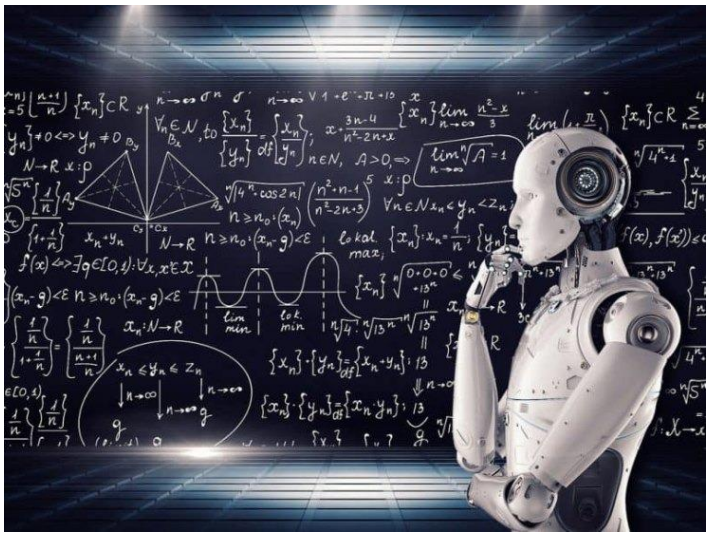
ΟΙΚΟΝΟΜΙΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ

ATHENS UNIVERSITY OF ECONOMICS AND BUSINESS

Τεχνητή Νοημοσύνη

Ακαδ. Έτος 2022-2023, Χειμερινό Εξάμηνο

1η ομαδική Εργασία



Όνομα	Αριθμός Μητρώου	E- Mail
Άννα Χατζηπαπαδοπούλου	3200219	p3200219@aueb.gr
Νεφέλη Ελένη Πολιού	3200243	p3200243@aueb.gr
Μιχάλης Χασάπης	3200218	p3200218@aueb.gr

Αναφορά Παράδοσης

Η συγκεκριμένη εργασία κάνει αναφορά σε 2 (δύο) δημοφιλή παιχνίδια. Ο κύβος του Rubik και το Reversi (παραλλαγή του Othello) είναι τα προβλήματα που καλείται κάθε ομάδα να επιλέξει και να λύσει. Η παρούσα ομάδα επέλεξε το δεύτερο. Ένα στρατηγικό παιχνίδι με σκοπό την επικράτηση του ενός χρώματος πούλιου έναντι του άλλου. Νικητής είναι εκείνος που έχει τα περισσότερα πούλια.

Η ανάπτυξη του προγράμματος έγινε με την βοήθεια της γλώσσας Java. Για ευκολία και ευανάγνωση του κώδικα, χωρίζεται σε 3 (τρεις) διαφορετικές κλάσεις, οι οποίες είναι οι εξής:

- **Move.java**, στην οποία υπάρχουν μεταβλητές που αφορούν στην δημιουργία του πίνακα και την εκτέλεση τον αλγόριθμό MiniMax. Οι δυο (2) ακέραιες μεταβλητές (row, col), με την βοήθεια των μεθόδων getRow και getCol επιστρέφουν τις γραμμές και τις στήλες που θα αποτελείται ο πίνακας του παιχνιδιού. Η μεταβλητή value αρχικοποιείται στον κατασκευαστή. Μέσω της μεθόδου μπορούμε να θέσουμε την τιμή της και με την μέθοδο είναι δυνατόν να πάρουμε την τιμή αυτή.

