Μυρτώ Χριστίνα Ελευθέρου – 3170046

Χρυσόστομος Ισκάκης – 3170052

Χρυσούλα Οικονόμου – 3170127

1η Εργασία - Reversi

Η υλοποίηση του προγράμματος γίνεται με τον αλγόριθμο *Minimax* με πριόνισμα α-β. Η υλοποίησή του Minimax γίνεται στην κλάση **GamePlayer.** Στην **Board** γίνονται όλοι οι έλεγχοι και οι κινήσεις που είναι απαραίτητες για το παιχνίδι.

Στην **Board**, κατασκευάζεται μέσω του constructor, ο αρχικός πίνακας, ο οποίος έχει διαστάσεις 8x8 και τοποθετούνται στις default θέσεις τα 4 πρώτα πούλια. Κάθε μία από τις 4 σταθερές, αντιπροσωπεύουν μία διαφορετική κατάσταση που μπορεί να υπάρχει σε μία συγκεκριμένη θέση του πίνακα. Ο Χ θα αναπαρίσταται από την τιμή 1, ο Ο από την τιμή -1, μία άδεια θέση του πίνακα από την τιμή 0 και οι δυνατές τιμές στις οποίες μπορεί να τοποθετήσει το πούλι του ο παίκτης από την τιμή 100. Στην κλάση αυτή, υλοποιούνται επίσης και όλες οι μέθοδοι που είναι απαραίτητες για την διεξαγωγή του παιχνιδιού:

- η **makeMove** χρησιμοποιείται στην *Main*, και πραγματοποιεί την εκάστοτε κίνηση. Λαμβάνει ως ορίσματα την γραμμή και την στήλη στην οποία θα τοποθετηθεί το πούλι, το σύμβολο του παίκτη που παίζει κάθε φορά και την λίστα *dir* η οποία περιέχει όλες τις κατευθύνσεις, προς τις οποίες έχουμε ήδη ελέγξει ότι μπορεί να «πιάσει» τα πούλια του αντίπαλου παίκτη. Αφού τοποθετηθεί το πούλι στις συντεταγμένες *row*, *col*, στην συνέχεια, για κάθε κατεύθυνση που θα βρει στην *dir*, θα κινηθεί προς αυτή και θα μετατρέψει κάθε πούλι του αντίπαλου παίκτη στο δικό του χρώμα.
- η **print** χρησιμοποιείται στην *Main* και εκτυπώνει το ταμπλό του παιχνιδιού. Για κάθε κατάσταση εμφανίζει όλα τα πούλια, σε περίπτωση που παίζει ο παίκτης εμφανίζει και τις διαθέσιμες θέσεις, ενώ ακόμη, αριθμεί και τις θέσεις του ταμπλό.
- η **getChildren** χρησιμοποιείται στην **GamePlayer**, στις μεθόδους *min* και *max*. Διασχίζει όλο τον πίνακα και επιστρέφει ένα *Arraylist* children, το οποίο περιέχει όλα τα παιδιά(επόμενες καταστάσεις), που μπορούν να προκύψουν από την κατάσταση που βρισκόμαστε.
- η **isTerminal** χρησιμοποιείται για να ελέγχει αν το παιχνίδι πρέπει να συνεχιστεί ή αν πρέπει να τερματίσει. Κατά την υλοποίηση της καλεί τρεις ακόμα συναρτήσεις, τις **boardIsFull**, **onlyOneColor**, και **notValidMoves**.
  - Η **boardIsFull** ελέγχει αν όλες οι θέσεις του πίνακα είναι γεμάτες. Αν είναι επιστρέφει *true*, αλλιώς επιστρέφει *false*.

- Η **onlyOneColor**, ελέγχει αν ο πίνακας περιέχει πούλια μόνο του ενός συμβόλου (πχ. όλα Χ). Αν ναι τότε επιστρέφει *true*, αλλιώς *false*.
- Η **notValidMoves** ελέγχει αν κανείς από τους παίκτες δεν έχει διαθέσιμη κίνηση. Αν ισχύει τότε επιστρέφει *true*, αλλιώς *false*.

Τέλος αν και οι τρεις επιστρέφουν *false* στην **isTerminal**, τότε το παιχνίδι συνεχίζεται κανονικά, αλλιώς αν έστω και μία επιστρέψει *true* το παιχνίδι τερματίζει.

