ΟΙΚΟΝΟΜΙΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ
ATHENS UNIVERSITY OF ECONOMICS AND BUSINESS

ΣΧΟΛΗ ΕΠΙΣΤΗΜΩΝ & ΤΕΧΝΟΛΟΓΙΑΣ ΤΗΣ ΠΛΗΡΟΦΟΡΙΑΣ
SCHOOL OF INFORMATION SCIENCES & TECHNOLOGY
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ
DEPARTMENT OF INFORMATICS

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

ΤΕΧΝΗΤΗ ΝΟΗΜΟΣΥΝΗ
ΧΕΙΜΕΡΙΝΟ ΕΞΑΜΗΝΟ 2023-2024

Ελένη Κεχριώτη
Μαρία Σχοινάκη
Χρήστος Σταμούλος

OTHELLO-RIVERSI

Εισαγωγή

Η εργασία μας αφορά την υλοποίηση του παιχνιδιού Othello - Reversi, που είναι το δεύτερο θέμα της δοσμένης εκφώνησης. Η εφαρμογή μας επιτρέπει σε έναν χρήστη να παίζει Othello με αντίπαλο τον υπολογιστή, δίνοντας του τη δυνατότητα να διαλέξει την σειρά που θέλει να παίξει και τον βαθμό δυσκολίας του παιχνιδιού (1 έως 5).
Ο αλγόριθμος τεχνητής νοημοσύνης που χρησιμοποιήσαμε για την υλοποίηση του αντιπάλου AI είναι ο Minimax με πριόνισμα α-β.

Minimax με πριόνισμα a-b

Στο πρόγραμμα μας, όπως ειπώθηκε και στην εισαγωγή, εφαρμόζουμε τον αλγόριθμο minimax με πριόνισμα α-β. Ο αλγόριθμος αυτός βρίσκει εύκολα την βέλτιστη κίνηση που μπορεί να γίνει στο παιχνίδι, αφού είναι ένα παιχνίδι 2 αντιπάλων, με τον ένα μάλιστα να είναι ο υπολογιστής. Θέτοντας σε εφαρμογή τον minimax, δημιουργείται και για τους δύο παίκτες ένα δέντρο πιθανών μελλοντικών καταστάσεων με έναρξη την τρέχουσα κατάσταση του ταμπλό του παιχνιδιού. Κάθε κόμβος του δέντρου αναπαριστά μια κατάσταση του ταμπλό, στην οποία έχει παίξει είτε ο παίκτης με τα άσπρα πιόνια, είτε ο παίκτης με τα μαύρα πιόνια και συνοδεύεται από την αξία του, η οποία δίνεται μέσω της ευρετικής συνάρτησης evaluate.
Οι κόμβοι, όπως είπαμε, χωρίζονται σε καταστάσεις όπου είτε έχει παίξει τελευταίος ο παίκτης με τα μαύρα πιόνια, είτε ο παίκτης με τα άσπρα. Σύμφωνα και με την υλοποίηση της μεθόδου MiniMax (βλέπε παρακάτω), μπορούμε να πούμε ότι ο παίκτης με τα μαύρα πιόνια έχει τους max κόμβους και ο παίκτης με τα άσπρα πιόνια έχει τους min κόμβους. Επομένως, στόχος των μαύρων κόμβων είναι να μεγιστοποιήσουν την αξία τους. Για αυτό το λόγο, για να βρούμε τη επόμενη βέλτιστη κίνηση, για κάθε πιθανή κίνηση (παιδί της τρέχουσας κατάστασης του ταμπλό) επιλέγεται το παιδί της με την μεγαλύτερη αξία, η οποία γίνεται και η αξία της πιθανής κίνησης. Αντίστοιχα, και για τους άσπρους min κόμβους.
Όλη αυτή η διαδικασία, είναι χρονοβόρα για μεγάλα δέντρα αναζήτησης οπότε θέλουμε να περιορίσουμε την αναζήτηση πιθανών κινήσεων σε κλάδους που πιθανόν να επιλεχθούν. Αυτό επιτυγχάνεται με το πριόνισμα α-β.
Το πριόνισμα α-β έχει δύο όρια, το α και το β που περιορίζουν το σύνολο των πιθανών κινήσεων με βάση το δέντρο που έχει ήδη αναπτυχθεί. Το α είναι το μέγιστο κάτω όριο αξίας για τις πιθανές κινήσεις και το β είναι το ελάχιστο άνω όριο αξίας. Έτσι, αποφεύγεται η άσκοπη παραγωγή και αναζήτηση υπό-δέντρων που δεν θα επιλέγονταν.

