ΤΕΧΝΗΤΗ ΝΟΗΜΟΣΥΝΗ

ΕΡΓΑΣΙΑ ΕΞΑΜΗΝΟΥ

Μέλη ομάδας:

Αθανασόπουλος Αντύρας Φίλιππος | 5113

Νίκος Κωνσταντινίδης | 5155

Νίκος Παπαδόπουλος | 4938

Όλες οι μετρήσεις έγιναν στο παρακάτω μηχανήματα

Όνομα	#cores	Μνήμη
Dell G15	6	16 GB
Dell Aura R10	6	32 GB
dl380ws01	4	8GB

Οδηγίες εκτέλεσης στο τέλος της αναφοράς

Άσκηση 2

Στην $2^{η}$ άσκηση μας δίνεται ένα παιχνίδι καρτών. Η τράπουλα χωρίζεται σε ομάδες , η κάθε μια με τουλάχιστον 2 κάρτες. Για κάθε ομάδα ορίζεται ο μέγιστος αριθμός καρτών που μπορούμε να αφαιρέσουμε ανά γύρο. Οι παίκτες παίζουν εναλλάξ , βγάζοντας κάρτες από μια ομάδα ανά γύρο. Νικητής είναι ο παίκτης που βγάζει την τελευταία κάρτα από όλη τη τράπουλα.

Υλοποίηση καρτών , ομάδων ,τράπουλας και χειριστή τράπουλας.

CardGroup : σε κάθε ομάδα καρτών αποθηκεύεται το ID του , ο αριθμός καρτών και το μέγιστο πλήθος αφαιρέσιμων καρτών ανά γύρο.

CardDeck : η τράπουλα διαθέτει μια λίστα με όλες τις ομάδες καρτών καθώς και τον συνολικό αριθμό καρτών και ομάδων.

CardDealer : ο διαχειριστής της τράπουλας διαθέτει την τράπουλα και μεθόδους για την διαχείριση της. Στην αρχή του παιχνιδιού ζητά πληροφορίες για την κατασκευή της προπουλάς. Κατά την διάρκεια του παιχνιδιού ζητάει κινήσεις από τους παίκτες.

Υλοποίηση κόμβων (Node):

Κάθε Node αποτελεί μια κατάσταση του παιχνιδιού.
Συγκεκριμένα για κάθε κατάσταση αποθηκεύουμε :

- Τα παιδιά του
- Τις κάρτες που αφαιρέθηκαν από τον γονέα για να φτάσουμε στον κόμβο αυτό.
- Την τράπουλα όπως είναι αυτή τη στιγμή
- Την ομάδα από την οποία αφαιρέθηκαν οι κάρτες
- Αν ο κόμβος είναι Max η Min
- Το score του κόμβου

Σημαντικότερες μέθοδοι :

createChildren() : δέχεται ένα CardDeck και δημιουργεί κόμβους για όλες τις δυνατές κινήσεις που μπορούν να γίνουν. Για τις τελευταίες κινήσεις , δηλαδή τις κινήσεις που αφαιρείται το τελευταίο χαρτί από την τράπουλα ορίζει και το score.

Υλοποίηση AI (Botaki):

Το Botaki σε κάθε γύρο δέχεται την κατάσταση της τράπουλας και υπολογίζει την βέλτιστη κίνηση με τον αλγόριθμο MiniMax.

Σημαντικότερες μέθοδοι:

findBestMoveWithMinimax() : δέχεται το δέντρο του παιγνίου και εφαρμόζει τον αλγόριθμο MiniMax. Μετά την εκτέλεση του ο κόμβος root γνωρίζει το score του παιδιού που πρέπει να επιλέξει , δηλαδή την καλύτερη επόμενη κίνηση.

executeBestMove() : αφού εκτελεστεί ο αλγόριθμος MiniMax , το Botaki ανανεώνει την τιμή της ομάδας που πρέπει να διαλέξει και των καρτών που πρέπει να αφαιρέσει στον γύρο αυτό. Οι τιμές αυτές μετρά τον υπολογισμό τους δίνονται στον διαχειριστή της τράπουλας.

<u>Υλοποίηση αλγορίθμου MiniMax :</u>

Αφού δημιουργηθεί το δέντρο αποφάσεων , ο αλγόριθμος ακολουθεί τα εξής βήματα :