- η **heuristic** υλοποιεί την ευρετική συνάρτηση. Ως ευρετική, έχουμε ορίσει η συνάρτηση να κάνει τα εξής:
    1. Να μετράει τον αριθμό από τα πούλια που έχει συνολικά ο κάθε παίκτης (κόστος: 1).
    2. Να ελέγχει αν βρίσκει τελική κατάσταση, δηλαδή αν ο πίνακας αποτελείται μόνο από πούλια του παίκτη που παίζει, και επομένως ο παίκτης να κερδίζει. (κόστος: 10000)
    3. Να ελέγχει αν ο παίκτης έχει ως διαθέσιμη κίνηση κάποια γωνία (κόστος: 100).
    4. Να ελέγχει αν ο παίκτης έχει κάποια διαθέσιμη κίνηση σε κάποια ακριανή στήλη ή γραμμή (κόστος 10).

Τα παραπάνω είναι ουσιαστικά καλές στρατηγικές για να κερδίσει κάποιος το παιχνίδι.

Πιο συγκεκριμένα, η τρίτη αποτελεί μία από τις σημαντικότερες, καθώς είναι καλή τεχνική να παίζει κάποιος στις γωνίες, αφού στο μέλλον είναι αδύνατο να τις αλλάξει ο αντίπαλος. Παράλληλα οι γωνίες, είναι σχεδόν σίγουρο ότι θα μας δώσουν διαγώνιους και στήλες/γραμμές με τα πούλια μας. Συνεπώς, οι γωνίες είναι τα πιο «δυνατά» τετράγωνα που μπορούμε να επιλέξουμε. Όσο πιο πολλές γωνίες έχουμε στην κατοχή μας, τόσο πιο πολλές οι πιθανότητες να κερδίσουμε το παιχνίδι.

Μία ακόμα καλή στρατηγική που αναφέραμε είναι αυτή που επιλέγουμε τις διαθέσιμες κινήσεις στις άκρες (δηλαδή ακριανές στήλες ή γραμμές), διότι αν βάλουμε δικό μας πούλι είναι δύσκολο να μεταβληθεί από τον αντίπαλο παίκτη.

Τέλος, οι πρώτες δύο στρατηγικές είναι πιο απλές, καθώς στην πρώτη η συνάρτηση μετρά απλά τον αριθμό από τα πούλια που θα αλλάξουν στο δικό μας σύμβολο, και στη δεύτερη ελέγχει αν με τις κατάλληλες κινήσεις μπορεί ο παίκτης να «εξολοθρεύσει» τα πούλια του αντιπάλου και να κερδίσει.

- η **availableMoves** ενημερώνει τις διαθέσιμες κινήσεις και επιστρέφει μία *boolean* τιμή και καλείται στην *Main*. Στην περίπτωση που είναι η σειρά του παίκτη να παίξει, λαμβάνει ως όρισμα την τιμή *false*, έτσι ώστε να ψάξει αν υπάρχει κάποια κίνηση διαθέσιμη. Αν υπάρχουν διαθέσιμες κινήσεις, τότε ορίζει ποιες είναι αυτές και επιστρέφει την τιμή *true*. Σε κάθε άλλη περίπτωση

επιστρέφει *false*. Εάν είναι σειρά του υπολογιστή να παίξει ή αν ο παίκτης έχει παίξει και θέλουμε να ενημερώσουμε το ταμπλό μετά την κίνηση που έκανε, τότε δίνουμε σαν όρισμα την τιμή true και όλα τα * μετατρέπονται σε κενά.

