2025

AI REPORT 2

Presented by Ioanna Avanidi(4977) & Ioannis kokkinis(5109)

Άσκηση 2
Κατασκευή Παιγνίου

Ioanna Avanidi(4977) & Giannis Kokkinis(5109)
MINMAX Algorithm

Εισαγωγή

Το ζητούμενο αυτής της εργασίας είναι η δημιουργία ενός παιχνιδιού τύπου tic-tac-toe με χρήση του αλγορίθμου MINMAX. Το παιχνίδι λαμβάνει χώρα σε πλέγμα 4x3 και παίζεται από δύο παίκτες:

- Ο MAX που παίζει με το γράμμα X
- Ο MIN που παίζει με το γράμμα O

Οι παίκτες παίζουν εναλλάξ και τοποθετούν τα γράμματα τους σε κενές θέσεις. Το παιχνίδι τερματίζει όταν σχηματιστεί οριζόντια, κάθετα ή διαγώνια, μια από τις εξής τελικές τριάδες:

- XOX, όπου νικητής είναι ο MAX
- OXO, όπου νικητής είναι ο MIN

Ή όταν γεμίσει το πλέγμα χωρίς να σχηματιστεί καμία από τις παραπάνω τριάδες(τότε έχουμε ισοπαλία).

Περιγραφή Υλοποίησης

Ορισμός Κατάστασης

Η κατάσταση του παιγνίου αναπαρίσταται με:

- Έναν πίνακα grid που περιέχει ¨X¨, ¨O¨ ή ¨¨(κενό)

```
private String[][] grid = new String[4][3];
```

- Μια μεταβλητή isXplay που καθορίζει τη σείρα που παίζουν οι παίκτες(Όταν είναι true παίζει ο X, όταν είναι false παίζει ο O)

```
private boolean isXplay;
```

- Μια μεταβλητή minOrMax που υποδεικνύει αν ο κόμβος ανήκει στον MAX(true) ή στον MIN(false)

```
private boolean minOrMax;
```

- Μια λίστα από παιδιά-κόμβους kids για το Minimax tree

```
private List<Node> kids = new ArrayList<Node>();
```

- Μια μεταβλητή value που καθορίζεται από τη utilityFunction

```
private int value;
```

Utility Function

Η συνάρτηση ελέγχει όλους τους δυνατούς τρόπους σχηματισμού κάθε τριάδας(οριζόντια, κάθετα, διαγώνια) και επιστρέφει έναν αριθμό ανάλογα με το τι συμβαίνει στο grid:

- Αν σχηματιστεί το XOX, επιστρέφει 1

```
private int UtilityFunction(){

    if (
        (grid[0][0].equals(anObject:"X") && grid[0][1].equals(anObject:"O") && grid[0][2].equals(anObject:"X"))
        ||
        (grid[0][0].equals(anObject:"X") && grid[1][0].equals(anObject:"O") && grid[2][0].equals(anObject:"X"))
        ||
        (grid[0][0].equals(anObject:"X") && grid[1][1].equals(anObject:"O") && grid[2][2].equals(anObject:"X"))
        ||
        (grid[0][1].equals(anObject:"X") && grid[1][1].equals(anObject:"O") && grid[2][1].equals(anObject:"X"))
        ||
        (grid[0][2].equals(anObject:"X") && grid[1][2].equals(anObject:"O") && grid[3][2].equals(anObject:"X"))
        ||
        (grid[1][0].equals(anObject:"X") && grid[1][1].equals(anObject:"O") && grid[1][2].equals(anObject:"X"))
        ||
        (grid[1][0].equals(anObject:"X") && grid[2][0].equals(anObject:"O") && grid[3][0].equals(anObject:"X"))
        ||
        (grid[1][1].equals(anObject:"X") && grid[2][1].equals(anObject:"O") && grid[3][1].equals(anObject:"X"))
        ||
        (grid[1][2].equals(anObject:"X") && grid[2][1].equals(anObject:"O") && grid[3][0].equals(anObject:"X"))
        ||
        (grid[1][2].equals(anObject:"X") && grid[2][2].equals(anObject:"O") && grid[3][2].equals(anObject:"X"))
        ||
        (grid[2][0].equals(anObject:"X") && grid[2][1].equals(anObject:"O") && grid[2][2].equals(anObject:"X"))
        ||
        (grid[3][0].equals(anObject:"X") && grid[3][1].equals(anObject:"O") && grid[3][2].equals(anObject:"X"))
    ){
        return 1;
```