- **Reversi.java**, το αρχείο περιέχει την κλάση Reversi η οποία αποτελεί το πρόγραμμα εκτέλεσης του παιχνιδιού. Αρχικά, σε αυτή την κλάση καλείται η μέθοδος main στην οποία δημιουργούμε ένα αντικείμενο της κλάσης scanner ώστε να μπορούμε να δεχόμαστε δεδομένα αριθμών από τον χρήστη. Έπειτα δημιουργούμε μία οντότητα της κλάσης board η οποία θα λειτουργήσει ως το ταμπλό του παιχνιδιού (εκεί καλείται και η MiniMax και ελέγχονται οι κανόνες του παιχνιδιού). Καλούμε δυο (2) φορές διαδοχικά την scanner. Πρωτίστως για να μας δοθεί το βάθος αναζήτησης του MiniMax και στην συνέχεια για να αποφασιστεί εάν θα παίξει πρώτος είτε ο παίκτης είτε ο υπολογιστής. Αρχικοποιούμε τις μεταβλητές σειρά (row) και στήλη (col) ώστε να μπορέσουμε να ξεκινήσουμε με την επιλογή συντεταγμένων. Επιπλέον δημιουργούμε ένα array list, το οποίο περιλαμβάνει όλες τις πιθανές κινήσεις του παίκτη. Παράλληλα, γίνεται έλεγχος μέσω ενός επαναληπτικού κόμβου για την ολοκλήρωση του παιχνιδιού. Όσο δεν έχουμε φτάσει σε τελική κατάσταση εισάγουμε στο array list τις κινήσεις που μπορεί να επιλέξει ο παίκτης όταν έρθει η δική του σειρά. Σε περίπτωση που ξεκινήσει ο υπολογιστής την αδειάζουμε αλλιώς τυπώνουμε το ταμπλό. Αν το array list είναι άδειο εμφανίζουμε μήνυμα πως δεν υπάρχουν δυνατές κινήσεις, διαφορετικά καλούμε την scanner και σώζουμε στις μεταβλητές συντεταγμένων τις συντεταγμένες που έδωσε ο παίκτης (τις μειώνουμε κατά 1 ώστε να είναι σε αντιστοιχία με τους δείκτες του

πίνακα) και αυξάνουμε τον δείκτη σειράς. Εάν το array list των κινήσεων του παίκτη δεν περιέχει διαθέσιμες κινήσεις, αλλάζουμε τον παίκτη στο ταμπλό να είναι ο υπολογιστής και μέσω του αλγόριθμου MiniMax βρίσκουμε την καλύτερη δυνατή κίνηση για τον υπολογιστή. Αν υπάρχει κάποια κίνηση την εφαρμόζουμε αλλιώς αλλάζουμε τον παίκτη του ταμπλό να είναι ο χρήστης. Τέλος δημιουργούμε δύο μεταβλητές που αποθηκεύουν το τελικό σκορ. Μέσα από μία διάσχιση του ταμπλό προσθέτουμε όταν βρίσκουμε 1 (X) στο σκορ του παίκτη αλλιώς αν είναι -1 (O) στο σκορ του υπολογιστή. Ελέγχουμε ποιο σκορ είναι μεγαλύτερο και εμφανίζουμε αντίστοιχο μήνυμα για τον νικητή και με τι σκορ κέρδισε. Υπάρχει και η πιθανότητα της ισοπαλίας.

- Τελευταίο αρχείο για την ολοκλήρωση του παιχνιδιού Reversi είναι το **Board.java**, ο πίνακας περιλαμβάνει στοιχεία τύπου int όπου 1 συμβολίζει τα X, 2 τα O και 0 την κενή (ελεύθερη θέση). Η μεταβλητή size ορίζει το μέγεθος του πίνακα και είναι ορισμένη με την τιμή 8 καθώς είναι το στάνταρ μέγεθος του πίνακα του παιχνιδιού Reversi. Η μεταβλητή player εναλλάσσεται από την τιμή 1 σε 2 και αντίστροφα. Ο πίνακας weights συμβολίζει τα βάρη που θα χρησιμοποιήσουμε αργότερα στην ευρετική μας (function evaluate). Οι συναρτήσεις που την αποτελούν είναι οι εξής:
  - ✓ **`public Board():`** Στην κλάση Board ορίζουμε τον πίνακα σε διάσταση [8][8] αλλά αυτό μπορεί να αλλάξει αν αλλάξουμε την τιμή της size για μικρότερο πίνακα. Αρχικοποιούμε όλες τις θέσεις του πίνακα με την τιμή 0 δηλαδή την τιμή της κενής θέσης. Έπειτα τοποθετούμε τα δύο X και τα δύο O στις σωστές θέσεις του πίνακα όπως ορίζεται από το παιχνίδι Reversi. Όπως αναφέραμε προηγουμένως το 1 συμβολίζει τα X και το 2 τα O. Τέλος καλούμε την συνάρτηση setWeights() της οποίας η περιγραφή αναφέρεται παρακάτω.