- η **directions** καλείται στην *Main* και επιστρέφει μία λίστα με όλες τις κατευθύνσεις προς τις οποίες μπορεί να μετατρέψει τα πούλια του αντιπάλου σε δικά του. Αν η λίστα που επιστρέψει είναι κενή τότε σημαίνει ότι στην θέση που επέλεξε, δεν μπορεί να τοποθετηθεί πούλι και άρα δεν είναι έγκυρη. Λαμβάνει ως ορίσματα την γραμμή και την στήλη που ο παίκτης ή το πρόγραμμα προσπαθεί να τοποθετήσει το πούλι και το γράμμα του εκάστοτε παίκτη. Η directions ελέγχει όλες τις γειτονικές θέσεις του σημείου που θέλουμε να τοποθετήσουμε το πούλι(πχ. το τετράγωνο κάτω, το τετράγωνο διαγώνια δεξιά κ.λπ.), και ψάχνει αν υπάρχει το αντίθετο γράμμα(π.χ. αν παίζει ο Χ, κοιτάει αν υπάρχει δίπλα του, προς οποιαδήποτε κατεύθυνση κάποιο Ο). Αν βρει το αντίθετο γράμμα, τότε καλεί την *isValid* και σε περίπτωση που αυτή η κατεύθυνση είναι έγκυρη την προσθέτει στην λίστα.
- η **isValid** καλείται από την *directions* και παίρνει ως ορίσματα την θέση στην οποία βρέθηκε το πούλι του αντίπαλου παίκτη, τον παίκτη που παίζει και ένα string το οποίο δείχνει σε ποια κατεύθυνση βρέθηκε το αντίπαλο πούλι(π.χ. αν <u>κάτω</u> από το Ο βρεθεί ένα Χ, τότε το string θα έχει την τιμή "*d*"). Στην συνέχεια, ελέγχει αν προς την συγκεκριμένη κατεύθυνση, βρεθεί άλλο πούλι του παίκτη που παίζει. Αν βρει, τότε επιστρέφει την κατεύθυνση αυτή ώστε να προστεθεί στην λίστα *directions*, διαφορετικά επιστρέφει *null*.

Ο έλεγχος για το αν κάποια κίνηση είναι έγκυρη έγινε με αυτόν τον τρόπο καθώς αρχικά, θέλουμε να ελέγξουμε αν δίπλα από την θέση που ζητάμε να τοποθετηθεί το πούλι, υπάρχει πούλι του αντίπαλου παίκτη. Αν βρεθεί πούλι του αντιπάλου, τότε ελέγχουμε αν ,στην συνέχεια, βρεθεί πούλι του παίκτη που παίζει, χωρίς να μεσολαβεί κάποιο κενό. Αν δεν υπάρχει πούλι του αντίπαλου παίκτη δίπλα, τότε σίγουρα η κίνηση αυτή δεν είναι έγκυρη.

- η **score** εκτυπώνει πόσα πούλια από τον κάθε παίκτη υπάρχουν στο ταμπλό, μετά από κάθε κίνηση.
- η **winner** εκτυπώνει στο τέλος ποιος είναι ο νικητής ή σε περίπτωση ισοπαλίας εμφανίζει αντίστοιχο μήνυμα.
- η **increaseScore** διασχίζει ολόκληρο τον πίνακα και για κάθε πούλι του Χ ή του Ο που συναντάει, αυξάνει το σκορ του.

Στην **GamePlayer** υλοποιείται ο αλγόριθμος *Minimax* με πριόνισμα α-β. Αρχικά, στην μέθοδο *move*, καλείται η μέθοδος *max* αν είναι σειρά του Χ να παίξει ή η *min* αν είναι σειρά του Ο να παίξει αντίστοιχα. Οι 2 μέθοδοι, έχουν παρόμοια λειτουργία. Για κάθε παιδί που δημιουργείται από την *getChildren*, ελέγχουμε αν το *value* του συγκεκριμένου παιδιού είναι μεγαλύτερο(για τον *max*) ή μικρότερο(για τον *min*) από την τιμή που είχαμε ως τώρα. Στο τέλος, επιστρέφεται η ελάχιστη και