- Αν σχηματιστεί το ΟΧΟ, επιστρέφει -1

```
}else{
    if (
    (grid[0][0].equals(anObject:"O") && grid[0][1].equals(anObject:"X") && grid[0][2].equals(anObject:"O"))
    ||
    (grid[0][0].equals(anObject:"O") && grid[1][0].equals(anObject:"X") && grid[2][0].equals(anObject:"O"))
    ||
    (grid[0][0].equals(anObject:"O") && grid[1][1].equals(anObject:"X") && grid[2][2].equals(anObject:"O"))
    ||
    (grid[0][1].equals(anObject:"O") && grid[1][1].equals(anObject:"X") && grid[2][1].equals(anObject:"O"))
    ||
    (grid[0][2].equals(anObject:"O") && grid[1][2].equals(anObject:"X") && grid[2][2].equals(anObject:"O"))
    ||
    (grid[1][0].equals(anObject:"O") && grid[1][1].equals(anObject:"X") && grid[1][2].equals(anObject:"O"))
    ||
    (grid[1][0].equals(anObject:"O") && grid[2][0].equals(anObject:"X") && grid[3][0].equals(anObject:"O"))
    ||
    (grid[1][1].equals(anObject:"O") && grid[2][1].equals(anObject:"X") && grid[3][1].equals(anObject:"O"))
    ||
    (grid[1][2].equals(anObject:"O") && grid[2][1].equals(anObject:"X") && grid[3][0].equals(anObject:"O"))
    ||
    (grid[1][2].equals(anObject:"O") && grid[2][2].equals(anObject:"X") && grid[3][2].equals(anObject:"O"))
    ||
    (grid[2][0].equals(anObject:"O") && grid[2][1].equals(anObject:"X") && grid[2][2].equals(anObject:"O"))
    ||
    (grid[3][0].equals(anObject:"O") && grid[3][1].equals(anObject:"X") && grid[3][2].equals(anObject:"O"))
    ){
        return -1;
```

- Αν γεμίσει το πλέγμα χωρίς νικητή, επιστρέφει 0

```
}else{
    if (isGridFilled()){
        return 0;
    }
}
```

- Αν το παιχνίδι δεν έχει ολοκληρωθεί επιστρέφει -2

```
    return -2; //not finish yet
}
```

GenerateKids

Η συνάρτηση generateKids() έχει ως στόχο την δημιουργία όλων των πιθανών παιδιών ενός συγκεκριμένου κόμβου, δηλαδή όλων των πιθανών καταστάσεων του παιχνιδιού που μπορούν να προκύψουν από μία μόνο κίνηση του παίκτη X ή του παίκτη O.

```
private void generateKids(){
    //generate kids here
    for(int i=0; i<4; i++){
        for (int j=0; j<3; j++){
            if(grid[i][j].length()==0){ //empty slot found
                String[][] tempGrid = new String[4][3];

                //copy grid to temp grid to set on new node
                for (int c1=0; c1<4; c1++){
                    for (int c2=0; c2<3; c2++){
                        tempGrid[c1][c2] = grid[c1][c2];
                    }
                }
                //set the possible move
                if(isXplay){
                    tempGrid[i][j]="X";
                }else{
                    tempGrid[i][j]="O";
                }
                Node kid = new Node(tempGrid, !isXplay, !minOrMax);
                this.kids.add(kid);
            }
        }
    }
}
```

MiniMax

Η συνάρτηση miniMax αποτελεί τον πυρήνα του ζητούμενου αλγορίθμου, ο οποίος σύμφωνα με την εκφώνηση πρέπει να υλοποιηθεί αναδρομικά, κάτι που επιτυγχάνεται ακολουθώντας την παρακάτω διαδικασία.

```
public int miniMax(){

    int endValue = UtilityFunction();
    if (endValue!=-2){ //we found end state here
        value=endValue;
        return endValue;
    }

    //generate next nodes
    generateKids();

    if(minOrMax){ //MAX LOOKING
        int max = 0;
        for(Node kid: kids){
            max = Math.max(max, kid.miniMax());
        }
        value=max;
        return max;

    }else{ //MIN LOOKING
        int min=kids.get(index:0).miniMax();
        for (int kidInd=1; kidInd<kids.size(); kidInd++){
            Math.min(min, kids.get(kidInd).miniMax());
        }
        value=min;
        return min;

    }

}
```

