Τεχνητή Νοημοσύνη
Εαρινό Εξάμηνο 2023
Εργαστηριακή Άσκηση 2 (Κατασκευή Παιγνίου)

Ανδρεόπουλος Στάθης ΑΜ: 4630
Κοντάκης Σπύρος ΑΜ: 4702

1.Πως ορίζετε την αξία των τελικών καταστάσεων.

Οι τιμές των MAX και MIN μπορεί να διαφέρουν ανάλογα με το παιχνίδι που παίζεται και το σύστημα βαθμολόγησης ή τη βοηθητική λειτουργία που σχετίζεται με αυτό. Στον αλγόριθμο minimax, αυτές οι τιμές χρησιμοποιούνται για να αντιπροσωπεύουν τα καλύτερα και χειρότερα δυνατά αποτελέσματα για τους παίκτες.
Συνήθως, η τιμή MAX θα οριστεί σε μια υψηλή θετική τιμή, υποδεικνύοντας την καλύτερη δυνατή βαθμολογία ή χρησιμότητα που μπορεί να επιτύχει ο παίκτης MAX. Αντίθετα, η τιμή MIN θα οριστεί σε μια χαμηλή αρνητική τιμή, υποδεικνύοντας τη χειρότερη δυνατή βαθμολογία ή χρησιμότητα για το πρόγραμμα αναπαραγωγής MIN. Αυτές οι τιμές επιλέγονται για να διασφαλιστεί ότι τυχόν πραγματικές βαθμολογίες που λαμβάνονται κατά την αξιολόγηση των καταστάσεων του παιχνιδιού θα εμπίπτουν στο εύρος των τιμών MAX και MIN.

```
 8   #define M_Max 1000   // Maximum value for M
 9   #define MAX 1000    // Maximum value for MAX
10   #define MIN -1000   // Minimum value for MIN
```

Στον συγκεκριμένο κωδικό, οι τιμές 1000 και -1000 για MAX και MIN, αντίστοιχα, χρησιμοποιούνται ως σύμβολα κράτησης θέσης (Γραμμές 5-6).

2.Πως ορίζετε την αξία του παίγνιου.

Η δομή min_max ορίζει τη δομή κάθε κόμβου στο δέντρο του παιχνιδιού. Έχει τα ακόλουθα μέλη(Γραμμές 17-21):

```
17  {
18      struct min_max *kids[M_Max];        // Array of pointers to child nodes in the min-max tree
19      struct min_max *best_kid;           // Pointer to the next best move in the tree
20      int *N;                             // Array to store the number of cards each team has in the current state
21  };
```

a) kids: Μια σειρά δεικτών σε θυγατρικούς κόμβους. Αυτός ο πίνακας μπορεί να κρατήσει έως και M_Max αριθμό θυγατρικών κόμβων, αντιπροσωπεύοντας τις πιθανές μετακινήσεις από την τρέχουσα κατάσταση.

b) best_kid: Ένας δείκτης στον επόμενο κόμβο στο δέντρο του παιχνιδιού. Χρησιμοποιείται για να παρακολουθείτε τις καλύτερες κινήσεις σε κάθε επίπεδο.

c) N: Ένας δείκτης σε έναν πίνακα που αντιπροσωπεύει τον αριθμό των φύλλων σε κάθε ομάδα για την τρέχουσα κατάσταση.

Στον κώδικα, κάθε κόμβος στο δέντρο του παιχνιδιού κατανέμεται δυναμικά χρησιμοποιώντας malloc όταν είναι απαραίτητο. Για παράδειγμα, στη συνάρτηση min_max, δημιουργούνται νέοι κόμβοι χρησιμοποιώντας malloc σε αυτή τη γραμμή(170):

```
169                 // Allocate memory for the next state and its array
170                 tree->kids[level] = malloc(sizeof(struct min_max));
```

Ομοίως, η μνήμη εκχωρείται για τον πίνακα N σε κάθε κόμβο χρησιμοποιώντας malloc σε αυτή τη γραμμή(171):

```
171                     tree->kids[level]->N = malloc(K * sizeof(int));
```

Η παράμετρος δέντρου στη συνάρτηση min_max αντιπροσωπεύει τον τρέχοντα κόμβο στο δέντρο παιχνιδιού που αξιολογείται. Η συνάρτηση εξερευνά αναδρομικά το δέντρο του παιχνιδιού, δημιουργώντας και συνδέοντας θυγατρικούς κόμβους όπως απαιτείται. Κάθε κόμβος αντιπροσωπεύει μια πιθανή κατάσταση στο παιχνίδι και η δομή δέντρου χτίζεται με σύνδεση κόμβων με βάση τις πιθανές κινήσεις σε κάθε επίπεδο.

