1);

// for (PScores pas : board.rootsChildrenScores) {

// System.out.println("Point: " + pas.point + " Score: " + pas.score);

// }

board.placeAMove(board.returnBestMove(), 1);

board.displayBoard();

System.***out***.println("count: "+*count\_child*);

}

**while** (!board.isGameOver()) {

System.***out***.println("-------------------------------------------------------------------------------------------------------");

System.***out***.println("Your move: ");

Scanner scan = **new** Scanner(System.***in***);

System.***out***.println("Enter row(Starting from 1 to "+*n*+")");

**int** x = scan.nextInt();

**while**(x<1||x>3)

{

System.***out***.println("Enter valid row number(Starting from 1 to "+*n*+")");

x = scan.nextInt();

}

x--;

System.***out***.println("Enter Column(Starting from 1 to "+*n*+")");

**int** y = scan.nextInt();

**while**(y<1||y>3)

{

System.***out***.println("Enter valid column number(Starting from 1 to "+*n*+")");

y = scan.nextInt();

}

y--;

Point userMove = **new** Point(x, y);

board.placeAMove(userMove, 2); //2 for O and O is the user

board.displayBoard();

**if** (board.isGameOver()) {

**break**;

}

System.***out***.println("-------------------------------------------------------------------------------------------------------");

System.***out***.println("Computer move");

board.callMinimax(0, 1);

**for** (PScores pas : board.rootsChildrenScores) {

System.***out***.println("Score at point" + pas.point +" is "+ pas.score);

}

board.placeAMove(board.returnBestMove(), 1);

board.displayBoard();

System.***out***.println("count: "+*count\_child*);

}

**if** (board.hasXWon()) {

System.***out***.println("Computer: I Won!");

} **else** **if** (board.hasOWon()) {

System.***out***.println("Congratz, Player Won.");

} **else** {

System.***out***.println("It's a draw!");

}

}

}

Output1: For Board dimension 1\*1

Enter Board dimension

1

|

Choose the player who start's the game?

1. Computer

2. User

2

-------------------------------------------------------------------------------------------------------

Your move:

Enter row(Starting from 1 to 1)

1

Enter Column(Starting from 1 to 1)

1

O |

Congratz, Player Won.

Output1: For Board dimension 2\*2

Enter Board dimension

2

| |

| |

Choose the player who start's the game?

1. Computer

2. User

2

-------------------------------------------------------------------------------------------------------

Your move:

Enter row(Starting from 1 to 2)

1

Enter Column(Starting from 1 to 2)

1

O | |

| |

-------------------------------------------------------------------------------------------------------

Computer move

O | X |

| |

O | X |

O | |

O | X |

| O |

O | |

X | |

O | O |

X | |

O | |

X | O |

O | |

| X |

O | O |

| X |

O | |

O | X |

Score at point[0, 1] is -1

Score at point[1, 0] is -1

Score at point[1, 1] is -1

O | X |

| |

count: 9

-------------------------------------------------------------------------------------------------------

Your move:

Enter row(Starting from 1 to 2)

2

Enter Column(Starting from 1 to 2)

1

O | X |

O | |

Congratz, Player Won.

Output3: For Board dimension 3\*3

Enter Board dimension

3

| | |

| | |

| | |

Choose the player who start's the game?

1. Computer

2. User

2

-------------------------------------------------------------------------------------------------------

Your move:

Enter row(1 or 2 or 3)

2

Enter Column(1 or 2 or 3)

2

| | |

| O | |

| | |

-------------------------------------------------------------------------------------------------------

Computer move

count:65104

Score at point[0, 0] is 0

Score at point[0, 1] is -1

Score at point[0, 2] is 0

Score at point[1, 0] is -1

Score at point[1, 2] is -1

Score at point[2, 0] is 0

Score at point[2, 1] is -1

Score at point[2, 2] is 0

X | | |

| O | |

| | |

-------------------------------------------------------------------------------------------------------

Your move:

Enter row(1 or 2 or 3)

1

Enter Column(1 or 2 or 3)

2

X | O | |

| O | |

| | |

-------------------------------------------------------------------------------------------------------

Computer move

count:1150

Score at point[0, 2] is -1

Score at point[1, 0] is -1

Score at point[1, 2] is -1

Score at point[2, 0] is -1

Score at point[2, 1] is 0

Score at point[2, 2] is -1

X | O | |

| O | |

| X | |

-------------------------------------------------------------------------------------------------------

Your move:

Enter row(1 or 2 or 3)

1

Enter Column(1 or 2 or 3)