MAIN

main

Η κύρια μέθοδος της εργασίας μας είναι η **Main(),** εκεί οι περισσότερες μέθοδοι του υπόλοιπου προγράμματός συνδέονται και παράγεται το τελικό αποτέλεσμα του **Othello-Reversi.** Ας εξετάσουμε την λειτουργία της βήμα προς βήμα.

Η ροή της μεθόδου μας εξελίσσεται με τον έλεγχο εισόδου του χρήστη. Για να επιτευχθεί αυτό χρησιμοποιούμε 2 δομές επανάληψης while οι οποίες εμφανίζουν κατάλληλο μήνυμα στον χρήστη εφόσον εκχωρήσει λανθασμένη είσοδο. Στην συνέχεια εκτυπώνουμε στην οθόνη την αρχική κατάσταση του παιχνιδιού.

Ακολουθεί η κεντρική δομή επανάληψης while στην οποία βασίζεται το μεγαλύτερο κομμάτι της **Main().** Όσο το παιχνίδι δεν βρίσκεται σε τελική κατάσταση (μέσω της isTerminal()) η διαμάχη μεταξύ του παίκτη και του AI συνεχίζεται! Εσωτερικά της επανάληψης while διακρίνουμε τις περιπτώσεις της switch /case.. Η switch χρησιμοποιεί την συνάρτηση **getLastPlayer()** κλάσεως **Board** και μέσω αυτής καθορίζεται ποιο χρώμα από τα πούλια έπαιξε τελευταία (**-1/B** ή **1/W**).

Στην περίπτωση που η τελευταία κίνηση έγινε από πιόνι μαύρου χρώματος και ο χρήστης έχει επιλέξει να παίξει δεύτερος τότε έφτασε η σειρά του! Έτσι δημιουργούμε μια νέα κίνηση χρήστη και εμφανίζουμε τις συντεταγμένες που αυτός επέλεξε.

Εναλλακτικά σε αντίθετη περίπτωση είναι η σειρά του συστήματος να κάνει κίνηση! Στο σημείο αυτό τίθεται σε λειτουργία ο αλγόριθμος **Minimax** και το σύστημα υπολογίζει την βέλτιστη κίνηση που του επιτρέπεται να κάνει την δεδομένη στιγμή. Εμφανίζει στην οθόνη τις συντεταγμένες που επιλέγει και εφαρμόζει τελικά την κίνηση του μέσω της **makeMove()** και των αντίστοιχων δεδομένων .

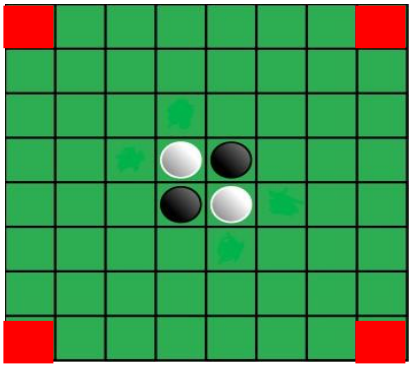
Πανομοιότυπη λογική εφαρμόζεται και στην υπόλοιπες περιπτώσεις χρωμάτων από πιόνια και σειράς παίκτη.