*Για κάθε **παιδί** του κόμβου :*

***ΑΝ** το παιδί είναι ΤΚ :*

Πρόσθεσε το score στη λίστα σύγκρισης

***ΑΛΛΙΩΣ** :*

Επανέλαβε την μέθοδο για το παιδί

*/*Έχουμε τα score από όλα τα παιδιά ενός κόμβου*/*

***ΑΝ** ο κόμβος είναι Max : score <- max(Scores)*

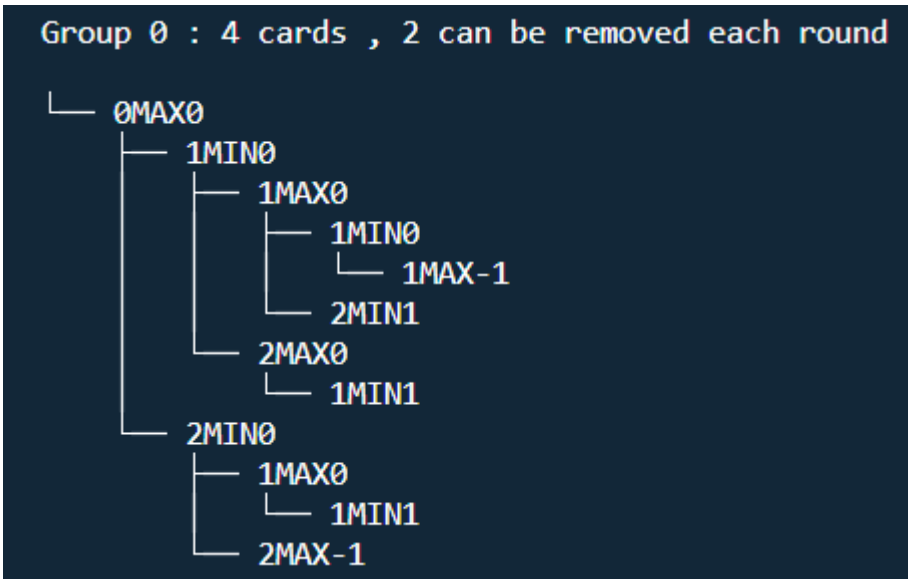
***ΑΛΛΙΩΣ** : score <- min(Scores)*

Ο αλγόριθμος είναι αναδρομικός. Ο τελευταίος κόμβος που θα ενημερωθεί είναι η ρίζα root.

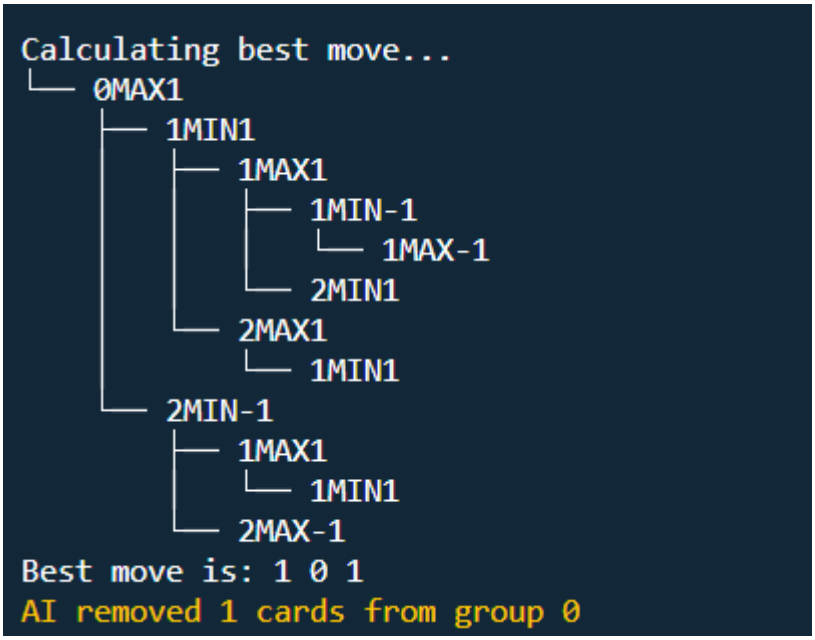
Στο παιχνίδι αυτό δεν υπάρχει κάποιο συγκεκριμένο score ανά γύρο. Τα μονά αποτελέσματα είναι το Botaki να νικήσει ή να χάσει. Επομένως ορίσαμε τις τιμές των score ως εξής :

- *Αν το τελευταίο χαρτί το αφαιρέσει ο MAX , τότε το Botaki νικάει και το score είναι +1.*
- *Διαφορετικά νικάει ο MIN και το score είναι -1.*

Για παράδειγμα , το αρχικό δένδρο αποφάσεων για μια ομάδα με 4 κάρτες , οπού μπορούμε να αφαιρέσουμε μέχρι 2 κάρτες ανά γύρο :



Οπού μετά την εφαρμογή του MiniMax έχει αυτή τη δομή :



Σε κάθε κόμβο αναγράφεται : αριθμός καρτών που αφαιρέθηκαν, MIN ή MAX και score.

Ο αλγόριθμος MiniMax με την χρήση **DFS** έχει πολυπλοκότητα χρόνου ***O(bm)*** *,* οπού b ο συντελεστής διακλάδωσης και m το μέγιστο βάθος.

Ο συντελεστής διακλάδωσης είναι το πλήθος των δυνατών κινήσεων σε κάθε γύρο , ενώ το μέγιστο βάθος θα είναι ίσο με τον αριθμό των χαρτιών στην αρχή του παιχνιδιού.

Για τον λόγο αυτό , αν δώσουμε πολλές κάρτες στην είσοδο (π.χ. 15 κάρτες και 4 ομάδες), το μέτωπο αναζήτησης είναι αρκετά μεγάλο και ο MiniMax είναι πιθανό να μην τερματίσει.

Ο αλγόριθμος MiniMax βρίσκει πάντα την βέλτιστη λύση , επομένως το Botaki διαλέγει την βέλτιστη κίνηση.

Για να μεταγλωττίσουμε τις κλασεις εκτελούμε την εντολή **javac *.java**

Για να τρέξουμε το πρόγραμμα εκτελούμε **java Table**

Συνίσταται να μην ξεπεράσετε τις 10 κάρτες