3. Περιγραφή του κώδικα που αναπτύξαμε.

Γραμμές 17-21 struct min_max

Η δομή min_max αποτελείται από τα ακόλουθα μέλη:

kids: Είναι ένας πίνακας δεικτών σε θυγατρικούς κόμβους στο δέντρο min_max. Το μέγεθος του πίνακα είναι M_Max, το οποίο ορίζεται ως σταθερά με μέγιστη τιμή 1000. Οι δείκτες σε αυτόν τον πίνακα αντιπροσωπεύουν τις πιθανές κινήσεις ή καταστάσεις που μπορούν να προσεγγιστούν από την τρέχουσα κατάσταση.

Best_kid: Είναι ένας δείκτης για την επόμενη καλύτερη κίνηση στο δέντρο. Αυτός ο δείκτης δείχνει τον θυγατρικό κόμβο που αντιστοιχεί στη βέλτιστη κίνηση στο δέντρο του παιχνιδιού.

N: Είναι ένας δείκτης σε έναν πίνακα που αποθηκεύει τον αριθμό των φύλλων που έχει κάθε ομάδα στην τρέχουσα κατάσταση. Το μέγεθος του πίνακα είναι K, το οποίο αντιπροσωπεύει τον συνολικό αριθμό των ομάδων.

Κάθε στοιχείο του πίνακα αντιπροσωπεύει τον αριθμό των καρτών για μια συγκεκριμένη ομάδα. Η δομή min_max χρησιμοποιείται για την αναπαράσταση κόμβων σε ένα δέντρο παιχνιδιού, όπου κάθε κόμβος αντιστοιχεί σε μια συγκεκριμένη κατάσταση παιχνιδιού. Οι δείκτες στη διάταξη δεικτών επιτρέπουν τη δημιουργία θυγατρικών κόμβων που αντιπροσωπεύουν διαφορετικές κινήσεις ή μεταβάσεις από την τρέχουσα κατάσταση. Ο επόμενος δείκτης χρησιμοποιείται για να παρακολουθείτε την καλύτερη κίνηση από την τρέχουσα κατάσταση. Ο πίνακας N αποθηκεύει τον αριθμό των φύλλων σε κάθε ομάδα για μια συγκεκριμένη κατάσταση παιχνιδιού.

```
17  {
18      struct min_max *kids[M_Max];        // Array of pointers to child nodes in the min-max tree
19      struct min_max *best_kid;           // Pointer to the next best move in the tree
20      int *N;                             // Array to store the number of cards each team has in the current state
21  };
```

Γραμμές 23-109 void initialization(void)