BOARD

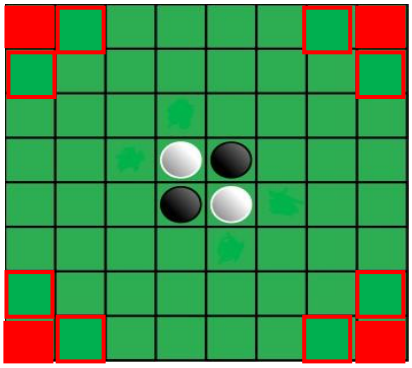
evaluate

Όπως και σε κάθε παιχνίδι 2 αντιπάλων, έτσι και εδώ πρέπει να εφαρμόσουμε την λογική του *minimax* αλγορίθμου. Για να μπορέσει ο αλγόριθμός μας να επιλέξει την βέλτιστη σύμφωνα με τα δεδομένα επόμενη κίνηση, θα πρέπει να δώσουμε αξία στις κινήσεις. Έτσι ώστε, από όλα τα πιθανά ταμπλό που έχουν παιχτεί οι πιθανές κινήσεις του υπολογιστή, ο αλγόριθμος να επιλέγει το καλύτερο. Η μέθοδος-συνάρτηση που δίνει αξία στις κινήσεις-ταμπλό, είναι η ευρετική(***int evaluate()***). Η μέθοδος αυτή, επιστρέφει έναν ακέραιο, που αντιπροσωπεύει την αξία του ταμπλό μετά από κάποια πιθανή κίνηση.

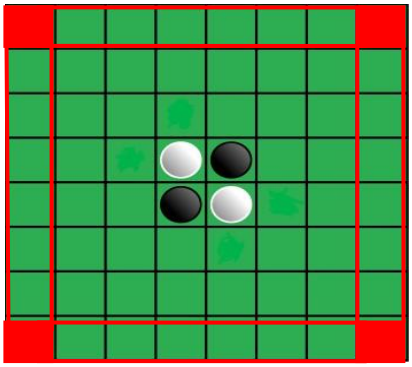
Παρατηρήσαμε ότι όποιος παίχτης καταλάβει μία εκ των 4 γωνιών του ταμπλό, μετά το πιόνι αυτό δεν μπορεί να το αναποδογυρίσει ο αντίπαλος. Είναι σταθερό για όσο ζει το παιχνίδι. Οπότε θεωρήσαμε τις κινήσεις που πιάνουν τις 4 γωνίες, ως τις πιο πολύτιμες, καθώς δεν δίνουν απλά πλεονέκτημα, δίνουν και την σιγουριά ότι αυτό το πιόνι δεν θα αναποδογυρίσει ποτέ.



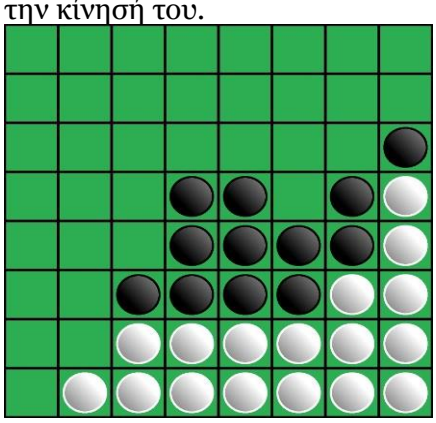
Με βάση αυτήν την παρατήρηση, δημιουργήθηκε και μία ακόμη. Αν κατέχεις μια εκ των 4 γωνιών, και καταλάβεις την θέση αριστερά, δεξιά, πάνω ή κάτω της, δημιουργείς μια αλυσίδα πιονιών, τα οποία, ο αντίπαλος δεν μπορεί να σου αναποδογυρίσει. Έτσι, σίγουρα το πλήθος με τα μόνιμα(δεν μπορεί να αναποδογυρίσει ο αντίπαλος), πιόνια σου, θα μεγαλώνει με εκθετικό ρυθμό.



