ΝΤΥΜΕΝΟΣ ΠΑΝΑΓΙΩΤΗΣ 3160120
ΠΑΤΡΙΚΗΣ ΓΕΩΡΓΙΟΣ 3160141
ΦΩΤΟΠΟΥΛΟΣ ΓΕΩΡΓΙΟΣ 3130220

19/11/18

ΤΕΧΝΗΤΗ ΝΟΗΜΟΣΥΝΗ – ΠΡΩΤΗ ΠΡΟΓΡΑΜΜΑΤΙΣΤΙΚΗ ΕΡΓΑΣΙΑ

Board

public Board(): Default κατασκευαστής, φτιάχνει ένα πίνακα 8x8 θέσεων, γεμίζει με κενά την κάθε θέση, ενώ στις μεσαίες βάζει τιμές 'X' 'O' εναλλάξ.
public void print(Board x): Εκτυπώνει τον πίνακα που δίνεται σαν όρισμα.
public void printWithMap(Board x): Εκτυπώνει τον πίνακα των δυνατών κινήσεων.

GameEngine

public static void main(String[] args): Μέθοδος που περιλαμβάνει όλη τη λειτουργικότητα της άσκησης, καλώντας τις μεθόδους ώστε να παίξει ο παίκτης αλλά και το AI, κάνει έλεγχο ότι το παιχνίδι μπορεί να συνεχίσει να παίζεται όσο έχει κίνηση έστω ο ένας από τους δύο και στο τέλος του παιχνιδιού υπολογίζει το τελικό σκορ αποφασίζοντας τον νικητή.
public static boolean isInteger(String input): Μέθοδος που επιστρέφει true αν το string που δίνεται σαν όρισμα είναι integer, διαφορετικά επιστρέφει false.
public static void playersTurn(char choice): Η μέθοδος αυτή εμφανίζει κάθε φορά την επιλογή (O ή X) του παίκτη σε ένα μήνυμα, ύστερα ζητάει input για τις τιμές των row, column όπου ο παίκτης θέλει να τοποθετήσει ένα δισκίο, ελέγχει την εγκυρότητα των τιμών που έδωσε ο χρήστης, ζητώντας του να δώσει εκ νέου τιμές αν αυτές δεν είναι έγκυρες ή δεν υπάρχει κίνηση για την δοσμένη θέση. Τελικά, τοποθετεί το δισκίο στις θέσεις που έδωσε ο χρήστης στον πίνακα που παίζεται το παιχνίδι.
public static void AITurn(char choice, char revChoice): Η μέθοδος αυτή εμφανίζει κάθε φορά την επιλογή (O ή X) του AI σε ένα μήνυμα, ύστερα, αν το AI έχει δυνατή κίνηση, καλούμε τη μέθοδο outcomeminimax που ορίζουμε στην κλάση Moves και επεξηγούμε παρακάτω, μέθοδος που περιλαμβάνει τη κλήση της συνάρτησης minimax που περιλαμβάνει τον αλγόριθμο minimax & a-b pruning που έχουμε χρησιμοποιήσει.
public static void calculateScore(Board B, char choice): Υπολογίζει το τελικό σκορ, τυπώνοντας τα κατάλληλα μηνύματα και τυπώνει τον τελικό πίνακα του παιχνιδιού όπως αυτός διαμορφώθηκε με την τελευταία κίνηση.

Moves

public int[][] Values = new int[][]: Αρχικοποιούμε ένα πίνακα 8x8, ο οποίος περιέχει το βάρος που έχει η κάθε θέση στον πίνακα, τον λαμβάνουμε υπόψιν αργότερα όταν κάνουμε το evaluate που έχει η κάθε θέση στο minimax.
public int numOfmoves(Board B, char choice): Μέθοδος που υπολογίζει τον αριθμό των διαθέσιμων επόμενων κινήσεων.
public boolean possibleMoves(Board B, char OX): Η μέθοδος αυτή επιστρέφει true αν υπάρχει έστω και μια διαθέσιμη κίνηση ελέγχοντας σε κάθε κατεύθυνση στον πίνακα, αν δε βρεθεί κίνηση επιστρέφει false.
public void SouthEast(Board B, int i, int j, char OX, char revOX): Μέθοδος που ελέγχει αν υπάρχει κίνηση στη κάτω δεξιά θέση, δηλαδή αν μεταξύ μιας κενής θέσης και της θέσης εκκίνησης παρεμβάλλεται τουλάχιστον ένα αντίπαλο δισκίο.