Η συνάρτηση void initialization(void) είναι υπεύθυνη για την προετοιμασία της κατάστασης του παιχνιδιού λαμβάνοντας τα δεδομένα του χρήστη για διάφορες παραμέτρους όπως ο αριθμός των φύλλων, ο αριθμός των ομάδων και ο αρχικός αριθμός των φύλλων για κάθε ομάδα. Ξεκινά δηλώνοντας κάποιες μεταβλητές και αρχικοποιώντας τη μεταβλητή αθροίσματος στο μηδέν. Η μεταβλητή sum θα χρησιμοποιηθεί για την παρακολούθηση του συνολικού αριθμού των καρτών που έχουν εκχωρηθεί στις ομάδες. Προτρέπει τον χρήστη να εισαγάγει τον αριθμό των καρτών (M) χρησιμοποιώντας έναν βρόχο do-while. Ο βρόχος συνεχίζεται μέχρι να δοθεί ένας θετικός αριθμός. Εάν η είσοδος δεν είναι θετική, εμφανίζεται ένα μήνυμα σφάλματος και ζητείται από τον χρήστη να προσπαθήσει ξανά. Ομοίως, η συνάρτηση προτρέπει τον χρήστη να εισαγάγει τον αριθμό των ομάδων (K) χρησιμοποιώντας έναν βρόχο do-while. Ο βρόχος συνεχίζεται μέχρι να δοθεί ένας θετικός αριθμός. Εάν η είσοδος δεν είναι θετική, εμφανίζεται ένα μήνυμα σφάλματος και ζητείται από τον χρήστη να προσπαθήσει ξανά. Επιπλέον, η συνάρτηση ελέγχει εάν ο αριθμός των ομάδων είναι μεγαλύτερος από τον αριθμό των φύλλων (M). Εάν είναι, εμφανίζεται ένα μήνυμα σφάλματος και ζητείται από τον χρήστη να προσπαθήσει ξανά. Στη συνέχεια, η συνάρτηση εκχωρεί δυναμικά τη μνήμη για τον πίνακα A χρησιμοποιώντας τη συνάρτηση malloc. Ο πίνακας A θα αποθηκεύσει τον αριθμό των φύλλων που έχει κάθε ομάδα. Το μέγεθος του πίνακα είναι K (αριθμός ομάδων) και κάθε στοιχείο θα είναι τύπου int. Ένας βρόχος χρησιμοποιείται για επανάληψη σε κάθε ομάδα, προτρέποντας τον χρήστη να εισαγάγει τον αριθμό των φύλλων (A[i]) για κάθε ομάδα χρησιμοποιώντας έναν βρόχο do-while. Ο βρόχος συνεχίζεται μέχρι να δοθεί ένας θετικός αριθμός. Εάν η είσοδος δεν είναι θετική, εμφανίζεται ένα μήνυμα σφάλματος και ζητείται από τον χρήστη να προσπαθήσει ξανά. Μετά από κάθε εισαγωγή, η μεταβλητή αθροίσματος ενημερώνεται προσθέτοντας τον αριθμό των φύλλων για την τρέχουσα ομάδα. Αυτό επιτρέπει στη συνάρτηση να ελέγχει εάν ο συνολικός αριθμός των φύλλων (M) είναι ίσος με το άθροισμα των φύλλων σε όλες τις ομάδες. Εάν δεν είναι ίσα, εμφανίζεται ένα μήνυμα σφάλματος και ζητείται από τον χρήστη να προσπαθήσει ξανά. Στη συνέχεια, η συνάρτηση εκχωρεί δυναμικά τη μνήμη για τον πίνακα B χρησιμοποιώντας τη συνάρτηση malloc. Ο πίνακας B θα αποθηκεύσει τον αριθμό των φύλλων που μπορούν να τραβήξουν από κάθε ομάδα. Το μέγεθος του πίνακα είναι K (αριθμός ομάδων) και κάθε στοιχείο θα είναι τύπου int. Ένας άλλος βρόχος χρησιμοποιείται για την επανάληψη σε κάθε ομάδα, προτρέποντας τον χρήστη να εισαγάγει τον αριθμό των φύλλων (B[i]) που μπορεί να τραβηχτεί από κάθε ομάδα χρησιμοποιώντας έναν βρόχο do-while. Ο βρόχος συνεχίζεται μέχρι να δοθεί ένας θετικός αριθμός και ελέγχει ότι ο αριθμός των φύλλων που μπορούν να τραβηχτούν δεν είναι μεγαλύτερος