  - ✓ **`public void setMaxDepth(int maxDepth):`** Επιλέγεται στη main από τον χρήστη το μέγιστο βάθος αναζήτησης και ενημερώνεται η μεταβλητή maxDepth.

  - ✓ **`public void setWeights():`** Όπως αναφέραμε προηγουμένως ο πίνακας weights μας χρησιμεύει στην ευρετική μας. Τοποθετούμε στις 4 γωνίες του πίνακα τα μεγαλύτερα βάρη (100) καθώς αυτά θα μας δώσουν καλύτερες θέσεις και προοπτικές αργότερα στην ευρετική και καθώς πηγαίνουμε προς το κέντρο του πίνακα τα βάρη μειώνονται.

  - ✓ **`Board(Board b):`** Παίρνουμε ένα αντίγραφο του πίνακά μας και του τρέχοντος παίκτη έτσι ώστε να χρησιμοποιηθεί αργότερα όταν

θελήσουμε να παράγουμε τα παιδιά της κάθε δυνατής κίνησης χωρίς να αλλοιώσουμε τον κανονικό πίνακά μας (this.board).

- ✓ **Public void setPlayer(int p):** Θέτουμε την τιμή του παίκτη. Χρησιμοποιείται στην main μας στην Reversi.java για να ορίσουμε την τιμή εκκίνησης του παίκτη και με ποιο σύμβολο Χ ή Ο θα ξεκινήσει αν και εμείς λαμβάνουμε ότι ο παίκτης πάντα παίζει με τα Χ και ο υπολογιστής με τα Ο.

- ✓ **Public void changePlayer():** Αλλάζουμε την τιμή του Player έτσι ώστε να παίξει ο αντίπαλος.

- ✓ **Public int otherPlayer():** Επιστρέφουμε την τιμή του αντίπαλου παίκτη δίχως να αλλάξουμε την τιμή του τρέχοντος παίκτη. Η χρήση της μεθόδου θα εξηγηθεί αναλυτικά σε παρακάτω μεθόδους (moveUp, moveDown…etc).

- ✓ **Public Boolean isValidMove(int x,int y):** Ελέγχουμε εάν η σειρά (x) και στήλη (y) είναι αρχικά εντός ορίων του πίνακα και δεύτερον εάν η θέση του πίνακα με συντεταγμένες x και y είναι κενή. Επιστρέφουμε αληθής εάν ισχύουν τα παραπάνω αλλιώς επιστρέφουμε την τιμή ψευδής.

- ✓ **Public Boolean checkDirections(int x,int y):** Καθεμία από τις συναρτήσεις moveRight, moveLeft που θα αναλυθούν παρακάτω επιστρέφουν μία τιμή int (-1 εάν δεν μπορεί να τοποθετηθεί στοιχείο ή την τιμή μέχρι την οποία θα τοποθετηθεί το στοιχείο) οπότε εάν το άθροισμα και των 8 αυτών συναρτήσεων είναι -8 σημαίνει πως δεν μπορεί να τοποθετηθεί στοιχείο προς καμία κατεύθυνση του πίνακα και επιστρέφουμε την τιμή ψευδής. Εάν κάποια συνάρτηση επιστρέψει τιμή διάφορη από το -1 τότε το άθροισμα θα είναι διάφορο του -8 και η συνάρτηση θα επιστρέψει την τιμή αληθής.

- ✓ **Public Boolean checkValidMove(int x,int y):Καλεί** τις δύο παραπάνω συναρτήσεις και αν είναι και οι δύο αληθείς επιστρέφει την τιμή αληθής αλλιώς την τιμή ψευδής.

- ✓ **Public ArrayList<Move> getAvailableMoves():** Βρίσκει όλες τις διαθέσιμες θέσεις που μπορεί ο player να τοποθετήσει το πούλι του. Διατρέχουμε όλες τις δυνατές συντεταγμένες του πίνακα και ελέγχουμε για κάθε ζεύγος αν είναι αληθής η checkValidMove.

Εάν ναι τότε προσθέτουμε το ζεύγος σε μία ArrayList τύπου Move την οποία και επιστρέφουμε.