3

X | O | O |

| O | |

| X | |

-------------------------------------------------------------------------------------------------------

Computer move

count:56

Score at point[1, 0] is 0

Score at point[1, 2] is 0

Score at point[2, 0] is 1

Score at point[2, 2] is 0

X | O | O |

| O | |

X | X | |

-------------------------------------------------------------------------------------------------------

Your move:

Enter row(1 or 2 or 3)

2

Enter Column(1 or 2 or 3)

3

X | O | O |

| O | O |

X | X | |

-------------------------------------------------------------------------------------------------------

Computer move

count:2

Score at point[1, 0] is 1

Score at point[2, 2] is 1

X | O | O |

X | O | O |

X | X | |

Computer: I Won :-)!

**Alpha-Beta Prunning**

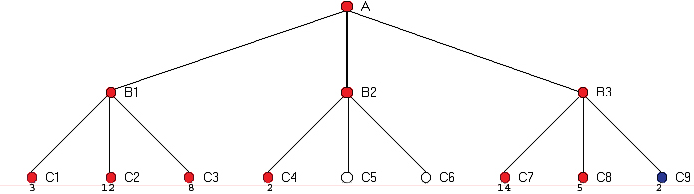
Alpha-beta pruning is an improvement over the minimax algorithm. The problem with minimax is that the number of game states it has to examine is exponential in the number of moves. While it is impossible to eliminate the exponent completely, we are able to cut it in half. It is possible to compute the correct minimax decision without looking at every node in the tree. Borrowing the idea of pruning, or eliminating possibilities from consideration without having to examine them, the algorithm allows us to discard large parts of the tree from consideration. When applied to a standard minimax tree, it returns the same move as minimax would, but prunes away branches that cannot possibly influence the final decision.

Alpha-beta pruning can be applied to trees of any depth and it often allows to prune away entire subtrees rather than justleaves. Here is the general algorithm:

1. Consider a node n somewhere in the tree, such that one can move to that node.  
2. If there is a better choice m either at the parent of the node n or at any choice point further up, n will never be reached.  
3. Once we have enough information about n to reach this conclusion, we can prune it.

Alpha-Beta pruning gets its name from the following parameters:

a = the value of the best choice we have found so far at any choice point along the path for MAX.  
ß = the value of the best choice we have found so far at any choice point along the path for MIN

As shown in above fig, Alpha-Beta search updates the values of a and ß as it goes along and prunes the remaining branches at a node as soon as the value of the current node is known to be worse than the current values of a and ß. The success of the algorithm depends on the order in which the successors are examined.

Java Code:

**import** java.util.ArrayList;

**import** java.util.List;

**import** java.util.Random;

**import** java.util.Scanner;

**public** **class** AlphaBeta {

**static** **int** *children\_count* = 0;

**public** **static** **void** main(String[] args) {

**class** XYCoordinate {

**int** x, y;

**public** XYCoordinate(**int** x, **int** y) {

**this**.x = x;

**this**.y = y;

}

@Override

**public** String toString() {

**return** "[" + x + ", " + y + "]";

}

}

**class** PScores {

**int** score;

XYCoordinate point;

PScores(**int** score, XYCoordinate point) {

**this**.score = score;

**this**.point = point;

}

}

**class** Board {

List<XYCoordinate> availablePoints;

Scanner scan = **new** Scanner(System.***in***);

**int**[][] board = **new** **int**[3][3];

List<PScores> rootsChildrenScore = **new** ArrayList<>();

**public** **int** evaluateBoard() {

**int** score = 0;

//Check all rows

**for** (**int** i = 0; i < 3; ++i) {

**int** blank = 0;

**int** X = 0;

**int** O = 0;

**for** (**int** j = 0; j < 3; ++j) {

**if** (board[i][j] == 0) {

blank++;

} **else** **if** (board[i][j] == 1) {

X++;

} **else** {

O++;

}

}

score+=changeInScore(X, O);

}

//Check all columns

**for** (**int** j = 0; j < 3; ++j) {

**int** blank = 0;

**int** X = 0;

**int** O = 0;

**for** (**int** i = 0; i < 3; ++i) {

**if** (board[i][j] == 0) {

blank++;

} **else** **if** (board[i][j] == 1) {

X++;

} **else** {

O++;

}

}

score+=changeInScore(X, O);

}

**int** blank = 0;

**int** X = 0;

**int** O = 0;

//Check diagonal (first)

**for** (**int** i = 0, j = 0; i < 3; ++i, ++j) {

**if** (board[i][j] == 1) {

X++;

