

Παράδειγμα εκτέλεσης:

Το πρόγραμμα ζητάει αρχικά από τον χρήστη το βάθος που θα χρησιμοποιηθεί στον αλγόριθμο MiniMax και στην συνέχεια ποιος θα παίξει πρώτος(1 για υπολογιστή, 2 για άνθρωπο). Στη συνέχεια ζητείται από τον χρήστη την επιθυμητή κίνηση, κάθε φορά που είναι η σειρά του μέχρι να τελειώσει το παιχνίδι.

```
Please choose depth:
9
Computer's turn: 1
Human's turn: 2
1
Move played: 1
3
Move played: 6
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
|X| | | | | |
|X| | | |O| | | |State evaluation: 0
Move played: 1
3
Move played: 3
| | | | | | | | |
| | | | | | | |
| | | | | | | |
|X| | | | | | | |
|O| | | | | | | |
|X| | | | | | | |
|X| | | | | | | |
|X| |O|O| | | | |State evaluation: 0
Move(1-7):
1
Move played: 1
2
Move played: 4
| | | | | | | | |
| | | | | | | |
|X| | | | | | | |
|O| | | | | | | |
|X| | | | | | | |
|X|X|O|O| | | | |State evaluation: 0
Move(1-7):
4
Move played: 4
| | | | | | | |
| | | | | | | |
| | | | | | | |
|X| | | |O| | | | |State evaluation: 0
Move(1-7):
5
Move played: 5
| | | | | | | | |
| | | | | | | |
|X| | | | | | | |
|O| | | | | | | |
|X| | | | | | | |
|X|X|O|O| | | | |State evaluation: -1
Move(1-7):
4
Move played: 2
| | | | | | | | |
| | | | | | | |
|X| | | | | | | |
|O| | | | | | | |
|X| | | | | | | |
|X|X|O|O| | | | |State evaluation: 0
Move(1-7):
4
Move played: 4
| | | | | | | | |
| | | | | | | |
|X| | | | | | | |
|O| | | | | | | |
|X| | | |O| | | | |State evaluation: 0
|X|X|O|O|O|O|X| | |State evaluation: 0
```

```
Move(1-7):
5
Move played: 5
| | | | | | | | | |
| | | | | | | |
|X| | | | | | | |
|O| | | |X| | | | |
|X| | | |O|O| | | |
|X|X|O|O|O|O|X| | |State evaluation: 0
Move(1-7):
6
Move played: 6
| | | | | | | | | |
| | | | | | | |
|X| | | | | | | |
|O| | | |X| | | | |
|X| | | |O|O| | | |
|X|X|O|O|O|O|X| | |State evaluation: 0
Move(1-7):
6
Move played: 2
| | | | | | | | | |
| | | | | | | |
|X| | | | | | | |
|O| | | |X| | | | |
|X| | | |O|O| | | |
|X|X|O|O|O|O|X| | |State evaluation: 1
|X|X|O|O|O|O|X| | |State evaluation: 0
Move(1-7):
5
Move played: 5
| | | | | | | | | | |
| | | | | | | |
|X| | | |O| | | | |
|O| |X| |X| |O|O| | |
|X| |X| |O|O|O|O| | |
|X|X|O|O|O|O|X| | |State evaluation: 198
Computer wins!
```

Στο πιο πάνω παράδειγμα, οι κινήσεις του χρήστη αντιστοιχούν στις κινήσεις που έκανε ο υπολογιστής από την υλοποίηση <http://www.mathsisfun.com/games/connect4.html>, στο επίπεδο hard.