- ✓ **Public int moveLeft(int x,int y):** Έστω ότι θέλουμε να τοποθετήσουμε στοιχείο στη θέση με συντεταγμένες x,y τότε πρέπει να τσεκάρουμε εάν μπορούμε προς κάποια δυνατή κατεύθυνση να αιχμαλωτίσουμε το πούλι του αντιπάλου και να αντικατασταθεί με το δικό μας. Για να κινηθούμε προς τα αριστερά παραμένουμε στην ίδια σειρά αλλά αλλάζουμε στήλη καθώς η στήλη μειώνεται. Οπότε αρχικά ελέγχουμε εάν με την μία κίνηση προς τα αριστερά δεν είμαστε εκτός ορίων του πίνακα. Εάν είμαστε εντός τότε ελέγχουμε εάν το στοιχείο που βρίσκεται σε αυτή τη θέση είναι του αντιπάλου καθώς εάν είναι το ίδιο ή η κενή θέση τότε δεν μας ενδιαφέρει αφού δεν μπορούμε να κάνουμε κίνηση προς αυτή την κατεύθυνση. Αυτό γίνεται με την χρήση της συνάρτησης otherPlayer() η οποία μας επιστρέφει το στοιχείο του αντίπαλου παίκτη χωρίς να αλλάζουμε τον παίκτη. Αν τώρα στην μία θέση αριστερά υπάρχει πούλι του αντιπάλου τότε πρέπει να διατρέξουμε τον πίνακα προς εκείνη την κατεύθυνση όσο υπάρχουν πούλια του αντιπάλου(και πάντα είμαστε εντός ορίων) και μέχρι να βρούμε δικό μας πούλι. Αν συμβεί αυτό τότε μπορούμε να τα αιχμαλωτίσουμε και επιστρέφουμε την τιμή της στήλης όπου βρήκαμε ξανά δικό μας πούλι. Εάν τώρα κατά τη διάρκεια που βρίσκουμε πούλια του αντιπάλου βρούμε την κενή θέση και όχι το δικό μας πούλι τότε η αναζήτηση σταματά καθώς δεν μπορούμε να αιχμαλωτίσουμε κανένα πούλι και επιστρέφουμε -1. Αντίστοιχα για τις συναρτήσεις moveRight, moveUp, moveDown.

- ✓ **Public int moveUpLeft(int x,int y):** Αντίστοιχα για αυτή την συνάρτηση με την από πάνω με τη διαφορά ότι στις διαγώνιους έχουμε μετακίνηση και της σειράς και της στήλης. Για να κινηθούμε πάνω αριστερά μειώνονται και οι σειρές και οι στήλες. Αντίστοιχα για τη moveUpRight μόνο που αυξάνεται η στήλη προς τα δεξιά και μειώνεται η σειρά προς τα πάνω, για τη moveDownLeft αυξάνεται η σειρά προς τα κάτω και μειώνεται η στήλη προς τα αριστερά και τέλος για τη moveDownRight αυξάνεται και η σειρά και η στήλη προς κάτω δεξιά.

- ✓ **Public void makeMove(int row,int col):** Η συνάρτηση καλεί τις 8 συναρτήσεις moveLeft-Right-Up… για κάθε πιθανή θέση που μπορεί να ελέγξει τοποθετώντας ένα πούλι (Χ ή Ο), δηλαδή πάνω, κάτω,