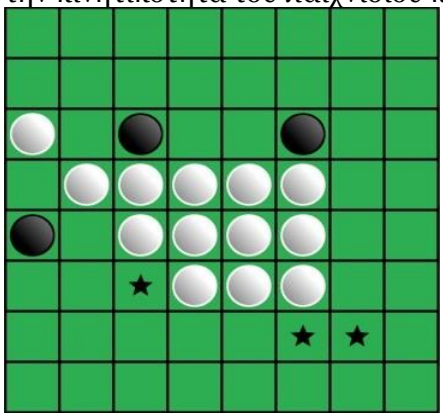
Αμέσως μετά, παρατηρήσαμε ότι σε πλεονεκτική θέση, βρίσκονται και τα πιόνια που βρίσκονται στα άκρα του πίνακα(ακριανές γραμμές-στήλες). Ο λόγος είναι ότι ο αντίπαλος μπορεί να σε αναποδογυρίσει μόνο οριζόντια(όχι διαγώνια ή κάθετα) στις ακριανές γραμμές, ενώ μόνο κάθετα(όχι οριζόντια ή διαγώνια) στις ακριανές στήλες. Επίσης, κατέχοντας τις άκρες, μπορείς εύκολα να κατευθύνεις το παιχνίδι, αναποδογυρίζοντας μεγάλο αριθμό πιονιών του αντιπάλου.



Μία ακόμη παρατήρηση, που είναι και αρκετά σημαντική, είναι η κίνηση που μπλοκάρει τον αντίπαλο. Με το να παίξεις κίνηση που αφαιρεί από τον αντίπαλο το δικαίωμα να παίξει, δεν κερδίζεις απλά μία περισσότερη κίνηση(συνεχόμενη), αλλά ελέγχεις και το παιχνίδι. Ο αντίπαλος πλέον, παίζει με τα δικά σου δεδομένα και ακόμα καλύτερα είναι τόσο λίγες οι πιθανές κινήσεις που ίσως έχει μετά, που μπορείς εύκολα να προβλέψεις την κίνησή του.



Μία παρόμοια, λιγότερο σημαντική παρατήρηση είναι αυτή της μέτρησης των πιθανών κινήσεων του αντιπάλου. Όπως προαναφέραμε, το Othello είναι καθαρά παιχνίδι στρατηγικής. Αν η κίνηση που παίζεις αφήνει ελάχιστες πιθανές κινήσεις για τον αντίπαλο, τότε γίνεσαι ο κυρίαρχος του παιχνιδιού. Ελέγχεις κάθε κίνηση, ακόμη και ολόκληρο το ταμπλό, και είσαι σίγουρα σε πλεονεκτική θέση. Ο αντίπαλος, πλέον δεν παίρνει αποφάσεις, απλά εκτελεί διαταγές που εσύ εμμέσως έχεις δώσει. Κατευθύνεις την κινητικότητα του παιχνιδιού και αδιαμφισβήτητα τις κινήσεις του αντιπάλου.



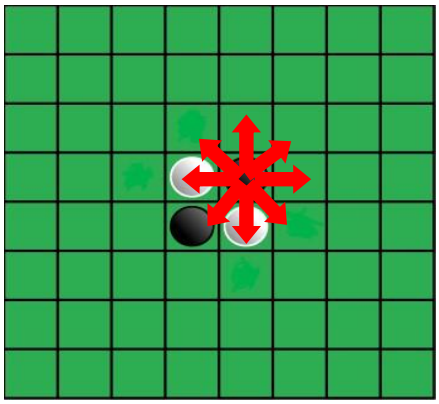
Κάναμε αρκετές παρατηρήσεις σχετικά και με άλλα κομμάτια του παιχνιδιού, όπως οι δεύτερες γωνίες, τα κεντρικά σημεία κτλπ. Όμως, θεωρήσαμε ότι δεν υπάρχει κάποιος γενικευμένος κανόνας που δίνει πλεονέκτημα σε αυτές τις θέσεις, παρά μόνο το ίδιο το ταμπλό ολόκληρο. Το οποίο θα προσέφερε μια κακή, χωρίς λόγο πολυπλοκότητα. Έτσι, κάθε άλλη θέσει προσφέρει μικρή αξία στην ευρετική μας.