} **else** **if** (board[i][j] == 2) {

O++;

} **else** {

blank++;

}

}

score+=changeInScore(X, O);

blank = 0;

X = 0;

O = 0;

//Check Diagonal (Second)

**for** (**int** i = 2, j = 0; i > -1; --i, ++j) {

**if** (board[i][j] == 1) {

X++;

} **else** **if** (board[i][j] == 2) {

O++;

} **else** {

blank++;

}

}

score+=changeInScore(X, O);

**return** score;

}

**private** **int** changeInScore(**int** X, **int** O){

**int** change;

**if** (X == 3) {

change = 100;

} **else** **if** (X == 2 && O == 0) {

change = 10;

} **else** **if** (X == 1 && O == 0) {

change = 1;

} **else** **if** (O == 3) {

change = -100;

} **else** **if** (O == 2 && X == 0) {

change = -10;

} **else** **if** (O == 1 && X == 0) {

change = -1;

} **else** {

change = 0;

}

**return** change;

}

//Set this to some value if you want to have some specified depth limit for search

**int** uptoDepth = -1;

**public** **int** alphaBetaMinimax(**int** alpha, **int** beta, **int** depth, **int** turn){

**if**(beta<=alpha){

System.***out***.println("Pruning at depth = "+depth);

**if**(turn == 1)

**return** Integer.***MAX\_VALUE***;

**else**

**return** Integer.***MIN\_VALUE***; }

**if**(depth == uptoDepth || isGameOver())

**return** evaluateBoard();

List<XYCoordinate> pointsAvailable = getAvailableStates();

**if**(pointsAvailable.isEmpty()) **return** 0;

**if**(depth==0) rootsChildrenScore.clear();

**int** maxValue = Integer.***MIN\_VALUE***, minValue = Integer.***MAX\_VALUE***;

**for**(**int** i=0;i<pointsAvailable.size(); ++i){

XYCoordinate point = pointsAvailable.get(i);

**int** currentScore = 0;

**if**(turn == 1){

placeAMove(point, 1);

*children\_count*++;

currentScore = alphaBetaMinimax(alpha, beta, depth+1, 2);

maxValue = Math.*max*(maxValue, currentScore);

//Set alpha

alpha = Math.*max*(currentScore, alpha);

**if**(depth == 0)

rootsChildrenScore.add(**new** PScores(currentScore, point));

}**else** **if**(turn == 2){

placeAMove(point, 2);

*children\_count*++;

currentScore = alphaBetaMinimax(alpha, beta, depth+1, 1);

minValue = Math.*min*(minValue, currentScore);

//Set beta

beta = Math.*min*(currentScore, beta);

}

//reset board

board[point.x][point.y] = 0;

//If a pruning has been done, don't evaluate the rest of the sibling states

**if**(currentScore == Integer.***MAX\_VALUE*** || currentScore == Integer.***MIN\_VALUE***) **break**;

}

**return** turn == 1 ? maxValue : minValue;

}

**public** **boolean** isGameOver() {

//Game is over if someone has won, or Available states will be empty when board is full (draw)

**return** (hasXWon() || hasOWon() || getAvailableStates().isEmpty());

}

**public** **boolean** hasXWon() {

**int** Dia\_compare = 0;

**for**(**int** i = 0;i<3;i++)

{

**int** row\_compare = 0;

**int** col\_compare = 0;

**for**(**int** j = 0;j<3;j++)

{

**if**(board[i][j]==1)

{

row\_compare++;

**if**(row\_compare == 3)

{

**return** **true**;

}

}

**if**(board[j][i]==1)

{

col\_compare++;

**if**(col\_compare==3)

{

**return** **true**;

}

}

**if**(i==j)

{

**if**(board[i][j]==1)

{

Dia\_compare++;

**if**(Dia\_compare == 3)

{

**return** **true**;

}

}

}

}

}

**return** **false**;

}

**public** **boolean** hasOWon() {

**int** Dia\_compare = 0;

**for**(**int** i = 0;i<3;i++)

{

**int** row\_compare = 0;

**int** col\_compare = 0;

**for**(**int** j = 0;j<3;j++)

{

**if**(board[i][j]==2)

{

row\_compare++;

**if**(row\_compare == 3)

{

**return** **true**;

}

}

**if**(board[j][i]==2)

{

col\_compare++;

**if**(col\_compare==3)

{

**return** **true**;

}

}

**if**(i==j)

{

**if**(board[i][j]==2)

{

Dia\_compare++;