Αρχικά, καλείται η συνάρτηση UtilityFunction() με σκοπό να διαπιστωθεί αν η παρούσα κατάσταση του παιχνιδιού είναι η τελική. Σε περίπτωση που η συνάρτηση επιστρέψει -1,0 ή 1 (δηλαδή ήττα ,ισοπαλία ή νίκη) τότε η συνάρτηση τερματίζεται και επιστρέφει την τιμή αυτή.

```
int endValue = UtilityFunction();
if (endValue!=-2){ //we found end state here
    value=endValue;
    return endValue;
}
```

Σε περίπτωση που η κατάσταση δεν είναι τελική(η συνάρτηση επιστρέφει -2), τότε με την χρήση της συνάρτησης generateKids() παράγονται οι επόμενοι δυνατοί κόμβοι/καταστάσεις που περιέχουν όλους τους δυνατούς συνδυασμούς που μπορεί να επιλέξει ο παίκτης.

```
//generate next nodes
generateKids();
```

Έπειτα ανάλογα με την σειρά που ανήκει στον παίκτη MAX ή στον παίκτη MIN, η miniMax καλείται αναδρομικά για κάθε παιδί, αναζητώντας την μέγιστη ή την ελάχιστη δυνατή τιμή αντίστοιχα και την επιστρέφει.

```
if(minOrMax){ //MAX LOOKING
    int max = 0;
    for(Node kid: kids){
        max = Math.max(max, kid.miniMax());
    }
    value=max;
    return max;

}else{ //MIN LOOKING
    int min=kids.get(index:0).miniMax();
    for (int kidInd=1; kidInd<kids.size(); kidInd++){
        Math.min(min, kids.get(kidInd).miniMax());
    }
    value=min;
    return min;

}
```

Με αυτό τον τρόπο, ο αλγόριθμος MiniMax εντοπίζει την καλύτερη δυνατή επόμενη κίνηση για τον MAX, εξασφαλίζοντας την βέλτιστη πορεία βάσει της τρέχουσας πληροφορίας.

Ροή παιχνιδιού

Το παιχνίδι ξεκινάει με την αρχικοποίηση ενός πίνακα grid.

```
private String grid[][] = { // 4X3
    {"","",""},
    {"","",""},
    {"","",""},
    {"","",""}
};
```

Για να ξεκινήσει το παιχνίδι από μία αρχική κατάσταση, στην οποία έχουν ήδη τοποθετηθεί τα σύμβολα “X” και “O” σε μη διαδοχικές θέσεις, χρησιμοποιούμε τον constructor TicTacToe()

```
public TicTacToe(){
    Random rand = new Random();
    int posX1,posX2, posO1, posO2;

    posX1 = rand.nextInt(bound:4);
    posX2 = rand.nextInt(bound:3);


    posO1 = rand.nextInt(bound:4);
    posO2 = rand.nextInt(bound:3);
    while (!checkIfSucPositions(posX1,posX2,posO1,posO2)){ //check position
        posO1 = rand.nextInt(bound:4);
        posO2 = rand.nextInt(bound:3);
    }


    this.grid[posX1][posX2] = "X";
    this.grid[posO1][posO2] = "O";


}
```