από τον αριθμό των φύλλων που έχει η ομάδα (A[i]). Εάν η είσοδος δεν είναι έγκυρη, εμφανίζεται ένα μήνυμα σφάλματος και ζητείται από τον χρήστη να προσπαθήσει ξανά.

```c
void initialization(void)
{
    int i, j;
    int sum = 0;

    // Get the number of cards (M)
    do {
        printf("Give me the number of cards M: ");
        scanf("%d", &M);
        if (M <= 0) {
            printf("You need to give a positive number for cards (M).\n");
            printf("Try again!!!).\n");
            printf("\n");
        }
    } while (M <= 0);
    printf("\n");

    // Get the number of teams (K)
    do{
        do {
            printf("Give me the number of teams K: ");
            scanf("%d", &K);
            if (K <= 0) {
                printf("You need to give a positive number for the teams (K).\n");
                printf("Try again!!!).\n");
            }
        } while (K <= 0);
        if (M <= K) {
            printf("The number of teams(K) is higher than the number of cards.\n");
            printf("Try again!!!).\n");
            printf("\n");
        }
    }while(M <= K);
    printf("\n");

    // Allocate memory for the array A
    do {
        A = (int *)malloc(K * sizeof(struct min_max));
        for (i = 0; i < K; i++)
        {
            // Get the number of cards each team will have (A[i])
            do {
                j=i+1;
                printf("Give the number of cards each team will have A[%d]: ", j);
                scanf("%d", &A[i]);
                sum += A[i];
                if (A[i] <= 0) {
                    printf("You need to give a positive number for the possible number of cards a team can have (A[%d]).\n",j);
                    printf("Try again!!!).\n");
                    printf("\n");
                }
            } while (A[i] <= 0);
        }
        if (sum != M) {
            printf("The total number of cards (M) is different from the sum of the cards in the teams.\n");
            printf("Try again!!!).\n");
            printf("\n");
            sum = 0;
```

```
80              sum = 0;
81          }
82      } while (sum != M);
83      printf("\n");
84
85      // Allocate memory for the array B
86      do {
87          B = (int *)malloc(K * sizeof(int));
88          for (i = 0; i < K; i++) {
89              // Get the number of cards you can draw from each team (B[i])
90              do {
91                  j=i+1;
92                  printf("Give the number of cards you can draw from each team B[%d]: ", j);
93                  scanf("%d", &B[i]);
94                  if (B[i] <= 0) {
95                      printf("You need to give a positive number for the possible number of cards you can draw from a team (B[%d]).\n",j);
96                      printf("Try again!!!).\n");
97                      printf("\n");
98                      j=j-1;
99                  } else if (B[i] > A[i]) {
100                     printf("The number of cards you can draw from a team (B[%d]) cannot be greater than the number of cards the team has (A[%d]).\n", j
101                     printf("Try again!!!).\n");
102                     printf("\n");
103                 }
104             } while (B[i] <= 0 || B[i] > A[i]);
105
106         }
107     } while (i < K);
108     printf("\n");
109 }
```

Γραμμές 111-120 int find_Winner(const struct min_max *state)

Αυτή η συνάρτηση χρησιμοποιείται για να προσδιορίσει εάν έχει επιτευχθεί μια κατάσταση νίκης. Παίρνει έναν δείκτη στην τρέχουσα κατάσταση του παιχνιδιού (state) ως είσοδο. Υπολογίζει το άθροισμα του αριθμού των φύλλων σε κάθε ομάδα και ελέγχει αν είναι μηδέν. Εάν το άθροισμα είναι μηδέν, σημαίνει ότι όλες οι ομάδες έχουν μηδέν φύλλα, υποδεικνύοντας μια κατάσταση νίκης. Η συνάρτηση επιστρέφει 1 (true) εάν βρεθεί μια κατάσταση νίκης, διαφορετικά επιστρέφει 0 (false).

Γραμμές 122-128 void copy_table (const int X[K], int Y[K])

Αυτή η συνάρτηση χρησιμοποιείται για την αντιγραφή των περιεχομένων ενός πίνακα (X) σε έναν άλλο πίνακα (Y). Παίρνει δύο πίνακες ως είσοδο: X και Y μεγέθους K. Χρησιμοποιεί τη συνάρτηση memcpy από τη βιβλιοθήκη string.h για να εκτελέσει ένα byte-wise αντίγραφο της μνήμης από το X στο Y. Η συνάρτηση διασφαλίζει ότι τα στοιχεία του Y είναι ακριβές αντίγραφο των στοιχείων του X.

```
111 int find_Winner(const struct min_max *state)
112 {
113     // Check if all teams have zero cards
114     int sum = 0;
115     for (int i = 0; i < K; i++)
116     {
117         sum += state->N[i];  // Add the number of cards in each team to the sum
118     }
119     return (sum == 0) ? 1 : 0;  // If the sum is zero, all teams have zero cards and return 1 (true), otherwise return 0 (false)
120 }
121
122 void copy_table(const int X[K], int Y[K])
123 {
124     // Copy the contents of array X to array Y using memcpy
125     // The memcpy function copies the specified number of bytes from the source (X) to the destination (Y)
126     // The number of bytes to copy is calculated by multiplying the size of each element (sizeof(int)) by the number of elements (K)
127     memcpy(Y, X, K * sizeof(int));
128 }
```