**if**(Dia\_compare == 3)

{

**return** **true**;

}

}

}

}

}

**return** **false**;

}

**public** List<XYCoordinate> getAvailableStates() {

availablePoints = **new** ArrayList<>();

**for** (**int** i = 0; i < 3; ++i) {

**for** (**int** j = 0; j < 3; ++j) {

**if** (board[i][j] == 0) {

availablePoints.add(**new** XYCoordinate(i, j));

}

}

}

**return** availablePoints;

}

**public** **void** placeAMove(XYCoordinate point, **int** player) {

board[point.x][point.y] = player; //player = 1 for X, 2 for O

}

**public** XYCoordinate returnBestMove() {

**int** MAX = -100000;

**int** best = -1;

**for** (**int** i = 0; i < rootsChildrenScore.size(); ++i) {

**if** (MAX < rootsChildrenScore.get(i).score) {

MAX = rootsChildrenScore.get(i).score;

best = i;

}

}

**return** rootsChildrenScore.get(best).point;

}

**void** takeHumanInput() {

System.***out***.println("Your move: ");

**int** x = scan.nextInt();

**int** y = scan.nextInt();

XYCoordinate point = **new** XYCoordinate(x, y);

placeAMove(point, 2);

}

**public** **void** displayBoard() {

System.***out***.println();

**for** (**int** i = 0; i < 3; ++i)

{

**for** (**int** j = 0; j < 3; ++j)

{

**if**(board[i][j] == 1)

{

System.***out***.print(" X |");

}

**else** **if**(board[i][j] == 2)

{

System.***out***.print(" O |");

}

**else**

{

System.***out***.print(" |");

}

}

System.***out***.println();

}

}

**public** **void** resetBoard() {

**for** (**int** i = 0; i < 3; ++i) {

**for** (**int** j = 0; j < 3; ++j) {

board[i][j] = 0;

}

}

}

}

Board b = **new** Board();

System.***out***.print("\*\*\*\*\*Board\*\*\*\*\*");

b.displayBoard();

System.***out***.println("Choose the player who start's the game?\n 1. Computer\n 2. User ");

**int** choice = b.scan.nextInt();

**if** (choice == 1) {

b.alphaBetaMinimax(Integer.***MIN\_VALUE***, Integer.***MAX\_VALUE***, 0, 1);

**for** (PScores pas : b.rootsChildrenScore)

System.***out***.println("Point: " + pas.point + " Score: " + pas.score);

b.placeAMove(b.returnBestMove(), 1);

System.***out***.println("Children count: "+ *children\_count*);

*children\_count* = 0;

b.displayBoard();

}

**while** (!b.isGameOver()) {

System.***out***.println("-------------------------------------------------------------------------------------------------------");

System.***out***.println("Your move: ");

Scanner scan = **new** Scanner(System.***in***);

System.***out***.println("Enter row(1 or 2 or 3)");

**int** x = scan.nextInt();

**while**(x<1||x>3)

{

System.***out***.println("Enter valid row number(1 or 2 or 3)");

x = scan.nextInt();

}

x--;

System.***out***.println("Enter Column(1 or 2 or 3)");

**int** y = scan.nextInt();

**while**(y<1||y>3)

{

System.***out***.println("Enter valid column number(1 or 2 or 3)");

y = scan.nextInt();

}

y--;

XYCoordinate userMove = **new** XYCoordinate(x, y);

b.placeAMove(userMove, 2); //2 for O and O is the user

b.displayBoard();

**if** (b.isGameOver()) **break**;

b.alphaBetaMinimax(Integer.***MIN\_VALUE***, Integer.***MAX\_VALUE***, 0, 1);

**for** (PScores pas : b.rootsChildrenScore)

System.***out***.println("Point: " + pas.point + " Score: " + pas.score);

b.placeAMove(b.returnBestMove(), 1);

System.***out***.println("Children count: "+ *children\_count*);

*children\_count* = 0;

b.displayBoard();

}

**if** (b.hasXWon()) {

System.***out***.println("Computer: I Won :-)!");

} **else** **if** (b.hasOWon()) {

System.***out***.println("Player Won.");

} **else** {

System.***out***.println("It's a draw!");

}

}

}

Output:

\*\*\*\*\*Board\*\*\*\*\*

| | |

| | |

| | |

Choose the player who start's the game?