Γωνίες	**1000**
Άκρα δίπλα σε κατεχόμενες γωνίες	**800**
Μπλοκάρισμα αντιπάλου	**700**
Άκρα	**500**
Πιθανές κινήσεις αντιπάλου	**100**
Υπόλοιπες θέσεις	**50**

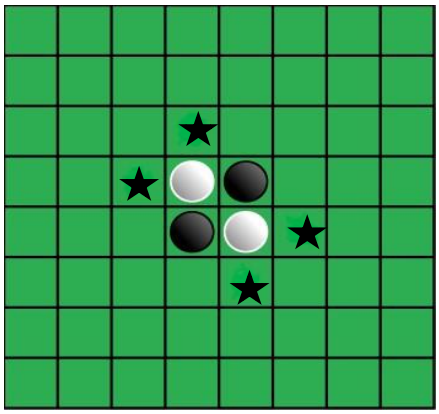
isValid

Μία άλλη, ζωτικής σημασίας μέθοδος-συνάρτηση είναι η ***bool isValid()***. Η μέθοδος αυτή, μας επιτρέπει να γνωρίζουμε αν μία κίνηση είναι επιτρεπτή. Επιτρεπτή κίνηση, είναι μία κίνηση όπου το πιόνι x τοποθετείται σε μια θέση που να μπορεί να αναποδογυρίσει 1 ή περισσότερα πιόνια του αντιπάλου. Προφανώς, για να επιτευχθεί αυτό, πρέπει το πιόνι x να περικλείει τα προς αναστροφή πιόνια του αντιπάλου, με την βοήθεια ενός ακόμη πιονιού, της ίδιας ομάδας(χρώματος), με το x. Προφανές αλλά αξιοσημείωτο επίσης, είναι ότι αυτή η συνάρτηση, ελέγχει και αν η θέση που πάει να τοποθετηθεί το πιόνι, είναι διαθέσιμη(ελεύθερη, κενή). Η μέθοδος αυτή, επίσης τσεκάρει αν η θέση που τείνει να τοποθετηθεί το πιόνι βρίσκεται εντός ορίων του πίνακα-ταμπλό(σε 8x8 ταμπλό, 0<=γραμμή<=7, 0<=στήλη<=7). Για να τσεκάρει αν πραγματοποιείται η αναστροφή πιονιών του αντιπάλου, η μέθοδος τσεκάρει τις 8 κατευθύνσεις(αν είναι διαθέσιμες και δεν ξεπερνάνε τα όρια του πίνακα-ταμπλό).

Για παράδειγμα, στο ταμπλό παρακάτω, η μέθοδος θα τσεκάρει όλες τις 8 κατευθύνσεις ***μέχρι***(break clause) να βρει μία, η οποία ικανοποιεί, τα όρια, την κενότητα και την αναστροφή πιονιών του αντιπάλου. Έτσι, το μαύρο πιόνι, έχει 4 πιθανές κινήσεις που ικανοποιούν την μέθοδο ***isValid()***.



Οι 4 πιθανές κινήσεις, φαίνονται παρακάτω.



getChildren

Επιστρέφει μια λίστα με τα παιδιά (αντικείμενα Board). Ένα παιδί είναι η τρέχουσα κατάσταση του πίνακα και η επόμενη πιθανή κίνηση που μπορεί να κάνει ο παίκτης με βάση το τρέχων πίνακα. Η εύρεση των παιδιών γίνεται διατρέχοντας τον πίνακα μας και βρίσκοντας όλες τις πιθανές κινήσεις που μπορεί να γίνουν από το ζητούμενο παίκτη. Για κάθε μία από αυτές τις κινήσεις, δημιουργούμε το "παιδί" ως ένα αντίγραφο του πίνακα στον οποίο γίνεται ύστερα η κίνηση αυτή.