δεξιά, αριστερά, και στις διαγώνιους. Η κάθε συνάρτηση από αυτές επιστρέφει έναν ακέραιο αριθμό, -1 αν δεν μπορεί να τοποθετηθεί πούλι ή έναν αριθμό από 0 έως και 7 αν μπορεί να τοποθετηθεί, όπου αυτός ο αριθμός είναι ο αριθμός μέχρι τον οποίο μπορεί να τοποθετηθεί το ίδιο στοιχείο. Αν επιστραφεί αριθμός εκτός του -1 καλείται η συνάρτηση setElement η οποία είναι υπεύθυνη για την τοποθέτηση του ίδιου στοιχείου. Η setElement λαμβάνει 4 ορίσματα τα πρώτα δύο είναι η σειρά row και η στήλη column του στοιχείου που προσπαθούμε να τοποθετήσουμε και τα επόμενα δύο είναι η σειρά και η στήλη του στοιχείου μέχρι το οποίο θα γεμίσουμε με αυτό τον πίνακα. Τώρα για τα δύο τελευταία ορίσματα εξαρτάται σε ποια συνάρτηση είμαστε για να τα συμπληρώσουμε αναλόγως. Για παράδειγμα όπως έχουμε ήδη αναφέρει στην moveLeft παραμένει ίδια η σειρά οπότε στη θέση της «καινούριας» σειράς έχουμε την αρχική και στη θέση της στήλης έχουμε την ίδια την συνάρτηση η οποία θα επιστρέψει την τιμή της «νέας» στήλης. Η διαδικασία είναι παρόμοια για τις υπόλοιπες συναρτήσεις με μερικές διαφορές για τις διαγώνιους. Γνωρίζουμε πως στις διαγώνιους αλλάζουν και οι σειρές και οι στήλες οπότε ας πάρουμε ως παράδειγμα την moveUpLeft η οποία θα επιστρέψει την τελική σειρά και θα την τοποθετήσει ως όρισμα της νέας σειράς, όμως τί θα θέσουμε ως όρισμα της στήλης. Στην συγκεκριμένη συνάρτηση μειώνεται και η σειρά και η στήλη οπότε η τελική τιμή της στήλης θα είναι μικρότερη από την αρχική. Γνωρίζουμε πως η αρχική σειρά είναι η row και η τελική σειρά (μικρότερη από την αρχική) είναι η moveUpLeft(row,col) οπότε η διαφορά αυτών των δύο είναι η row-moveUpLeft(row,col). Την ίδια διαφορά που θα έχουν οι σειρές θα έχουν και οι στήλες οπότε η τιμή της τελικής στήλης θα είναι η αρχική στήλη col μείον τη διαφορά row-moveUpLeft(row,col). Με ανάλογο τρόπο υπολογίζονται και οι υπόλοιπες διαγώνιοι. Στον κώδικα υπάρχουν παραδείγματα και επεξηγήσεις με σχόλια δίπλα στις υπόλοιπες διαγώνιους για επεξήγηση.

- ✓ **`Public void setElement(int x1,int y1,int x2,int y2):`** Η συνάρτηση setElement καλείται από την makeMove για την κατοχύρωση των πουλιών του αντιπάλου και το «γέμισμα» στον πίνακα. Χωρίζουμε το «γέμισμα» σε 4 κατηγορίες.
  1. Ίδια σειρά αλλάζει η στήλη (moveLeft, moveRight)
  2. Ίδια στήλη αλλάζει η σειρά (moveUp, moveDown)
  3. Άθροισμα αρχικών συντεταγμένων ίσο με άθροισμα τελικών συντεταγμένων (δευτερεύουσα διαγώνιος) (moveDownLeft, moveUpRight)

4. Άθροισμα αρχικών συντεταγμένων διάφορο με άθροισμα τελικών συντεταγμένων (κύρια διαγώνιος) (moveDownRight, moveUpLeft)

1. Αφού παραμένει σταθερή η σειρά ελέγχουμε ποια στήλη είναι μικρότερη και ποια μεγαλύτερη και παραμετροποιούμε αντίστοιχα και γεμίζουμε τον πίνακα με το στοιχείο του τρέχοντος παίκτη από low έως high.
2. Αντίστοιχα παραμετροποιούμε τις σειρές.
3. Ελέγχουμε ποια σειρά είναι η μικρότερη και αφού βρισκόμαστε στη δευτερεύουσα διαγώνιο τότε η στήλη θα έχει την αντίθετη κατεύθυνση από τη σειρά πχ αν αυξάνεται η σειρά τότε θα μειώνεται η στήλη. Διατρέχουμε τον πίνακα από τη μικρότερη έως τη μεγαλύτερη σειρά και από τη μεγαλύτερη στήλη προς τη μικρότερη και γεμίζουμε τον πίνακα.
4. Αντίστοιχα μόνο που οι σειρές και οι στήλες στην κύρια διαγώνιο έχουν την ίδια κατεύθυνση.