1. Computer

2. User

2

-------------------------------------------------------------------------------------------------------

Your move:

Enter row(1 or 2 or 3)

2

Enter Column(1 or 2 or 3)

2

| | |

| O | |

| | |

Pruning at depth = 6

Pruning at depth = 6

Pruning at depth = 6

Pruning at depth = 7

Pruning at depth = 7

Pruning at depth = 5

Pruning at depth = 7

Pruning at depth = 7

Pruning at depth = 7

Pruning at depth = 7

Pruning at depth = 7

Pruning at depth = 5

Pruning at depth = 7

Pruning at depth = 5

Pruning at depth = 6

Pruning at depth = 6

Pruning at depth = 6

Pruning at depth = 7

Pruning at depth = 5

Pruning at depth = 6

Pruning at depth = 6

Pruning at depth = 6

Pruning at depth = 6

Pruning at depth = 4

Pruning at depth = 6

Pruning at depth = 7

Pruning at depth = 6

Pruning at depth = 6

Pruning at depth = 7

Pruning at depth = 7

Pruning at depth = 6

Pruning at depth = 7

Pruning at depth = 7

Pruning at depth = 7

Pruning at depth = 5

Pruning at depth = 7

Pruning at depth = 6

Pruning at depth = 6

Pruning at depth = 6

Pruning at depth = 6

Pruning at depth = 4

Pruning at depth = 6

Pruning at depth = 6

Pruning at depth = 6

Pruning at depth = 6

Pruning at depth = 6

Pruning at depth = 6

Pruning at depth = 5

Pruning at depth = 6

Pruning at depth = 5

Pruning at depth = 6

Pruning at depth = 6

Pruning at depth = 7

Pruning at depth = 7

Pruning at depth = 6

Pruning at depth = 6

Pruning at depth = 7

Pruning at depth = 7

Pruning at depth = 6

Pruning at depth = 6

Pruning at depth = 6

Pruning at depth = 7

Pruning at depth = 7

Pruning at depth = 7

Pruning at depth = 5

Pruning at depth = 7

Pruning at depth = 7

Pruning at depth = 7

Pruning at depth = 5

Pruning at depth = 7

Pruning at depth = 7

Pruning at depth = 7

Pruning at depth = 5

Pruning at depth = 6

Pruning at depth = 6

Pruning at depth = 7

Pruning at depth = 4

Pruning at depth = 7

Pruning at depth = 7

Pruning at depth = 6

Pruning at depth = 6

Pruning at depth = 6

Pruning at depth = 7

Pruning at depth = 6

Pruning at depth = 6

Pruning at depth = 6

Pruning at depth = 7

Pruning at depth = 5

Pruning at depth = 7

Pruning at depth = 5

Pruning at depth = 7

Pruning at depth = 7

Pruning at depth = 5

Pruning at depth = 6

Pruning at depth = 5

Pruning at depth = 7

Pruning at depth = 5

Pruning at depth = 6

Pruning at depth = 6

Pruning at depth = 7

Pruning at depth = 5

Pruning at depth = 6

Pruning at depth = 7

Pruning at depth = 5

Pruning at depth = 6

Pruning at depth = 7

Pruning at depth = 7

Pruning at depth = 7

Pruning at depth = 6

Pruning at depth = 6

Pruning at depth = 6

Pruning at depth = 7

Pruning at depth = 7

Pruning at depth = 7

Pruning at depth = 5

Pruning at depth = 6

Pruning at depth = 6

Pruning at depth = 6

Pruning at depth = 6

Pruning at depth = 4

Pruning at depth = 5

Pruning at depth = 6

Pruning at depth = 7

Pruning at depth = 7

Pruning at depth = 7

Pruning at depth = 5

Pruning at depth = 6

Pruning at depth = 5

Pruning at depth = 7

Pruning at depth = 7

Pruning at depth = 7

Pruning at depth = 5

Pruning at depth = 7

Pruning at depth = 6

Pruning at depth = 5

Pruning at depth = 3

Pruning at depth = 7

Pruning at depth = 7

Pruning at depth = 6

Pruning at depth = 6

Pruning at depth = 6

Pruning at depth = 5

Pruning at depth = 5

Pruning at depth = 5

Pruning at depth = 7

Pruning at depth = 6

Pruning at depth = 6

Pruning at depth = 6

Pruning at depth = 5

Pruning at depth = 5

Pruning at depth = 5

Pruning at depth = 7

Pruning at depth = 6

Pruning at depth = 7

Pruning at depth = 6

Pruning at depth = 7

Pruning at depth = 7

