REPORT

LAB 2

ΧΑΣΑΝΗΣ ΕΥΑΓΓΕΛΟΣ cs05058

ΣΑΚΕΛΛΑΡΙΟΥ ΙΩΑΝΝΑ cs05125

ΜΟΥΣΕΛΙΜΗ ΑΜΑΛΙΑ – ΓΕΩΡΓΙΑ cs05074

Κατάσταση του παιγνίου είναι ο αριθμός των συνολικών καρτών(M), ο αριθμός των καρτών σε κάθε ομάδα(A) και ο επιτρεπτός αριθμός καρτών που μπορούν να αφαιρεθούν από κάθε ομάδα, από τον κάθε παίκτη, σε κάθε γύρο(B).

Οι ενέργειες σε κάθε μη-τελική κατάσταση εξαρτώνται από το B κάθε ομάδας, καθώς η αφαίρεση που μπορεί να γίνει πάνω σε κάθε ομάδα για κάθε γύρο, είναι ο αριθμός αυτός.

Εάν στην τελική κατάσταση πηγαίνουμε με κίνηση του MAX η αξία της είναι 1, ενώ αν πηγαίνουμε με κίνηση του MIN η αξία της είναι -1.

Για την class GameBoard.java η λειτουργία των μεθόδων εξηγείται στον κώδικα με σχόλια. Η χρήση της είναι σωστή αρχικοποίηση του παιχνιδιού(να είναι σωστά τα inputs του user και να τηρούν τους κανόνες του παιχνιδιού) και η αποθήκευση και ανάκτηση της τρέχουσας κατάστασης του παιχνιδιού.

Οι Human και Bot περιέχουν την μέθοδο remove η οποία τους δίνει τη δυνατότητα να αφαιρούν κάρτες.

Η team αρχικοποιεί την ομάδα και δίνει την δυνατότητα αφαίρεσης καρτών από αυτή.

Η class Node είναι απαραίτητη για τον σχηματισμό του GameTree καθώς κρατάει την πληροφορία για την αξία της κατάστασης, την ίδια την κατάσταση, την προηγούμενη αλλά και τις επόμενες καταστάσεις.

Επίσης με το isMaximizing γνωρίζουμε αν ο παίκτης που παίζει είναι ο MAX ή ο MIN που είναι απαραίτητο για την εκτέλεση του MINMAX στην πορεία.

Για να βρούμε κόμβο που είναι τελική κατάσταση χρησιμοποιούμε την

```
public boolean isTerminal()
```

Για να είναι τελιική μια κατάσταση πρέπει να έχουν αφαιρεθεί όλες οι κάρτες από όλες τις ομάδες, οπότε αυτό εξασφαλίζεται με έλεγχο κάθε ομάδας του τρέχοντος state

```
{

        for (int i = 0; i < state.size(); i++)
        {
            if (state.get(i) != 0)
            {
                return false;
            }
        }

        return true;
    }
```

Αν βρεθεί ομάδα με μη-μηδενικό αριθμό καρτών σημαίνει ότι δεν είναι τελική κατάσταση και όπως βλέπουμε παραπάνω πρέπει να επιστρέψουμε false.

Στην GameTree ξεκινάμε με το root που είναι η τρέχουσα κατάσταση και παράγουμε με το δέντρο παιχνιδιού με την generateTree(Node node).

Κάνουμε έναν αναδρομικό DFS με τον οποίο βάζουμε κάθε δυνατό συνδυασμό αφαίρεσης καρτών και αυτό οφείλεται στο B που έχει περιορισμό κάθε ομάδα.

```
for(int j = 1; j <= board.getB(i); j++)
```

Αν βρούμε ότι κάποιος κόμβος είναι τελική κατάσταση γυρνάμε -1 αν είναι isMaximizing γιατί αυτό σημαίνει ότι η κίνηση για να φτάσουμε σε αυτόν έγινε από !isMaximizing δηλαδή τον MIN.

Αντίστοιχα αν βρούμε ότι κάποιος κόμβος είναι τελική κατάσταση γυρνάμε 1 αν είναι !isMaximizing γιατί αυτό σημαίνει ότι η κίνηση για να φτάσουμε σε αυτόν έγινε από isMaximizing δηλαδή τον MAX.

Για την μέθοδο που εκτελεί τον minimax :

Είναι αναδρομικός και για αυτό παίρνουμε σε κάθε περίπτωση ξεχωριστά το bestValue.

Για Max : Αν κάποιο παιδί του έχει τιμή 1 σημαίνει ότι υπάρχει μονοπάτι που κερδίζει άρα το bestValue του πρέπει να ενημερωθεί σε 1.

```
 if(node.isMaximizing())
        {
            int bestValue = -1;
            for(Node child : node.getChildren())
            {
                int value = minimax(child);
                bestValue = Math.max(bestValue,
value);
            }
            node.setValue(bestValue);
            return bestValue;
        }
```

Για Μιν : Αν κάποιο παιδί του έχει τιμή -1 σημαίνει ότι υπάρχει μονοπάτι που χάνει άρα το bestValue του πρέπει να ενημερωθεί σε -1.

```
else
        {
            int bestValue = 1;
            for(Node child : node.getChildren())
```

```
            {
                int value = minimax(child);
                bestValue = Math.min(bestValue,
value);
            }
            node.setValue(bestValue);
            return bestValue;
        }
```

Για να γίνει η επιλογή της επόμενης κίνησης από το root χρησιμοποιούμε την :

```
public void pickChildToMakeMove()
```

Βρίσκει το πρώτο παιδι που εχει τιμή 1, δηλαδή έχει μονοπάτι που κερδίζει και καλεί την:

```
public void makeMove(Node rootChild)
```

στην οποία στην ουσία βρίσκουμε σε ποια ομάδα βρίσκονται οι κάρτες που πρέπει να αφαιρέσουμε και πόσες είναι αυτές, ώστε το τρέχον state να είναι ίδιο με το state που έχει αποθηκευμένο το παιδί που επιλέχθηκε.

Σε περίπτωση που ξεκινάει να παίζει ο παίκτης και όχι το bot υπάρχει πιθανότητα να έρθουμε αντιμέτωποι με σενάριο που τα παιδιά του root έχουν όλα τιμή -1.

Τότε καλείται η :

```
makeRiskyMove(root.getChild());
```

Η οποία αφαιρεί μόνο μια κάρτα από την ομάδα με τις περισσότερες με στόχο να υπάρξουν περισσότερες πιθανές μελλοντικές καταστάσεις άρα πιο πολλές πιθανότητες να βρεθούμε σε σενάριο που θα μας

επιστρέψει σε μονοπάτι που κερδίζει το bot αν ο παίκτης κάνει κάποια κίνηση που το επιτρέπει.

Να σημειωθεί βέβαια ότι υπάρχουν σενάρια που είναι μαθηματικά αδύνατο να κερδίσει το bot. Παράδειγμα :

Στην περίπτωση που έχουμε 4 κάρτες συνολικά, οι οποίες χωρίζονται σε δύο ομάδες και το Β και των δυο ομάδων είναι ίσο με 1, κερδίζει πάντα όποιος ξεκινάει να παίζει δεύτερος.

Cards(4)->bot(3) –> player(2) – >bot(1) –> player(0)

Cards(4)->player(3) –> bot(2) – >player(1) –> bot(0)

Η CardGame είναι η main class η οποία δημιουργεί το παιχνίδι, δίνει τη δυνατότητα επιλογής για το ποιος θα παίξει πρώτος και τυπώνει τον νικητή του παιχνιδιού.