Ευρετική συνάρτηση:

Χρησιμοποιούμε την ευρετική συνάρτηση, η οποία επιστρέφει τιμές με τον εξής αλγόριθμο:

$v <- 0$

Για κάθε στήλη

 Εάν ο max συμπληρώνει τετράδα, αν επιλέξει αυτήν τη στήλη

 Εάν είναι η σειρά του

$v <- v + 100$

 αλλιώς

$v <- v + 1$

 τέλος εάν

 Εάν ο min συμπληρώνει τετράδα, αν επιλέξει αυτήν τη στήλη

 Εάν είναι η σειρά του

$v <- v - 100$

 Αλλιώς

$v <- v - 1$

 Τέλος εάν

 Τέλος εάν

Τέλος για

Επίστρεψε v

Η μέθοδος που υλοποιεί την παραπάνω ευρετική είναι η `evaluate()`, η οποία υλοποιείται στην κλάση `State` και καλεί τις `sumThreats()` και `isThreat(int column)`.

Όταν φτάσουμε σε **State** όπου ο υπολογιστής δεν μπορεί να κάνει κάποια κίνηση ώστε να αποτρέψει την ήττα του (δηλαδή απειλείται σε περισσότερα από ένα σημεία) τότε αγνοεί την ευρετική και κάνει τυχαία κίνηση αφού η ήττα είναι προκαθορισμένη.

Υλοποίηση:

Κλάσεις: `Game`, `Player`, `State`, `Main`

Κλάση `Game`:

Η κλάση αναπαριστά μία παρτίδα, η οποία υλοποιείται στον κατασκευαστή της κλάσης.

Μέχρι η κατάσταση να είναι τελική ή μέχρι να έχουμε ισοπαλία, ζητάμε από τον χρήστη την επιθυμητή κίνηση όταν είναι η σειρά του και όταν είναι η σειρά του υπολογιστή, καλούμε την μέθοδο `play`, με παράμετρο το βάθος της αναζήτησης, από την κλάση `Player`, η οποία υλοποιεί τον αλγόριθμο μινιμαξ και επιστρέφει την κίνηση σύμφωνα με τον αλγόριθμο.

Κλάση Player:

Η κλάση υλοποιεί 3 μεθόδους την **play** την **min** και την **max** και 2 μεταβλητές *inintDepth*(μεγιστο βαθος) ,*move* (καλύτερη κίνηση). Όταν φτάσει η σειρά του υπολογιστή για να παίξει καλείται από την κλάση Game η μέθοδος **play** η οποία παίρνει σαν ορίσματα το μέγιστο βαθος που μπορεί να επεκταθεί το δέντρο και την κατάσταση του πίνακα που αναπαριστά το board του score4 και επιστρέφει την θέση του πίνακα που θα παίξει ο υπολογιστής. Αρχικά η **play** δημιουργεί όλες τις πιθανές καταστάσεις όπου ο υπολογιστής μπορεί να κάνει κίνηση και τις αποθηκεύει σε λίστα τύπου State, ύστερα κάνει Initialize τις μεταβλητές *alpha*, *beta* και καλεί την **max**. Η **max** δεχεται σαν ορίσμα την υπάρχουσα κατάσταση του Board τους score4 μια μεταβλητή που ελέγχει το βαθος *depth* και δυο μεταβλητές *alpha* και *beta* σε περίπτωση που γίνει πριονισμα του δέντρου και επιστρέφει μια βαθμολογία που αντιστοιχεί στην καλύτερη θέση που πρέπει να παίξει ο παίκτης ώστε να κερδίσει και να μην χάσει (Η **max** αναπαριστά τον αντιπαλο του υπολογιστή). Αρχικά ελέγχει αν έχουμε φτάσει σε μέγιστο βαθος ή αν ένας από τους δυο παίκτες έχει συμπληρώσει τετραδά, αν η συνθήκη είναι αληθής επιστρέφεται η **evaluate** της υπάρχουσας κατάστασης (του πίνακα του score4), αν δεν ισχύει η συνθήκη η **max** παράγει όλες τις πιθανές καταστάσεις του πίνακα στις οποίες μπορεί να παίξει ο αντιπαλος του υπολογιστή και τις αποθηκεύει σε μια λίστα. Στη συνέχεια για κάθε κατάσταση καλεί την **min** δίνοντας για ορίσμα την κατάσταση το, *depth-1* και τα *alpha* , *beta* και αποθηκεύει το αποτέλεσμα της **min** (ίδια υλοποίηση με **max** με την διαφορά του ότι η **min** θέλει να κερδίσει ο υπολογιστής). Ύστερα συγκρίνει το αποτέλεσμα της **min** με μια μεταβλητή που ορίσαμε στην αρχή της **max** και αν το αποτέλεσμα είναι μεγαλύτερο αποθηκεύεται στην μεταβλητή ενώ παράλληλα θετούμε την *move* ίση με την καλύτερη υπάρχουσα κίνηση εκείνη τη στιγμή. Επίσης όταν η αναδρομή επιστρέψει το πρώτο στάδιο, όπου *depth == initDepth* επιστρέφεται ο αριθμός της στήλης όπου αντιστοιχεί η καλύτερη κίνηση. Τέλος γίνεται ένας έλεγχος ο οποίος κοβεί το δέντρο εαν αυτο δεν χρειάζεται να επεκταθεί διότι βρεθηκε η καλύτερη πιθανή κίνηση. Ομοίως για την **min**, χωρίς να επιστρέφει την στήλη.