Pruning at depth = 7

Pruning at depth = 5

Pruning at depth = 6

Pruning at depth = 7

Pruning at depth = 7

Pruning at depth = 7

Pruning at depth = 5

Pruning at depth = 6

Pruning at depth = 7

Pruning at depth = 7

Pruning at depth = 7

Pruning at depth = 5

Pruning at depth = 7

Pruning at depth = 7

Pruning at depth = 7

Pruning at depth = 5

Pruning at depth = 7

Pruning at depth = 7

Pruning at depth = 7

Pruning at depth = 5

Pruning at depth = 3

Pruning at depth = 7

Pruning at depth = 6

Pruning at depth = 6

Pruning at depth = 6

Pruning at depth = 5

Pruning at depth = 5

Pruning at depth = 5

Pruning at depth = 7

Pruning at depth = 7

Pruning at depth = 6

Pruning at depth = 6

Pruning at depth = 6

Pruning at depth = 5

Pruning at depth = 5

Pruning at depth = 5

Pruning at depth = 7

Pruning at depth = 6

Pruning at depth = 5

Pruning at depth = 6

Pruning at depth = 5

Pruning at depth = 6

Pruning at depth = 7

Pruning at depth = 7

Pruning at depth = 7

Pruning at depth = 5

Pruning at depth = 7

Pruning at depth = 7

Pruning at depth = 7

Pruning at depth = 5

Pruning at depth = 6

Pruning at depth = 7

Pruning at depth = 7

Pruning at depth = 7

Pruning at depth = 5

Pruning at depth = 3

Pruning at depth = 6

Pruning at depth = 6

Pruning at depth = 5

Pruning at depth = 5

Pruning at depth = 5

Pruning at depth = 5

Pruning at depth = 7

Pruning at depth = 6

Pruning at depth = 7

Pruning at depth = 6

Pruning at depth = 7

Pruning at depth = 7

Pruning at depth = 7

Pruning at depth = 5

Pruning at depth = 5

Pruning at depth = 5

Pruning at depth = 5

Pruning at depth = 5

Pruning at depth = 3

Pruning at depth = 7

Pruning at depth = 7

Pruning at depth = 6

Pruning at depth = 7

Pruning at depth = 7

Pruning at depth = 7

Pruning at depth = 5

Pruning at depth = 5

Pruning at depth = 5

Pruning at depth = 5

Pruning at depth = 5

Pruning at depth = 3

Pruning at depth = 7

Pruning at depth = 7

Pruning at depth = 7

Pruning at depth = 6

Pruning at depth = 6

Pruning at depth = 5

Pruning at depth = 5

Pruning at depth = 5

Pruning at depth = 5

Pruning at depth = 7

Pruning at depth = 6

Pruning at depth = 7

Pruning at depth = 7

Pruning at depth = 7

Pruning at depth = 5

Pruning at depth = 5

Pruning at depth = 5

Pruning at depth = 5

Pruning at depth = 3

Pruning at depth = 6

Pruning at depth = 6

Pruning at depth = 6

Pruning at depth = 6

Pruning at depth = 4

Pruning at depth = 6

Pruning at depth = 6

Pruning at depth = 6

Pruning at depth = 6

Pruning at depth = 4

Pruning at depth = 6

Pruning at depth = 6

Pruning at depth = 6

Pruning at depth = 6

Pruning at depth = 4

Pruning at depth = 6

Pruning at depth = 6

Pruning at depth = 4

Pruning at depth = 6

Pruning at depth = 6

Pruning at depth = 4

Pruning at depth = 6

Pruning at depth = 6

Pruning at depth = 4

Pruning at depth = 2

Pruning at depth = 6

Pruning at depth = 6

Pruning at depth = 6

Pruning at depth = 6

Pruning at depth = 4

Pruning at depth = 6

Pruning at depth = 6

Pruning at depth = 6

Pruning at depth = 6

Pruning at depth = 4

Pruning at depth = 6

Pruning at depth = 6

Pruning at depth = 6

Pruning at depth = 6

Pruning at depth = 6

Pruning at depth = 6

Pruning at depth = 6

Pruning at depth = 6

Pruning at depth = 6

Pruning at depth = 6

Pruning at depth = 4

Pruning at depth = 6

Pruning at depth = 6

Pruning at depth = 6

Pruning at depth = 6

Pruning at depth = 7

Pruning at depth = 6

Pruning at depth = 4

Pruning at depth = 6

Pruning at depth = 6

Pruning at depth = 6

Pruning at depth = 6

Pruning at depth = 5

Pruning at depth = 6

Pruning at depth = 6