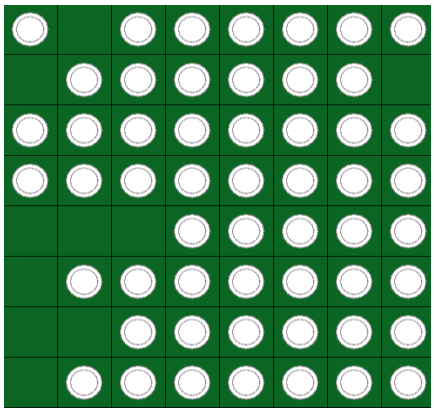
inBoarders

Ελέγχει αν η κίνηση είναι εντός του πίνακα και επιστρέφει True. Διαφορετικά, επιστρέφει False.

isTerminal

Διατρέχει τον δισδιάστατο μας πίνακα μετρώντας τα μαύρα πιόνια, τα άσπρα πιόνια και τις κενές θέσεις. Αν δεν υπάρχουν άλλες κενές θέσεις στον πίνακα ή ένας από τους δύο παίκτες δεν έχει άλλα πιόνια στο ταμπλό, τότε το παιχνίδι έχει τελειώσει και επιστρέφουμε True. Διαφορετικά, επιστρέφουμε False.

Για παράδειγμα, ένα τερματισμένο ταμπλό(παιχνίδι) φαίνεται παρακάτω:



Όπου, ο παίκτης με τα μαύρα πιόνια, δεν έχει άλλα πιόνια στο ταμπλό, άρα δεν μπορεί να παίξει κίνηση, και το ίδιο ο παίκτης με τα άσπρα πιόνια, καθώς δεν υπάρχει κάποιο μαύρο πιόνι να περικλείσει.

isBlocked

Διατρέχει τον δισδιάστατο μας πίνακα και ελέγχει αν υπάρχουν έγκυρες κινήσεις για τον παίκτη. Αν υπάρχει έστω μια έγκυρη κίνηση, τότε ο παίκτης δεν είναι μπλοκαρισμένος και επιστρέφουμε True. Διαφορετικά, επιστρέφουμε False.

makeMove

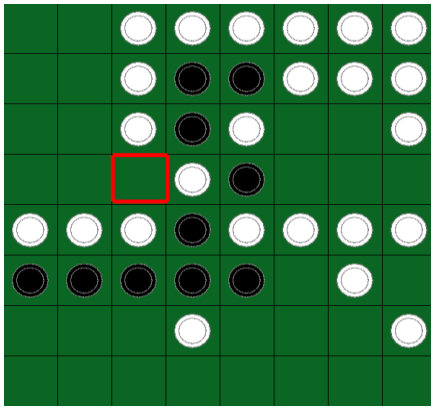
Ελέγχει εάν η κίνηση που θέλει να κάνει ο παίκτης είναι έγκυρη, και μόνο τότε επιτρέπει να γίνει η κίνηση, θέτοντας στο ταμπλό το πιόνι του παίκτη και καλώντας τη μέθοδο flipcheckers για να γυρίσουν τα πιόνια του αντιπάλου. Διαφορετικά, εμφανίζει μήνυμα μη έγκυρης κίνησης. Επίσης, δημιουργεί μια νέα κίνηση, με τις συντεταγμένες της

κίνησης που μόλις έγινε, την οποία θέτει ως τελευταία κίνηση και τον παίκτη που μόλις έπαιξε ως τελευταίο παίκτη.