Κλάση State:

Η κλάση State αναπαριστά μια κατάσταση του παιχνιδιού. Δηλαδή τον πίνακα που έχει προκύψει μετά από τις κινήσεις που έχουν παιχτεί (**table**), ποιός είναι ο παίκτης που πρέπει να παίξει (**playerTurn**) και συνεπώς ποιός ήταν ο τελευταίος, ποιά ήταν η τελευταία κίνηση που παίχτηκε (**lastMove**), έναν πίνακα που έχει αποθηκεύσει πόσες κινήσεις έχουν παιχτεί σε κάθε στήλη (**count**) και μια λίστα με τον αριθμό των πιθανών κινήσεων (**possMoves**). Όταν αρχικά δημιουργηθεί ένα αντικείμενο State, καλείται ο constructor που θα αρχικοποιήσει τον πίνακα, με παραμέτρους τις διαστάσεις του πίνακα και τον πρώτο παίκτη. Η κλάση περιλαμβάνει άλλους δύο διαφορετικούς constructors. Ο ένας επιστρέφει ένα αντίγραφο της τρέχουσας κατάστασης (χρησιμοποιείται στην **evaluate** μέσω της **isThreat**). Ο 3ος καλείται όταν θα γίνει κάποια κίνηση μέσω της **move**, με παραμέτρους την τρέχουσα κατάσταση και έναν ακέραιο που αναπαριστά την στήλη στην οποία θα γίνει κίνηση και επιστρέφει την καινούργια State. Επίσης η κλάση περιλαμβάνει την μέθοδο **move**, όπως αναφέρθηκε παραπάνω, η οποία παίρνοντας παράμετρο την στήλη στην οποία θα γίνει η επόμενη κίνηση, ελέγχει αν επιτρέπεται η κίνηση, και αν επιτρέπεται καλεί την **State** και επιστρέφει το αντικείμενο που θα πάρει από αυτή. Αν δεν είναι επιτρεπτή η κίνηση, επιστρέφει **null**. Η κλάση περιλαμβάνει επίσης την **print** η οποία τυπώνει τον πίνακα της τρέχουσας κατάστασης του παιχνιδιού, την **getChildren** η οποία επιστρέφει την λίστα με της καταστάσεις που προκύπτουν αν παιχτεί κάθε μία από τις δυνατές κινήσεις, μεθόδους **get** και **set**, την μέθοδο **isTerminal** η οποία ελέγχει αν έχει νικήσει κάποιος παίκτης και επιστρέφει τον νικητή (1 ή 2). Αν δεν έχει νικήσει κάποιος τότε επιστρέφει 0. Τέλος η λογική συνάρτηση **isDraw** επιστρέφει

true αν το παιχνίδι έχει λήξει με ισοπάλια. Δηλαδή καλεί την **isTerminal** και ελέγχει αν υπάρχουν η λίστα **possMoves** είναι άδεια.

Δημήτρης Γλυνάτσας 3140036

Κωνσταντίνος Καρακαξίδης 3140071

Λάμπρος Γαβαλάκης 3130031