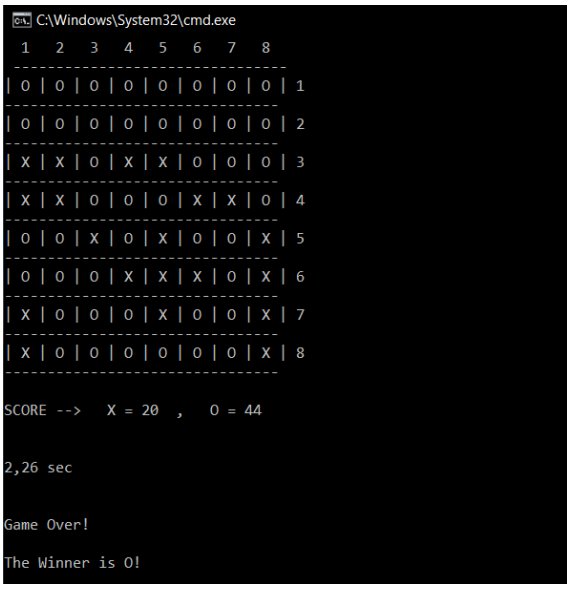
η μέγιστη τιμή που βρέθηκε σε κάθε μέθοδο. Για το πριόνισμα αρχικοποιούμε τις 2 τιμές *a* και *b* ώστε να είναι τα μετρά σύγκρισης. Κάθε τιμή που επιστρέφεται από τις *min* και *max* συγκρίνεται με το *a* και *b* αντίστοιχα και αποφασίζεται αν θα παραλειφθεί το άλλο κλαδί του κόμβου επιστροφής ή όχι.

Στην **Main,** έχουμε βάλει καθυστέρηση χρόνου κάθε φορά που εμφανίζεται το ταμπλό, έτσι ώστε να είναι πιο εύκολο για τον χρήστη να παρακολουθεί την πορεία του παιχνιδιού.

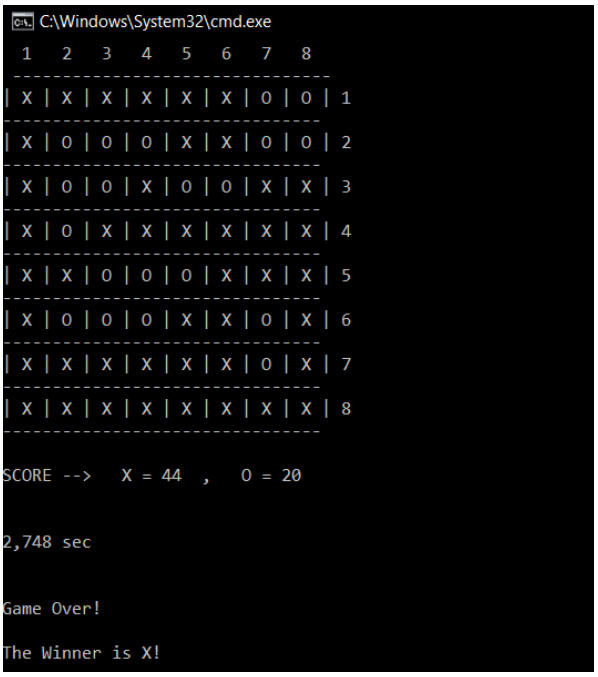
ΠΑΡΑΔΕΙΓΜΑΤΑ

- *depth* = 2

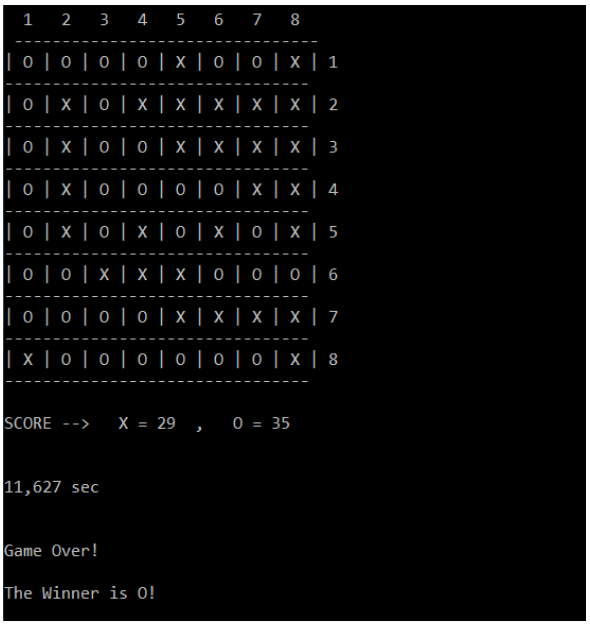
- *depth* = 3



- depth = 4



- depth =5



Οι δοκιμές έγιναν σε λειτουργικό *Windows*, 6 *GB ram*, 2πύρηνο επεξεργαστή και *java version* 14.