- ✓ **`Void print():`** Η all_plays είναι ένας μετρητής που δηλώνει συνολικά πόσες φορές έπαιξαν και οι δύο παίκτες αυτό μας ενδιαφέρει μόνο στην πρώτη φορά που θα εκτυπωθεί ο πίνακας καθώς δε θέλουμε να εκτυπωθεί human played or computer played καθώς είναι απλά ο αρχικός πίνακας και όχι η κίνηση κάποιου παίκτη. Διατρέχουμε τον πίνακα και άμα το στοιχείο είναι 1 εμφανίζει Χ αν είναι 2 εμφανίζει Ο και εάν είναι 0 τότε εμφανίζει ένα στοιχείο string το οποίο επειδή μερικές φορές στη cmd δεν αναγνωρίζεται θα αντικατασταθεί στον κώδικα με ?. Τα παραδείγματα του κώδικα και τα screenshots του πώς τρέχει θα είναι με το αρχικό στοιχείο.

- ✓ **`Public int freeMoves():`** Υπολογίζει τις κενές θέσεις του πίνακα.

- ✓ **`Public static Boolean isInList(final ArrayList<Move> list,int x,int y):`** Ελέγχει και επιστρέφει αληθής εάν υπάρχει στη λίστα στοιχείο με συντεταγμένες x και y. Χρησιμοποιείται στη συνάρτηση printAvailableMoves().

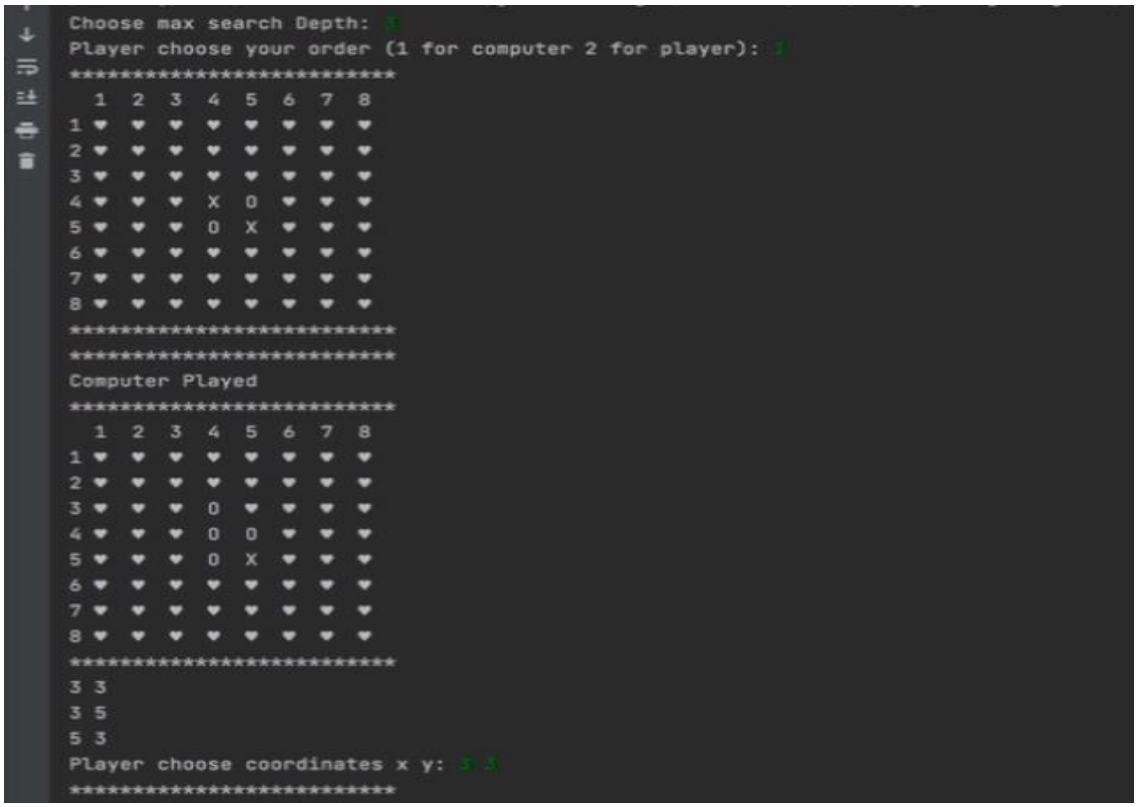
- ✓ **`Public int evaluate(Board b):`** Η ευρετική. Έχουμε δύο αθροίσματα ένα για τα Χ και ένα για τα Ο. Όποτε βρίσκει Χ αυξάνει το άθροισμα του Χ κατά το βάρος του πίνακα weights που βρίσκεται στη συγκεκριμένη θέση καθώς και το συνολικό πλήθος των Χ που βρίσκει (αύξηση κατά 1) και αντίστοιχα για το Ο και επιστρέφει τη διαφορά τους. Ο πίνακας weights έχει διαμορφωθεί με τέτοιο τρόπο έτσι ώστε στις