Για να ελέγξουμε τις θέσεις στις οποίες τοποθετούνται τα “X” και “O”, χρησιμοποιούμε την συνάρτηση checkIfSucPositions(). Η συνάρτηση αυτή ελέγχει αν τα σύμβολα έχουν τοποθετηθεί σε διαδοχική σειρά.

```
private Boolean checkIfSucPositions(int x1, int y1,int x2,int y2){

    /**
     *
     * SAME POSITION
     *
     */

    if ((x1==x2) && (y1==y2)){
        return false;
    }

    /**
     *  Right
    */

    if ((y1+1==y2) && (y1+1<3)){
        return  false;
    }

    /**
     * left
     *
     */
    if (y1-1==y2 && (y1-1>=0)){
        return false;
    }

    /**
     *
     * DOWN
     */
    if ((x1+1==x2) && (x1+1<4)){
        return false;
    }
```

```
    /**
     *
     * UP
     */
    if ((x1-1==x2) && (x1-1>=0)){
        return false;
    }

    /*
     * UP RIGHT
     */
    if ( ((x1-1==x2)&&(y1+1==y2)) && (x1-1>=0) && (y1+1<3)){
        return false;
    }

    /**
     * UP LEFT
     *
     */
    if (((x1-1==x2)&&(y1-1==y2)) && (x1-1>=0) && (y1-1>=0)){
        return false;
    }

    /**
     * DOWN RIGHT
     */
    if (((x1+1==x2)&&(y1+1==y2)) && (x1+1<4) && (y1+1<3)){
        return false;
    }

    /**
     * DOWN LEFT
     */
    if (((x1+1==x2)&&(y1-1==y2)) && (x1+1<4) && (y1-1>=0)){
        return false;
    }


    return true;
}
```

Με αυτό τον έλεγχο διασφαλίζεται ότι τα σύμβολα δεν τοποθετούνται σε διαδοχική σειρά, επιτρέποντας την ορθή εκκίνηση του παιχνιδιού.

Στην συνέχεια έχουμε φτιάξει 2 συναρτήσεις Xturn() και Oturn() που προσομοιώνουν τους παίκτες MAX και MIN

```
/**
 *
 *  return true -> still playing
 *  return false -> stop playing
 */
private boolean Xturn(){


    Node decNode = new Node(grid, updateIsXPlay:true, updateMinOrMax:true);
    int winValue = decNode.miniMax();
    if(decNode.getKids().isEmpty()){ //this means the game finish!
        printWhoWins(winValue);
        return false; //stop playing
    }
    copyGrid(decNode.getKids().get(getWinningNodeIndex(decNode, winValue)).getGrid());
    printGrid();
    return true; //still playing
}

private boolean Oturn(){

    Node decNode = new Node(grid, updateIsXPlay:false, updateMinOrMax:true);
    int winValue = decNode.miniMax();
    if(decNode.getKids().isEmpty()){ //this means the game finish!
        printWhoWins(winValue);
        return false; //stop playing
    }
    copyGrid(decNode.getKids().get(getWinningNodeIndex(decNode, winValue)).getGrid());
    printGrid();
    return true;//still playing

}
```

Κάθε φορά που παίζει ένας παίκτης , δημιουργείται ένας κόμβος αντιγράφοντας τον grid του παιχνιδιού για να αναπαραστήσουμε την τρέχουσα κατάσταση του παιχνιδιού. Έπειτα, καλούμε την συνάρτηση miniMax() , η οποία εφαρμόζει τον αλγόριθμο που εξηγήσαμε παραπάνω και επιστρέφει την εκτιμώμενη τιμή του βέλτιστου επόμενου κόμβου.

Στην περίπτωση που η τρέχουσα κατάσταση είναι τελικός κόμβος (δηλαδή έχει προκύψει νικητής ή ισοπαλία) τότε η συνάρτηση miniMax() δεν προχωρά στην δημιουργία παιδιών, οπότε τερματίζεται το παιχνίδι. Για να βρούμε τον νικητή , χρησιμοποιούμε την συνάρτηση printWhoWins() η οποία ανάλογα με τις τιμέ (1,0,-1) δείχνει και το ποιος κέρδισε.

```
private void printWhoWins(int winValue){
    if (winValue==1){
        System.out.println(x:"X WINS!");
    }else{
        if (winValue==0){
            System.out.println(x:"TIE!");
        }else{
            System.out.println(x:"O WINS!");
        }
    }
}
```

Αν η τρέχουσα κατάσταση δεν αποτελεί τελικό κόμβο, τότε με βάση την τιμή winValue που επιστρέφει ο αλγόριθμος miniMax(), επιλέγουμε τον επόμενο κόμβο που αντιστοιχεί στη βέλτιστη κίνηση. Στην συνέχεια, αντιγράφουμε το grid του επιλεγμένου κόμβου στο κύριο grid του παιχνιδιού.

Για την εκτέλεση του παιχνιδιού χρησιμοποιείται η συνάρτηση Play().