Γραμμές 131-202 int MINIMAX (struct min_max *tree, int turn)

Είναι ο πυρήνας του αλγορίθμου min-max. Υπολογίζει αναδρομικά τη βέλτιστη κίνηση για τον παίκτη AI αξιολογώντας το δέντρο του παιχνιδιού. Χρειάζεται δύο παραμέτρους: έναν δείκτη στην τρέχουσα κατάσταση του δέντρου του παιχνιδιού (tree) και έναν ακέραιο που αντιπροσωπεύει τη σειρά του παίκτη (turn). Η μεταβλητή turn μπορεί να είναι είτε MAX είτε MIN, υποδεικνύοντας ποιος παίκτης είναι η σειρά. Καθορίζει πρώτα τον επόμενο παίκτη (αντίπαλο) εναλλάσσοντας τη μεταβλητή turn. Αρχικοποιεί μεταβλητές όπως**ΣΠΥΡΟΣ** η level ,που δηλώνει το επιπεδο(για την

αποθήκευση των κινήσεων που δημιουργούνται), ο πίνακας memory(για την αποθήκευση τιμών αξιολόγησης για κάθε κίνηση) και ο n (πίνακας για την αποθήκευση δεικτών στις παραγόμενες καταστάσεις). Ελέγχει εάν η τρέχουσα κατάσταση είναι μια κατάσταση που κερδίζει καλώντας τη συνάρτηση find_Winner. Εάν είναι μια κατάσταση νίκης, επιστρέφει μια τιμή (-1 ή 1) που αντιπροσωπεύει το αποτέλεσμα του παιχνιδιού για τον παίκτη AI (turn). Εάν η τρέχουσα κατάσταση δεν είναι νικηφόρα κατάσταση, η συνάρτηση προχωρά στη δημιουργία όλων των πιθανών κινήσεων και στην αξιολόγηση τους. Επαναλαμβάνει πάνω από κάθε ομάδα και ελέγχει εάν η ομάδα έχει απομείνει φύλλα (tree->N[i] > 0). Για κάθε ομάδα που έχει φύλλα που απομένουν, καθορίζει τον μέγιστο αριθμό φύλλων που μπορούν να τραβηχτούν από αυτήν την ομάδα (seen). Στη συνέχεια δημιουργεί όλες τις πιθανές κινήσεις για την τρέχουσα ομάδα, όπου κάθε κίνηση περιλαμβάνει το τράβηγμα ενός συγκεκριμένου αριθμού καρτών από αυτήν την ομάδα. Για κάθε πιθανή κίνηση, εκχωρεί μνήμη για την επόμενη κατάσταση και τον πίνακα της, δημιουργεί ένα αντίγραφο της τρέχουσας κατάστασης, την ενημερώνει με την κίνηση και καλεί αναδρομικά τη συνάρτηση MINIMAX στην επόμενη κατάσταση. Η τιμή αξιολόγησης κάθε κίνησης αποθηκεύεται στον πίνακα δοκιμής και ο δείκτης στην αντίστοιχη κατάσταση αποθηκεύεται στον πίνακα n. Αφού αξιολογήσει όλες τις πιθανές κινήσεις, καθορίζει την καλύτερη κίνηση με βάση την αξία αξιολόγησης και τη σειρά του παίκτη. Τέλος, ενημερώνει το δέντρο με την καλύτερη κίνηση εκχωρώντας τον επόμενο δείκτη της τρέχουσας κατάστασης στην κατάσταση που αντιπροσωπεύει την καλύτερη κίνηση. Επιστρέφει επίσης την αξία αξιολόγησης της καλύτερης κίνησης.

```
131  int MINIMAX(struct min_max *tree, int turn)
132  {
133      // Recursive function to compute the optimal move using the min-max algorithm
134
135      // Determine the next player (opponent)
136      int next_player = (turn == MAX) ? MIN : MAX;
137
138      // Position counter for storing generated moves
139      int level = 0;
140
141      // Array to store evaluation values for each move
142      int memory[K];
143
144      // Array to store pointers to generated states
145      struct min_max *n[K];
146
147      // Best evaluation value and pointer to the best state
148      int best_node;
149      struct min_max *best_state;
150
151      // Maximum number of cards that can be drawn from a team
152      int seen;
153
154      // Base case: check if the current state is a winning state
155      if (find_Winner(tree))
156          return (turn == MAX) ? -1 : 1;  // Return -1 for MAX player win, 1 for MIN player win
157
158      // Generate all possible moves and evaluate them
159      for (int i = 0; i < K; i++)
160      {
```

```
        if (tree->N[i] > 0)
        {
            // Determine the maximum number of cards that can be drawn from the team
            seen = (B[i] < tree->N[i]) ? B[i] : tree->N[i];

            // Generate all possible moves for the current team
            for (int j = 1; j <= seen; j++)
            {
                // Allocate memory for the next state and its array
                tree->kids[level] = malloc(sizeof(struct min_max));
                tree->kids[level]->N = malloc(K * sizeof(int));

                // Create a copy of the current state and update it with the move
                copy_table(tree->N, tree->kids[level]->N);
                int decrement = j;
                tree->kids[level]->N[i] -= decrement;

                // Recursively evaluate the subtree
                memory[level] = MINIMAX(tree->kids[level], next_player);
                n[level] = tree->kids[level];
                level++;
            }
        }
    }

    // Determine the best move based on the evaluation
    best_node = memory[0];
    best_state = n[0];

    for (int i = 1; i < level; i++)
    {
        if ((turn == MAX && memory[i] > best_node) || (turn == MIN && memory[i] < best_node))
        {
            best_node = memory[i];
            best_state = n[i];
        }
    }

    // Update the tree with the best move and return the best move
    tree->best_kid = best_state;
    return best_node;
}
```