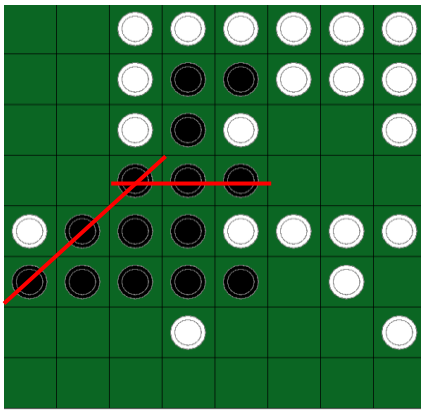
flipcheckers

Δεδομένου ότι μόλις έχει γίνει μια έγκυρη κίνηση, γυρνάει τα πιόνια που πρέπει να γυριστούν, δηλαδή αυτά που περικλείονται από τα πιόνια του αντιπάλου με αυτή τη κίνηση. Για κάθε κατεύθυνση (πάνω, κάτω, δεξιά, αριστερά, διαγώνια πάνω δεξιά, διαγώνια κάτω δεξιά, διαγώνια κάτω αριστερά, διαγώνια πάνω αριστερά) από το σημείο που κάνει την κίνηση του ελέγχει αν υπάρχουν πιόνια του αντιπάλου και σταματάει μόλις βρεθεί πιόνι του παίκτη. Αν δεν βρεθεί, τότε δεν εγκλωβίζονται προς αυτή τη κατεύθυνση πιόνια του αντίπαλου και άρα δεν γυρνάνε τα πιόνια του. Αντίθετα αν βρεθεί πιόνι του παίκτη, τότε γυρνάμε τα πιόνια του αντιπάλου.

Μία προσομοίωση της μεθόδου, φαίνεται παρακάτω.



Αν ο παίκτης με τα μαύρα πιόνια, επιλέξει, το επιλεγμένο κουτί, τότε παρατηρούμε ότι συμπληρώνει μία διαγώνια τριάδα, και μία οριζόντια, όπου περικλείει ένα πιόνι του αντιπάλου σε κάθε μια κατεύθυνση. Άρα το ταμπλό μετά την κίνηση του μαύρου στην επιλεγμένη θέση και σύμφωνα και με την μέθοδο flipcheckers, θα γίνει:



getScores

Επιστρέφει σύμφωνα με το δεδομένο ταμπλό, πόσα πιόνια έχει κάθε παίχτης.

setLastMove

Καθορίζει την τελευταία κίνηση.

getLastMove

Επιστρέφει την τελευταία κίνηση που έγινε.

setLastPlayer

Θέτει τον τελευταίο παίκτη που έπαιξε.

getLastPlayer

Επιστρέφει τον τελευταίο παίκτη που έπαιξε.

setGameBoard

Καθορίζει τον δισδιάστατο πίνακα που αναπαριστά το ταμπλό μας.

getGameBoard

Επιστρέφει τον πίνακα του παιχνιδιού μας

MOVE

Constructors

Η Κλάση **Move** του προγράμματος μας είναι υπεύθυνη για την προσομοίωση των κινήσεων τις οποίες θα πραγματοποιεί είτε ο χρήστης είτε ο υπολογιστής πάνω στο ταμπλό του παιχνιδιού.(Η υλοποίηση των κινήσεων γίνεται μέσω της **Makemove()** κλάσεως **Board**).
Το αρχείο Move απαρτίζεται από τα απαραίτητα πεδία συντεταγμένων (γραμμής - στήλης) καθώς και μιας τιμής **value** για τον καθορισμό χρώματος από τα πιόνια. Τα παραπάνω δεδομένα χρησιμοποιούνται από 4 κατασκευαστές κίνησης καθώς και συναρτήσεις εξαγωγής/ανάθεσης τιμών(setters/getters) .