Η συνάρτηση αυτή είναι υπεύθυνη για τον καθορισμό του παίκτη που έχει τη σειρά να παίξει, καθώς και για τον έλεγχο τερματισμού του παιχνιδιού, με βάση τις τιμές που επιστρέφουν οι συναρτήσεις Xturn() και Oturn().

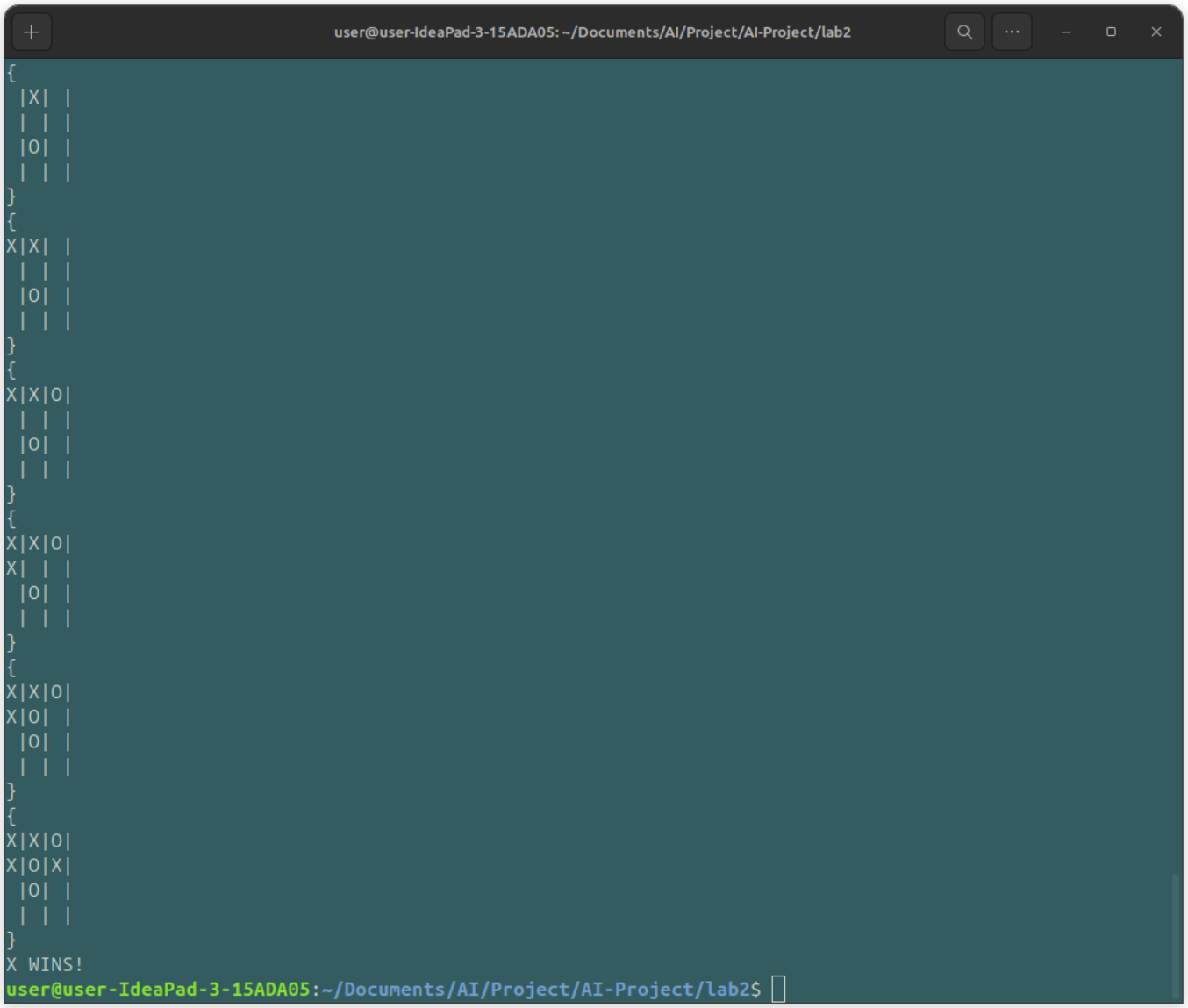
```
public void play(){


    boolean stillPlaying = true;

    while(stillPlaying){
        stillPlaying = Xturn();
        if (stillPlaying==false){
            break;
        }
        try {
            Thread.sleep(millis:2000);
        } catch (Exception e) {
            e.printStackTrace();
        }
        stillPlaying = Oturn();


    }
}
```