Γραμμές 205-285 main()

Είναι το σημείο εισόδου του προγράμματος και χρησιμεύει ως οδηγός για το παιχνίδι. Ξεκινά με την εκχώρηση μνήμης για την αρχική κατάσταση του παιχνιδιού χρησιμοποιώντας malloc. Καλεί τη συνάρτηση initialization για να λάβει στοιχεία από τον χρήστη για τον αριθμό των φύλλων, τον αριθμό των ομάδων, τον αριθμό των φύλλων που έχει κάθε ομάδα (A) και τον αριθμό των φύλλων που μπορούν να τραβήξουν από κάθε ομάδα (B). Η αρχική κατάσταση παιχνιδιού ορίζεται με την εκχώρηση μνήμης για τον πίνακα N σε κατάσταση και την αντιγραφή των περιεχομένων του πίνακα A στο N χρησιμοποιώντας τη συνάρτηση copy_table. Ξεκινά ο βρόχος παιχνιδιού, ο οποίος συνεχίζεται μέχρι να επιτευχθεί η κατάσταση νίκης. Εντός του βρόχου παιχνιδιού, πρώτα ελέγχει εάν ο τρέχων παίκτης (AI ή παίκτης) έχει κερδίσει καλώντας τη συνάρτηση find_Winner. Εάν βρεθεί μια κατάσταση νίκης, εκτυπώνει το κατάλληλο μήνυμα νίκης και βγαίνει από τον βρόχο του παιχνιδιού. Εάν δεν βρεθεί κατάσταση νίκης, προχωρά είτε στο γύρο του AI είτε στο γύρο του παίκτη με βάση την τιμή της μεταβλητής turn. Για τη σειρά του AI (turn == MAX), καλεί τη συνάρτηση MINIMAX για να υπολογίσει τη βέλτιστη κίνηση για το AI. Η κατάσταση που προκύπτει ενημερώνεται εκχωρώντας κατάσταση σε state-> best_kid. Για τη σειρά του παίκτη (turn == MIN), ζητά από τον παίκτη να επιλέξει μια ομάδα από την οποία θα τραβήξει φύλλα και τον αριθμό των φύλλων που θα τραβήξει. Η επιλεγμένη ομάδα και ο αριθμός των καρτών