Pruning at depth = 6

Pruning at depth = 6

Pruning at depth = 4

Pruning at depth = 2

Pruning at depth = 6

Pruning at depth = 6

Pruning at depth = 6

Pruning at depth = 6

Pruning at depth = 4

Pruning at depth = 6

Pruning at depth = 6

Pruning at depth = 6

Pruning at depth = 6

Pruning at depth = 4

Pruning at depth = 6

Pruning at depth = 6

Pruning at depth = 6

Pruning at depth = 6

Pruning at depth = 4

Pruning at depth = 6

Pruning at depth = 6

Pruning at depth = 6

Pruning at depth = 6

Pruning at depth = 4

Pruning at depth = 6

Pruning at depth = 6

Pruning at depth = 6

Pruning at depth = 6

Pruning at depth = 4

Pruning at depth = 6

Pruning at depth = 6

Pruning at depth = 6

Pruning at depth = 6

Pruning at depth = 4

Pruning at depth = 2

Pruning at depth = 6

Pruning at depth = 6

Pruning at depth = 6

Pruning at depth = 6

Pruning at depth = 4

Pruning at depth = 6

Pruning at depth = 6

Pruning at depth = 6

Pruning at depth = 6

Pruning at depth = 6

Pruning at depth = 6

Pruning at depth = 6

Pruning at depth = 6

Pruning at depth = 6

Pruning at depth = 6

Pruning at depth = 6

Pruning at depth = 6

Pruning at depth = 4

Pruning at depth = 6

Pruning at depth = 6

Pruning at depth = 6

Pruning at depth = 6

Pruning at depth = 4

Pruning at depth = 6

Pruning at depth = 6

Pruning at depth = 6

Pruning at depth = 4

Pruning at depth = 6

Pruning at depth = 6

Pruning at depth = 6

Pruning at depth = 6

Pruning at depth = 6

Pruning at depth = 6

Pruning at depth = 6

Pruning at depth = 4

Pruning at depth = 2

Pruning at depth = 6

Pruning at depth = 6

Pruning at depth = 4

Pruning at depth = 6

Pruning at depth = 6

Pruning at depth = 4

Pruning at depth = 6

Pruning at depth = 6

Pruning at depth = 6

Pruning at depth = 6

Pruning at depth = 4

Pruning at depth = 6

Pruning at depth = 6

Pruning at depth = 6

Pruning at depth = 6

Pruning at depth = 4

Pruning at depth = 6

Pruning at depth = 6

Pruning at depth = 6

Pruning at depth = 6

Pruning at depth = 6

Pruning at depth = 6

Pruning at depth = 6

Pruning at depth = 6

Pruning at depth = 6

Pruning at depth = 6

Pruning at depth = 6

Pruning at depth = 6

Pruning at depth = 6

Pruning at depth = 7

Pruning at depth = 6

Pruning at depth = 7

Pruning at depth = 5

Pruning at depth = 6

Pruning at depth = 6

Pruning at depth = 6

Pruning at depth = 6

Pruning at depth = 6

Pruning at depth = 6

Pruning at depth = 2

Pruning at depth = 6

Pruning at depth = 6

Pruning at depth = 4

Pruning at depth = 6

Pruning at depth = 6

Pruning at depth = 6

Pruning at depth = 6

Pruning at depth = 7

Pruning at depth = 6

Pruning at depth = 4

Pruning at depth = 6

Pruning at depth = 6

Pruning at depth = 6

Pruning at depth = 6

Pruning at depth = 4

Pruning at depth = 6

Pruning at depth = 6

Pruning at depth = 6

Pruning at depth = 4

Pruning at depth = 6

Pruning at depth = 6

Pruning at depth = 6

Pruning at depth = 6

Pruning at depth = 6

Pruning at depth = 6

Pruning at depth = 6

Pruning at depth = 6

Pruning at depth = 6

Pruning at depth = 6

Pruning at depth = 6

Pruning at depth = 6

Pruning at depth = 6

Pruning at depth = 7

Pruning at depth = 6

Pruning at depth = 7

Pruning at depth = 6

Pruning at depth = 6

Pruning at depth = 6

Pruning at depth = 6

Pruning at depth = 6

Pruning at depth = 7

Pruning at depth = 6

Pruning at depth = 2

Pruning at depth = 6

Pruning at depth = 6

Pruning at depth = 4

Pruning at depth = 6

Pruning at depth = 6

Pruning at depth = 6

Pruning at depth = 6

Pruning at depth = 5

Pruning at depth = 6

Pruning at depth = 6

Pruning at depth = 6

Pruning at depth = 6

Pruning at depth = 4