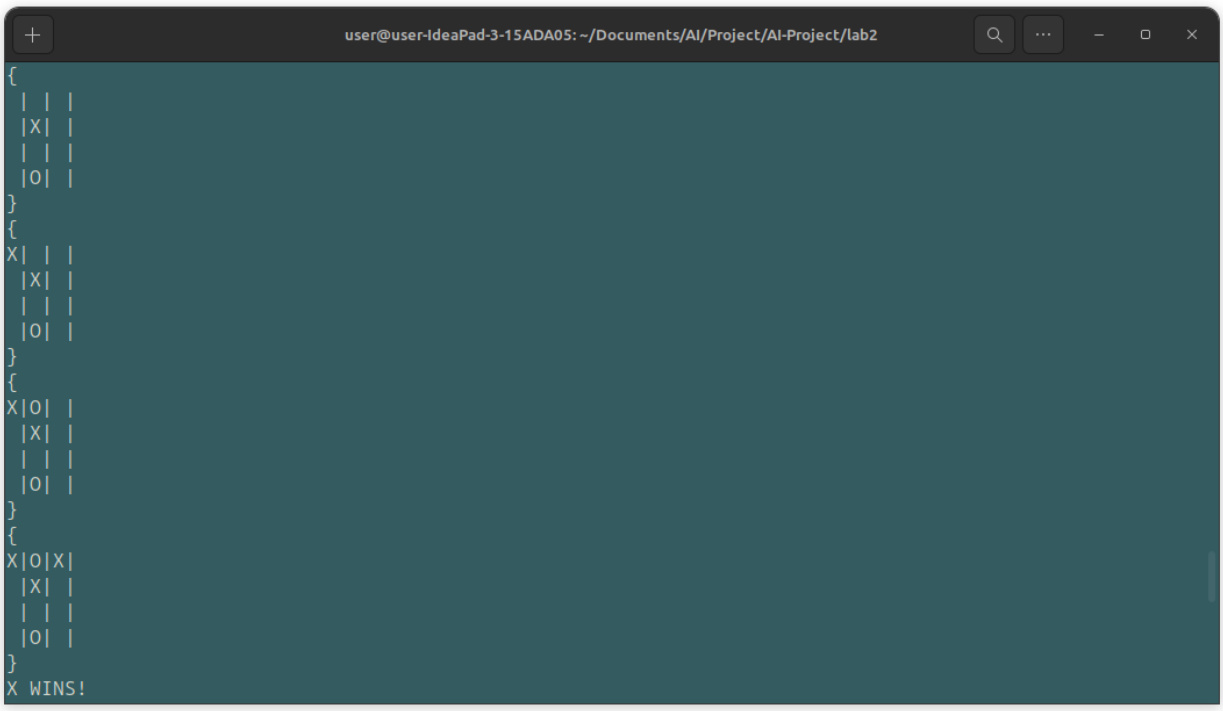
Για να το πετύχει αυτό, η συνάρτηση Play() αρχικοποιεί μία μεταβλητή stillPlaying, η οποία χρησιμοποιείται για να ελέγχει αν το παιχνίδι βρίσκεται ακόμα σε εξέλιξη. Αν κάποια από τις συναρτήσεις επιστρέψει false, αυτό υποδηλώνει είτε την ύπαρξη νικητή είτε την επίτευξη ισοπαλίας, οπότε και το παιχνίδι τερματίζει.