γωνίες του πίνακα έχει τα μεγαλύτερα βάρη καθώς άμα κάποιος παίκτης κατοχυρώσει αυτή τη θέση έχει πλεονέκτημα καθώς ο αντίπαλος παίκτης δεν μπορεί να τον αιχμαλωτίσει και να του αλλάξει τα πούλια σε εκείνη τη θέση. Προς τα κεντρικά σημεία του πίνακα έχουμε μικρότερα βάρη γιατί δεν είναι το ίδιο καλές θέσεις με τις ακριανές.

- ✓ **Public Move pickMove(Board b):** Παίρνουμε στην μεταβλητή moves που είναι τύπου Arraylist τις ελεύθερες κινήσεις του υπολογιστή και προχωρούμε εάν έχει ελεύθερες κινήσεις. Αρχικοποιούμε μια μεταβλητή position με 0 η οποία θα κρατάει την θέση της τρέχουσας καλύτερης κίνησης μέχρι να βρεθεί καλύτερη. Παίρνουμε όλα τα παιδιά της κάθε κίνησης (getchildren) από τη moves και για κάθε παιδί καλούμε τη minimax η οποία θα μας επιστρέψει έναν αριθμό ο οποίος θα χαρακτηρίζει το πόσο καλή είναι η κίνηση του συγκεκριμένου παιδιού. Εάν είναι καλύτερη τότε η μεταβλητή maxMove θα πάρει την τιμή που επιστράφηκε και θα κρατήσουμε σε ποια θέση του πίνακα βρέθηκε. Τέλος θα επιστρέψουμε την κίνηση από τον πίνακα moves που θα έχει τη θέση της καλύτερης που έχουμε βρει.

- ✓ **Public int minimax(Board b,int depth,int maxMove,int minMove,Boolean max player):** Κρατάμε πάλι σε μια μεταβλητή τύπου arraylist τις ελεύθερες κινήσεις του υπολογιστή. Και αν ο πίνακας δεν είναι τερματικός δηλαδή τουλάχιστον ένας από τους δύο παίκτες έχει ακόμη κινήσεις να παίξει ή δεν έχουμε φτάσει στο μέγιστο βάθος αναζήτησης προχωρούμε στη max(max_player =true ή στη min αντίστοιχα(max_player =false). Για κάθε κίνηση ελέγχουμε εάν η maxMove είναι μεγαλύτερη ή ίση με τη minMove ή αν η κίνηση είναι κενή. Εάν ισχύει κάποια από τις δύο περιπτώσεις δεν υπάρχει λόγος να συνεχίσουμε και επιστρέφουμε την κίνηση. Αν δεν βρισκόμαστε σε αυτή την περίπτωση τότε παράγουμε ένα αντίγραφο του board μας και κάνουμε κίνηση στον πίνακα καλώντας μετά την αντίθετη μέθοδο (max->min ή min->max) για το επόμενο βάθος ψάχνοντας την καλύτερη κίνηση την οποία θα επιστρέψουμε στο τέλος. Εάν οι κινήσεις έχουν ίδια «αξία» τότε επιλέγουμε μία στην τύχη.