ΝΤΥΜΕΝΟΣ ΠΑΝΑΓΙΩΤΗΣ 3160120
ΠΑΤΡΙΚΗΣ ΓΕΩΡΓΙΟΣ 3160141
ΦΩΤΟΠΟΥΛΟΣ ΓΕΩΡΓΙΟΣ 3130220

19/11/18

ΤΕΧΝΗΤΗ ΝΟΗΜΟΣΥΝΗ – ΠΡΩΤΗ ΠΡΟΓΡΑΜΜΑΤΙΣΤΙΚΗ ΕΡΓΑΣΙΑ

public void South(Board B, int i, int j, char OX, char revOX): Μέθοδος που ελέγχει αν υπάρχει κίνηση στη κάτω θέση, δηλαδή αν μεταξύ μιας κενής θέσης και της θέσης εκκίνησης παρεμβάλλεται τουλάχιστον ένα αντίπαλο δισκίο.
public void SouthWest(Board B, int i, int j, char OX, char revOX): Μέθοδος που ελέγχει αν υπάρχει κίνηση στη κάτω αριστερή θέση, δηλαδή αν μεταξύ μιας κενής θέσης και της θέσης εκκίνησης παρεμβάλλεται τουλάχιστον ένα αντίπαλο δισκίο.
public void East(Board B, int i, int j, char OX, char revOX): Μέθοδος που ελέγχει αν υπάρχει κίνηση στη δεξιά θέση, δηλαδή αν μεταξύ μιας κενής θέσης και της θέσης εκκίνησης παρεμβάλλεται τουλάχιστον ένα αντίπαλο δισκίο.
public void West(Board B, int i, int j, char OX, char revOX): Μέθοδος που ελέγχει αν υπάρχει κίνηση στη αριστερή θέση, δηλαδή αν μεταξύ μιας κενής θέσης και της θέσης εκκίνησης παρεμβάλλεται τουλάχιστον ένα αντίπαλο δισκίο.
public void NorthEast(Board B, int i, int j, char OX, char revOX): Μέθοδος που ελέγχει αν υπάρχει κίνηση στη πάνω δεξιά θέση, δηλαδή αν μεταξύ μιας κενής θέσης και της θέσης εκκίνησης παρεμβάλλεται τουλάχιστον ένα αντίπαλο δισκίο.
public void North(Board B, int i, int j, char OX, char revOX): Μέθοδος που ελέγχει αν υπάρχει κίνηση στη πάνω θέση, δηλαδή αν μεταξύ μιας κενής θέσης και της θέσης εκκίνησης παρεμβάλλεται τουλάχιστον ένα αντίπαλο δισκίο.
public void NorthWest(Board B, int i, int j, char OX, char revOX): Μέθοδος που ελέγχει αν υπάρχει κίνηση στη πάνω αριστερή θέση, δηλαδή αν μεταξύ μιας κενής θέσης και της θέσης εκκίνησης παρεμβάλλεται τουλάχιστον ένα αντίπαλο δισκίο.
public Board outcomeminimax(Board B, int depth, int alpha, int beta, char choice, char revchoice, boolean maximizingPlayer): Μέθοδος που καλεί την minimax. Κρατάει κάποιες πληροφορίες αποθηκευμένες, οι οποίες μπορεί στη συνέχεια να αλλάξουν μες τη minimax. Επιστρέφει τον πίνακα που προκύπτει μετά την κίνηση του AI.
public int minimax(Board B, int depth, int alpha, int beta, char choice, char revchoice, boolean maximizingPlayer): Πρόκειται για μια εφαρμογή του αλγορίθμου που διδαχθήκαμε με μικρή παραλλαγή σχετικά με το βάθος του δέντρου. Θεωρούμε ότι ξεκινάμε με τον πατέρα κόμβο ο οποίος έχει το μέγιστο βάθος, κατεβαίνουμε προς τα παιδιά μέχρι να φτάσουμε στο επίπεδο 0 όπου και γίνεται η αξιολόγηση μέσω της κλήσης της evaluation. Σε κάθε επίπεδο ελέγχουμε αν έχουμε maximizing ή minimizing παίκτη, αν έχουμε maximizing ορίζουμε το maxEval ως -100.000, διαφορετικά ορίζουμε το minEval ως 100.000. Σε κάθε περίπτωση δημιουργούμε τα παιδιά μέσω της possibleMoves για την εκάστοτε επιλογή. Αν δεν υπάρχουν παιδιά ελέγχουμε αν ο άλλος παίκτης έχει κίνηση. Αν δεν έχει, αξιολογούμε την κατάσταση ως τερματική μέσω της GameOver, διαφορετικά αν ο ένας παίκτης δε μπορεί να παίξει αλλά μπορεί ο αντίπαλός του, ξανακαλούμε την minimax για τον άλλο παίκτη περνώντας τον ίδιο πίνακα σαν όρισμα. Εφόσον η possibleMoves δημιουργήσει τουλάχιστον ένα παιδί, η minimax τρέχει επαναληπτικά για όλο τον αριθμό των παιδιών και εφόσον βρίσκεται στο μέγιστο βάθος και είναι το πρώτο παιδί που καλείται, αποθηκεύουμε τις συντεταγμένες του παιδιού. Στη συνέχεια, για το κάθε παιδί καλούμε αναδρομικά τη minimax μειώνοντας το βάθος κατά ένα περνώντας τα alpha beta και σχολιάζοντας αν είναι max ή min κόμβος. Ελέγχουμε αν το τελικό αποτέλεσμα που προκύπτει από τον κάθε κόμβο του depth 0 ή της GameOver είναι μεγαλύτερο του maxEval ή του alpha και όπου είναι κάνουμε τις αντίστοιχες αντιμεταθέσεις. Αν αντιμεταθέσουμε το alpha με το eval και βρισκόμαστε σε depth = maxDepth σημαίνει πως έχουμε μια πλέον καλύτερη κίνηση στο πρώτο επίπεδο του δέντρου, οπότε κρατάμε τις συντεταγμένες αυτές. Αν το beta είναι μικρότερο ή ίσο του alpha, χρησιμοποιούμε ένα boolean breakflag (τεχνητό break), ώστε να βγούμε από τα εμφωλευμένα for. Με τον ίδιο ακριβώς τρόπο λειτουργούμε και στα min επίπεδα. Η minimax επιστρέφει μια int τιμή όμως με την κλήση της και ελέγχους στο βάθος maxDepth κάνουμε αποθηκεύσεις τιμών στο finali, finalj, μεταβλητές που μας δείχνουν από ποιο παιδί του αρχικού κόμβου προήλθε το αποτέλεσμα της minimax.