Pruning at depth = 6

Pruning at depth = 6

Pruning at depth = 6

Pruning at depth = 6

Pruning at depth = 4

Pruning at depth = 6

Pruning at depth = 6

Pruning at depth = 6

Pruning at depth = 6

Pruning at depth = 6

Pruning at depth = 6

Pruning at depth = 6

Pruning at depth = 4

Pruning at depth = 6

Pruning at depth = 6

Pruning at depth = 6

Pruning at depth = 6

Pruning at depth = 6

Pruning at depth = 7

Pruning at depth = 6

Pruning at depth = 7

Pruning at depth = 5

Pruning at depth = 6

Pruning at depth = 6

Pruning at depth = 6

Pruning at depth = 6

Pruning at depth = 6

Pruning at depth = 6

Pruning at depth = 6

Pruning at depth = 6

Pruning at depth = 6

Pruning at depth = 6

Pruning at depth = 6

Pruning at depth = 7

Pruning at depth = 6

Pruning at depth = 7

Pruning at depth = 6

Pruning at depth = 6

Pruning at depth = 6

Pruning at depth = 6

Pruning at depth = 6

Pruning at depth = 7

Pruning at depth = 6

Pruning at depth = 2

Point: [0, 0] Score: 0

Point: [0, 1] Score: 0

Point: [0, 2] Score: 0

Point: [1, 0] Score: 0

Point: [1, 2] Score: 0

Point: [2, 0] Score: 0

Point: [2, 1] Score: 0

Point: [2, 2] Score: 0

Children count: 3393

X | | |

| O | |

| | |

-------------------------------------------------------------------------------------------------------

Your move:

Enter row(1 or 2 or 3)

1

Enter Column(1 or 2 or 3)

3

X | | O |

| O | |

| | |

Pruning at depth = 5

Pruning at depth = 5

Pruning at depth = 4

Pruning at depth = 4

Pruning at depth = 4

Pruning at depth = 5

Pruning at depth = 4

Pruning at depth = 4

Pruning at depth = 4

Pruning at depth = 5

Pruning at depth = 3

Pruning at depth = 5

Pruning at depth = 3

Pruning at depth = 5

Pruning at depth = 5

Pruning at depth = 3

Pruning at depth = 4

Pruning at depth = 4

Pruning at depth = 4

Pruning at depth = 5

Pruning at depth = 3

Pruning at depth = 4

Pruning at depth = 4

Pruning at depth = 5

Pruning at depth = 3

Pruning at depth = 4

Pruning at depth = 5

Pruning at depth = 3

Pruning at depth = 4

Pruning at depth = 5

Pruning at depth = 4

Pruning at depth = 4

Pruning at depth = 4

Pruning at depth = 5

Pruning at depth = 5

Pruning at depth = 5

Pruning at depth = 3

Pruning at depth = 4

Pruning at depth = 4

Pruning at depth = 4

Pruning at depth = 4

Pruning at depth = 2

Pruning at depth = 4

Pruning at depth = 4

Pruning at depth = 4

Pruning at depth = 4

Pruning at depth = 5

Pruning at depth = 5

Pruning at depth = 4

Pruning at depth = 4

Pruning at depth = 4

Pruning at depth = 3

Pruning at depth = 5

Pruning at depth = 5

Pruning at depth = 5

Pruning at depth = 3

Pruning at depth = 5

Pruning at depth = 4

Pruning at depth = 3

Pruning at depth = 4

Pruning at depth = 4

Pruning at depth = 5

Pruning at depth = 5

Pruning at depth = 4

Pruning at depth = 4

Pruning at depth = 4

Pruning at depth = 4

Pruning at depth = 4

Pruning at depth = 2

Pruning at depth = 4

Pruning at depth = 4

Pruning at depth = 5

Pruning at depth = 2

Point: [0, 1] Score: -101

Point: [1, 0] Score: -100

Point: [1, 2] Score: -100

Point: [2, 0] Score: 0

Point: [2, 1] Score: 0

Point: [2, 2] Score: -91

Children count: 532

X | | O |

| O | |

X | | |

-------------------------------------------------------------------------------------------------------

Your move:

Enter row(1 or 2 or 3)

2

Enter Column(1 or 2 or 3)

3

X | | O |

| O | O |

X | | |

Pruning at depth = 3

Pruning at depth = 2

Pruning at depth = 2

Point: [0, 1] Score: -109

Point: [1, 0] Score: 90

Point: [2, 1] Score: -100

Point: [2, 2] Score: -91

Children count: 19

X | | O |

X | O | O |

X | | |

Computer: I Won :-)!