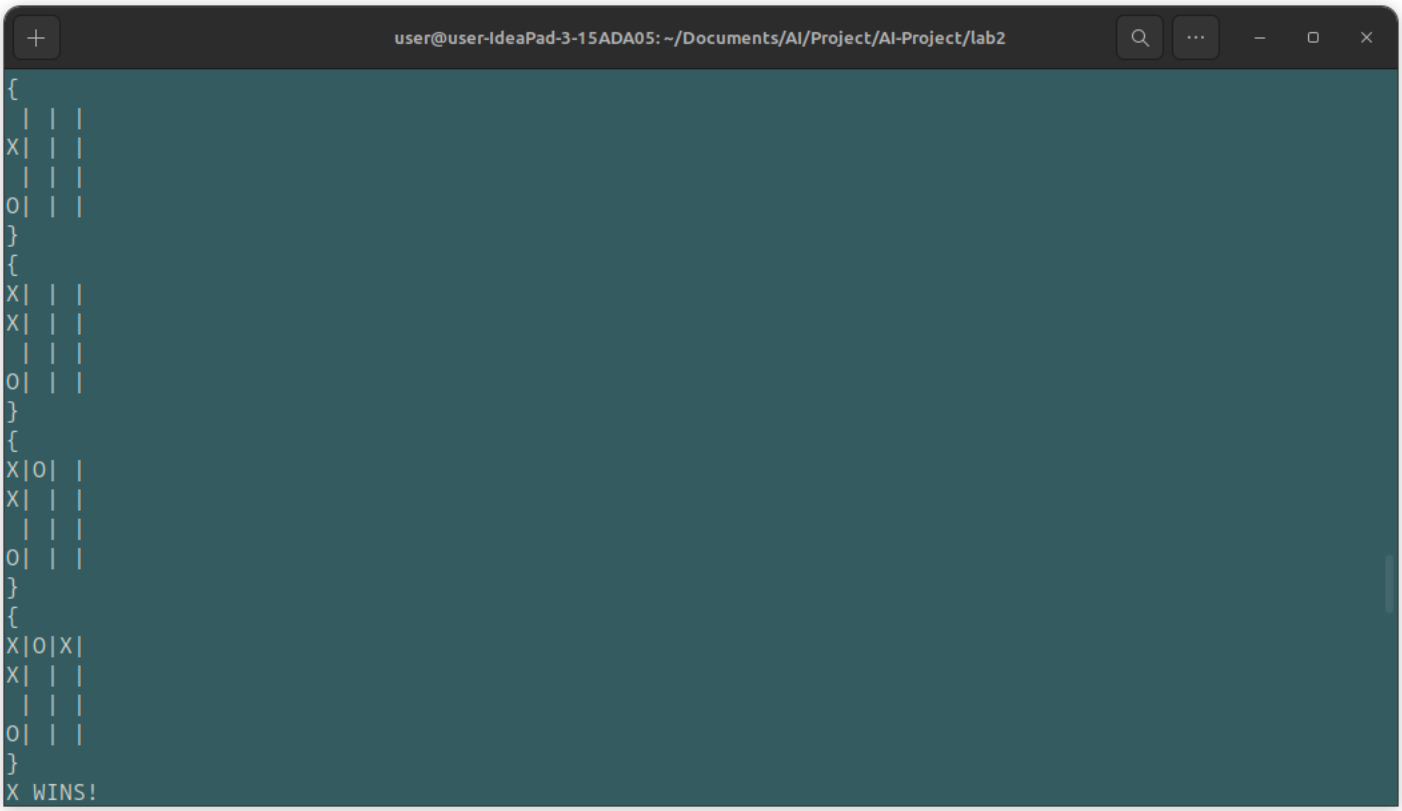
Για αυτό τον λόγο, οι συναρτήσεις Xturn() και Oturn() είναι σχεδιασμένες να επιστρέφουν τιμές boolean.

Ολοκληρώσαμε έτσι την υλοποίηση μας, αξιοποιώντας τον αναδρομικό αλγόριθμο MiniMax() σε συνδυασμό με τους κατάλληλους ελέγχους για τον προσδιορισμό της τερματικής κατάστασης και την ορθή εναλλαγή των παικτών. Η προσέγγιση αυτή επιτρέπει στο σύστημα να λαμβάνει βέλτιστες αποφάσεις σε κάθε βήμα του παιχνιδιού.

Παρακάτω σας παρουσιάζουμε ενδεικτικά αποτελέσματα δοκιμών:






user@user-IdeaPad-3-15ADA05: ~/Documents/AI/Project/AI-Project/lab2
{
 | | |
X| | |
 | | |
O| | |
}
{
X| | |
X| | |
 | | |
O| | |
}
{
X|O| |
X| | |
 | | |
O| | |
}
{
X|O|X|
X| | |
 | | |
O| | |
}
X WINS!