ενημερώνονται στην κατάσταση του παιχνιδιού. Μετά από κάθε σειρά, εκτυπώνει την τρέχουσα κατάσταση κάθε ομάδας. Τέλος, μόλις σπάσει ο βρόχος του παιχνιδιού, εκτυπώνει το μήνυμα νίκης με βάση τον παίκτη που κέρδισε το παιχνίδι (AI ή παίκτης).

```c
int main()
{
    // Initialize the game state
    struct min_max *state = malloc(sizeof(struct min_max));
    int turn = MAX, team, number, i, j;
    int temp;
    initialization();

    state->N = malloc(K * sizeof(int));
    copy_table(A, state->N);

    printf("---Game Starts---\n");
    for (i = 0; i < K; i++)
    {
        j=i+1;
        printf("team %d : %d cards\n", j, state->N[i]);
    }
    printf("\n");

    // Game loop
    while (1)
    {
        // Check if the current player has won
        if (turn == MAX && find_Winner(state) == 1)
        {
            printf("---The Player(MIN) wins---\n");
            break;
        }
        else if (turn == MIN && find_Winner(state) == 1)
        {
            printf("---The AI(MAX) wins---\n");
            break;
        }
        else if (turn == MAX)
        {
            // AI's turn
            MINIMAX(state, MAX);
            state = state->best_kid;
            printf("AI's turn\n");
            for (i = 0; i < K; i++)
            {
                j=i+1;
                printf("team %d : %d cards\n", j, state->N[i]);
            }
            printf("\n");
            turn = MIN;
        }
        else
        {
            // Player's turn
            printf("Player's turn\n");
            printf("Time to choose\n");

            // Print the teams that still have cards
            for (i = 0; i < K; i++)
            {
                j=i+1;
                if (state->N[i] > 0)
                {
                    printf("Press %d to choose team %d\n", j, j);
```

```
265                 }
266             }
267             scanf("%d", &team); // Get the chosen team
268             temp = team-1;
269             printf("Choose the number of cards you will draw\n");
270             int maxNumber = (B[temp] < state->N[temp]) ? B[temp] : state->N[temp];
271             printf("Choose from 1 to %d\n", maxNumber);
272             scanf("%d", &number); // Get the number of cards to draw
273 
274             state->N[temp] -= number; // Update the number of cards in the chosen team
275             printf("Player's turn\n");
276             for (i = 0; i < K; i++)
277             {
278                 j=i+1;
279                 printf("team %d : %d cards\n", j, state->N[i]); // Print the updated number of cards in each team
280             }
281             printf("\n");
282             turn = MAX; // Switch to AI's turn
283         }
284     }
285 }
```

4. Παράδειγμα εκτέλεσης του κώδικά μας.

Με bold είναι οι τιμές που βάζουμε ώστε να τρέξει το πρόγραμμα

stathis@LAPTOP-E6ESFTS7:/mnt/c/Users/Stathis/Desktop/aia2$ ./a.out
Give me the number of cards M: **10**

Give me the number of teams K: **3**

Give the number of cards each team will have A[1]: **5**
Give the number of cards each team will have A[2]: **3**
Give the number of cards each team will have A[3]: **2**

Give the number of cards you can draw from each team B[1]: **3**
Give the number of cards you can draw from each team B[2]: **2**
Give the number of cards you can draw from each team B[3]: **1**

---Game Starts---
team 1 : 5 cards
team 2 : 3 cards
team 3 : 2 cards

AI's turn
team 1 : 4 cards
team 2 : 3 cards
team 3 : 2 cards

Player's turn
Time to choose
Press 1 to choose team 1
Press 2 to choose team 2
Press 3 to choose team 3
1

Choose the number of cards you will draw
Choose from 1 to 3
3
Player's turn
team 1 : 1 cards
team 2 : 3 cards
team 3 : 2 cards

AI's turn
team 1 : 0 cards
team 2 : 3 cards
team 3 : 2 cards

Player's turn
Time to choose
Press 2 to choose team 2
Press 3 to choose team 3
3
Choose the number of cards you will draw
Choose from 1 to 1
1
Player's turn
team 1 : 0 cards
team 2 : 3 cards
team 3 : 1 cards

AI's turn
team 1 : 0 cards
team 2 : 1 cards
team 3 : 1 cards

Player's turn
Time to choose
Press 2 to choose team 2
Press 3 to choose team 3
2
Choose the number of cards you will draw
Choose from 1 to 1
1
Player's turn
team 1 : 0 cards
team 2 : 0 cards
team 3 : 1 cards

AI's turn
team 1 : 0 cards
team 2 : 0 cards
team 3 : 0 cards

---The AI(MAX) wins---