**Τεχνητή Νοημοσύνη - Εργασία**

*Άσκηση 2*

Στην άσκηση 2 έχουμε να υλοποιήσουμε ένα card game. Όποιος βγάλει το τελευταίο φύλλο από την τελευταία ομάδα νικάει. Η υλοποίηση έχει πραγματοποιηθεί δημιουργώντας συνολικά 7 κλάσεις.

***Κλάσεις*** και σημαντικές *μέθοδοι***CardDeck**: **Η CardDeck έχει όλες τις στοίβες που έχουμε δημιουργήσει πάνω στο τραπέζι. Κρατάει πληροφορίες κυρίως για τις στοίβες χαρτιών.**

* removeCards(): Αφαιρεί κάρτες από κάποια στοίβα .
* addCards(): Προσθέτει κάρτες σε κάποια στοίβα.

copy(): Δημιουργεί ένα αντίγραφο του CardDeck και αντιγράφει το σύνολο των στοιβών, με τα φύλλα τους και το συνολικό πλήθος ομάδων που υπάρχουν στο τραπέζι.

**CardGroup**: **Η CardGroup κρατάει πληροφορίες για την κάθε στοίβα και συγκεκριμένα τα φύλλα της και ποιος είναι ο μέγιστος αριθμός φύλλων που μπορούμε να αφαιρέσουμε από μια στοίβα.**

* removeCards(): Αφαιρεί κάρτες από το group. Αν δεν μπορεί για κάποιο λογο να πραγματοποιηθεί η αφαίρεση βγάζει μηνύματα λάθους.
* addCards(): Προσθέτει κάρτες στο group.

**CardDealer**: **Η CardDealer είναι μια κλάση που "οργανώνει" το παιχνίδι. Είναι σαν ενας πραγματικός dealer στο casino, όπου μοιράζει τα φύλλα, δημιουργεί τις ομάδες και κανονίζει την σειρά που παίζει ο κάθε παίχτης.**

* createCardDeck(): Δημιουργεί την τράπουλα, με τα συνολικά φύλλα και τις ομάδες της.
* askPlayersMove(): Ρωτάει τον παίχτη ποια κίνηση θέλει να κάνει, δηλαδή ποια ομάδα να επιλέξει και πόσα φύλλα θα αφαιρέσει από αυτήν.
* removeCards(): Αφαιρεί κάρτες από μια στοίβα και ενημερώνει όλα τα πεδία και όλη την τράπουλα για τις αλλαγές που έχουν πραγματοποιηθεί.
* checkCards(): Ελέγχει αν εχουν αφαιρεθεί όλες οι κάρτες από κάποια ομάδα. Αν έχουν αφαιρεθεί όλες οι κάρτες, διαγραφέται και το group.

**Player**: **Εδώ αυτή η κλάση παίρνει απλά το όνομα του χρήστη που παίζει το παιχνίδι. Έχει εναν constructor που δημιουργεί το όνομα , μια μέθοδο που παίρνει το όνομα και μια μέθοδο που θέτει το όνομα.**

**Node**: **Η κλάση Node δημιουργεί το δέντρο που θα χρησιμοποιήσουμε για να υλοποιήσουμε τον min-max αλγόριθμο. Κάθε κόμβος έχει παιδιά, είτε δεν έχει παιδιά. Αν δεν έχει παιδιά σημαίνει ότι είναι τα φύλλα του δέντρου.**

* printTreeHelper(): Εδώ τυπώνει στην οθόνη όλο το δέντρο που έχει δημιουργηθεί με τον min-max algorithm.
* createChildren(): Δημιουργεί παιδιά σε κάθε κόμβο και αναλόγως το βάθος του δέντρου τα ονοματίζει max ή min.

**Botaki**: **Η κλάση Botaki υλοποιεί τον min-max αλγόριθμο**

* calculateBestMove(): Παίρνει την τρέχουσα τράπουλα και υπολογίζει την καλύτερη κίνηση που θα παίξει το Bot ενάντια στον παίχτη. Καλεί την findBestMoveWithMinimax() εμφωλευμένα.
* findBestMoveWithMinimax(): Βρίσκει την καλύτερη κίνηση με τον min-max αλγόριθμο.

**Table**: **Στο table πραγματοποιείται το παιχνίδι. Δημιουργεί αντικείμενα από τις κλάσεις Player, Botaki και CardDealer και στην main δημιουργεί το παιχνίδι.**

* calculateWinner():Υπολογίζει ποιος νίκησε. Είτε το Botaki, έιτε ο χρήστης.

**Min-max αλγόριθμος**:

Η ιδέα του min-max αλγόριθμου ήταν η εξής. Πρώτα δημιουργούμε όλο το δέντρο αποφάσεων, δηλαδή όλες τις δυνατές επιλογές που μπορούν να γίνουν στο παιχνίδι δεδομένης της αρχικής τράπουλας χρησιμοποιώντας BFS(Breadth First Search ). Ο κόμβος root, δηλαδή ο αρχικός, ορίζεται πάντα ως μαξ. Οι κόμβοι των επόμεων επιπέδων ορίζονται είτε min, είτε max εναλλάξ. Για κάθε τελικό κόμβο, δηλαδή κίνηση που αφαιρεί το τελευταίο χαρτί από το τραπέζι, δίνουμε μια τιμή score. Αν την τελευταία κίνηση την κάνει ο παίχτης(min), τότε το score είναι -1, δίοτι το Bot χάνει. Αν την τελευταία κίνηση την κάνει το Bot, το score είναι +1, διότι κερδίζει.