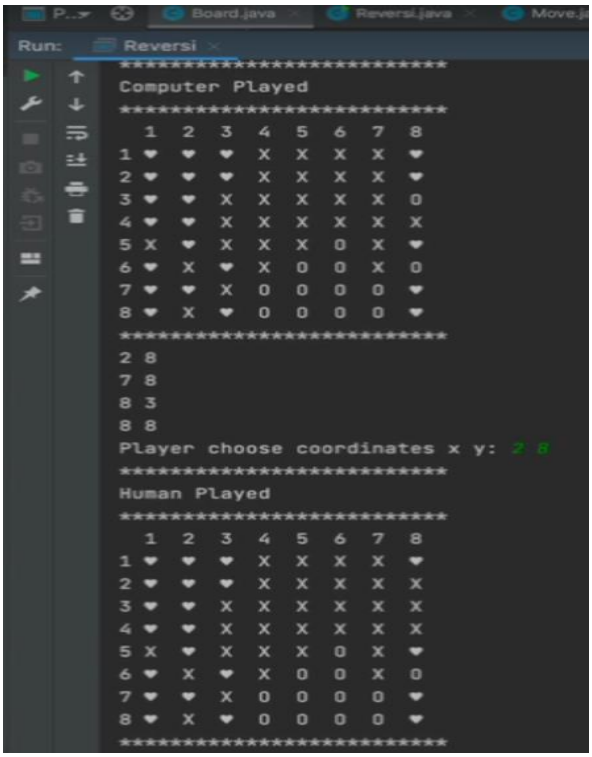
- ✓ **Boolean isTerminal():** Ελέγχουμε αν οι παίκτες έχουν ελεύθερες κινήσεις. Εάν έστω και ένας παίκτης έχει τότε το παιχνίδι συνεχίζεται. Εάν και οι δύο παίκτες δεν έχουν κινήσεις τότε το παιχνίδι τελειώνει και τυπώνεται το τελικό σκορ.

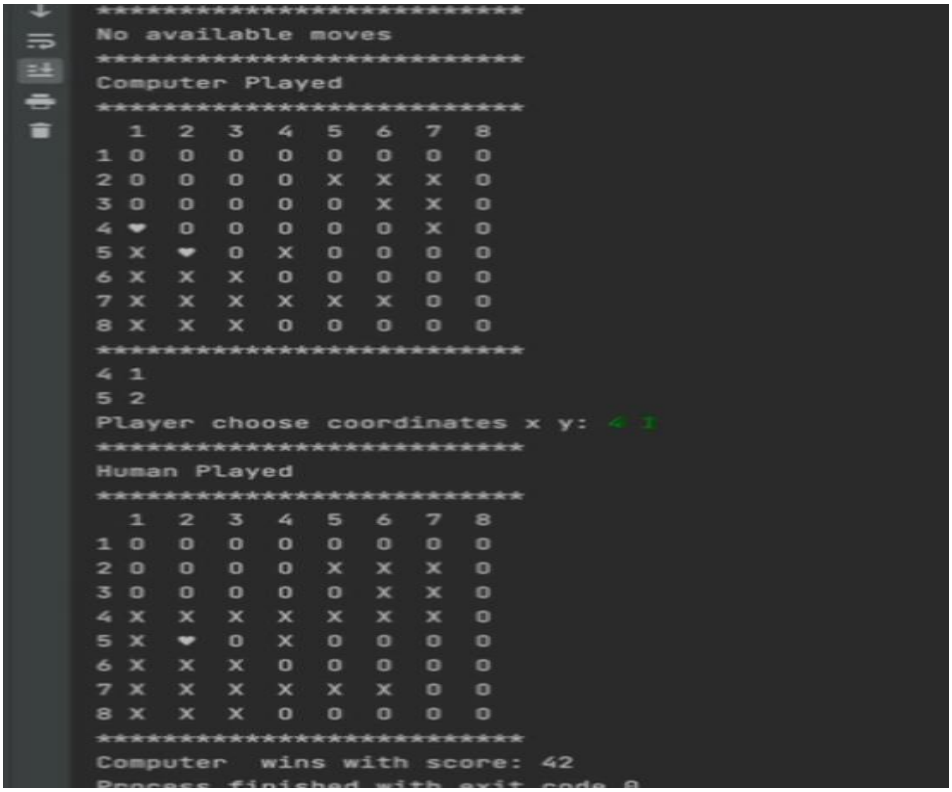


Μέγιστο βάθος: 3, ξεκινάει ο υπολογιστής και οι επιλογή συντεταγμένων του χρήστη είναι: 3 3

Ο παίκτης δεν έχει διαθέσιμες κινήσεις και παίζει ο υπολογιστής

Ενδιάμεσο στάδιο του παιχνιδιού





Τελειώνει το παιχνίδι με νικητή τον υπολογιστή με σκορ 42 πούλια με σχήμα Ο.