Συνδυάζοντας τις δυνατότητες που παρέχει η **Move** οποιοδήποτε ανανέωση στο ταμπλό πραγματοποιείται με τον ελάχιστο προγραμματιστικό κόπο.

PLAYER

setMaxDepth

Καθορίζει τον αριθμό δυσκολίας του παιχνιδιού.

setPlayerLetter

Καθορίζει το χρώμα/γράμμα B ή W για τον παίκτη.

MiniMax

Καλεί την max, αν ο παίκτης παίζει με τα μαύρα πιόνια, γιατί θέλουμε να μεγιστοποιήσουμε την αξία του, αλλιώς για τα άσπρα πιόνια καλεί την min.

min

Ψάχνει να βρει την κίνηση με την ελάχιστη αξία (μέσω ευρετικής) από τις πιθανές κινήσεις που μπορεί να γίνουν.

max

Ψάχνει να βρει την κίνηση με την μέγιστη αξία (μέσω ευρετικής) από τις πιθανές κινήσεις που μπορεί να γίνουν.
Οι min και max καλούνται μεταξύ τους εναλλάξ για το επόμενο επίπεδο του δέντρου πιθανών κινήσεων.

WINDOW

Η κλάση στην οποία υλοποιείται η γραφική διεπαφή του παιχνιδιού **Othello**.

Window

Δημιουργεί ένα παράθυρο (frame) στο οποίο θα διεξαχθεί το παιχνίδι. Στη κλάση αυτή δημιουργούνται και μορφοποιούνται όλα τα panels και τα κουμπιά που θα χρησιμοποιηθούν για τη διεξαγωγή του παιχνιδιού. Τα panel είναι τρία και χωρίζονται στη:
* αρχική οθόνη, όπου ο χρήστης προσαρμόζει το παιχνίδι όπως του αρέσει και το ξεκινάει
* κύρια οθόνη, όπου ο χρήστης παίζει ενάντια στον υπολογιστή
* τελική οθόνη, όπου φαίνεται το τελικό σκορ και ο νικητής του γύρου. Υπάρχει επιλογή για τερματισμό του παιχνιδιού (έξοδος) και αρχή νέου γύρου.

MousePressed

Χρησιμοποιείται κυρίως για την κύρια οθόνη. Για κάθε πάτημα που κάνει ο χρήστης στην οθόνη, παίρνουμε τις συντεταγμένες του κέρσορα στην οθόνη και βρίσκουμε σε ποιο κελί αντιστοιχεί στο ταμπλό. Αυτό γίνεται κρατώντας το αποτέλεσμα της ακέραιας διαίρεσης της τεταγμένης/τετμημένης με το 60 (ο αριθμός των pixels της διάστασης κάθε κελιού). Μετά από αυτό το σημείο ακολουθείται η ίδια λογική με την main:Main για την διεξαγωγή των κινήσεων.

Επίλογος

Η εργασία αυτή έγινε στα πλαίσια του μαθήματος **«Τεχνητή Νοημοσύνη»**, το οποίο αποτελεί υποχρεωτικό μάθημα πυρήνα του 3ου έτους του τμήματος της Πληροφορικής. Το pdf, έχει γραφεί αποκλειστικά από τους φοιτητές της ομάδας, καθώς και όλος ο κώδικας που έχει υλοποιηθεί. Αρωγός σε αυτήν την προσπάθεια αποτέλεσαν οι σημειώσεις του μαθήματος, καθώς και των φροντιστηρίων. Χρήσιμες πληροφορίες επίσης πάρθηκαν από τα βιβλία των **S.Russell** και **P.Norvig «Τεχνητή Νοημοσύνη, μία σύγχρονη προσέγγιση»** 4η αμερικανική έκδοση και **Ι.Βλαχάβα, Π.Κεφαλά, Ν.Βασιλειάδη, Φ.Κόκκορα και Η.Σκελλαρίου «Τεχνητή Νοημοσύνη»** Δ έκδοση.