ΝΤΥΜΕΝΟΣ ΠΑΝΑΓΙΩΤΗΣ 3160120
ΠΑΤΡΙΚΗΣ ΓΕΩΡΓΙΟΣ 3160141
ΦΩΤΟΠΟΥΛΟΣ ΓΕΩΡΓΙΟΣ 3130220

19/11/18

ΤΕΧΝΗΤΗ ΝΟΗΜΟΣΥΝΗ – ΠΡΩΤΗ ΠΡΟΓΡΑΜΜΑΤΙΣΤΙΚΗ ΕΡΓΑΣΙΑ

private int GameOver(Board B, char choice, char revchoice): Μέθοδος που αξιολογεί τις κινήσεις ως καλές ή κακές αναλόγως με το αποτέλεσμα του παιχνιδιού.

private int evaluation(Board B, char choice, char revchoice): Η μέθοδος αξιολόγησης της εργασίας μας στηρίζεται σε τέσσερις ευρετικές συναρτήσεις, οι οποίες είναι οι ακόλουθες:

1. Γωνίες: Δίνουμε στις γωνίες μεγάλη αξία διότι βοηθούν να κερδηθεί το παιχνίδι.
2. Στενότητα Γωνιών: Είναι γενικά καλό να αναγκάσουμε τον αντίπαλο να πάρει θέση στις γειτονικές της γωνίας θέσεις, έτσι ώστε ακόμα κι αν δεν έχουμε δισκίο στη γωνία, με πάτημα την κίνηση αυτή του αντιπάλου, να μπορέσουμε να τοποθετήσουμε αργότερα.
3. Βάρος Θέσης: Χρησιμοποιώντας τον πίνακα που αρχικοποιήσαμε νωρίτερα, μπορούμε να κρίνουμε αν μια θέση έχει καλό βάρος ή όχι σε σύγκριση με άλλες διαθέσιμες θέσεις.
4. Συνολικός αριθμός δισκίων για τον κάθε παίκτη.

Η δική μας ευρετική: Συνδυάζει όλα τα παραπάνω κριτήρια, λαμβάνοντας υπόψιν και την θέση του αντιπάλου σε κάθε ένα από αυτά τα κριτήρια. Παρόλα αυτά δεν έχουν όλα τα κριτήρια την ίδια σημασία, λαμβάνονται υπόψιν σε διαφορετικό βαθμό. (Γωνίες πολύ σημαντικές, Στενότητα σημαντική, Βάρος σημαντικό, Συνολικός αριθμός δισκίων πρακτικά σχεδόν αδιάφορος).

Επεξηγήσεις για τη λειτουργικότητα των συναρτήσεων μπορούν να βρεθούν και στα σχόλια στα .java αρχεία!