```c
/* O sistema de ajuste de apostas utilizado neste projecto segue as instruções
 * explicadas aqui: http://casinogambling.about.com/od/blackjack/a/hilo.htm
 */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdbool.h>
#include <math.h>
#include "error.h"
#include "logic.h"
#include "ea.h"

/*
 * O formato do ficheiro de estratégia das EAs é:
 * 10 (HARD_LINES) linhas com 10 caracteres cada para as decisões Hard
 * uma linha em branco (um \n)
 * 8 (SOFT_LINES) linhas com 10 caracteres cada para as decisões Soft
 * os caracteres podem ser:
 *  H - hit
 *  S - stand
 *  R - surrender
 *  D - double senão hit
 *  E - double senão stand
 */

void write_matrix(Move ***matrix, FILE *file, int lines)
{
    char buffer[COLUMNS+2] = {0}; // COLUMNS + '\n' e '\0'
    *matrix = (Move **) ecalloc(lines, sizeof(Move *));

    for (int i = 0; i < lines; i++) {
        fgets(buffer, COLUMNS+2, file);
        // Se o caracter exatamente após COLUMNS colunas da linha não for \n,
        // sabemos que a linha não tem exatamente COLUMNS caracteres.
        if (buffer[COLUMNS] != '\n') {
            fprintf(stderr, "Erro: Ficheiro de estratégia das EAs mal formatado.\n");
            exit(EXIT_FAILURE);
        }
        (*matrix)[i] = (Move *) ecalloc(10, sizeof(Move));
         for (int j = 0; j < COLUMNS; j++)
            (*matrix)[i][j] =  buffer[j];
    }
}

void destroy_matrix(Move **matrix, int lines)
{
    for (int i = 0; i < lines; i++)
        free(matrix[i]);

    free(matrix);
}


// Ler a estrategia do ficheiro de configuração
Strategy *read_strategy(char *filename)
{
    char check[2] = {0}; // \n e \0
    FILE *config_file = efopen(filename, "r");

    Strategy *strategy = (Strategy *) ecalloc(1, sizeof(Strategy));

    strategy->hard = NULL;
    strategy->soft = NULL;

    write_matrix(&strategy->hard, config_file, HARD_LINES);

    // Verificar \n de separação das matrizes
    fgets(check, 2, config_file);
```

```
70      if (check[0] != '\n') {
71          fprintf(stderr, "Erro: Ficheiro de estratégia das EAs mal formatado.\n
    ");
72          exit(EXIT_FAILURE);
73      }
74
75      write_matrix(&strategy->soft, config_file, SOFT_LINES);
76
77      fclose(config_file);
78
79      return strategy;
80  }
81
82  /*
83   * Decide que matriz utilizar: hard ou soft
84   * Calcula a coluna e linha da matriz
85   * Retorna a decisão a tomar
86   */
87  Move get_decision(Player *player, Card *house_card, Strategy *strategy)
88  {
89      bool ace = false;
90      int line = 0, column = 0;
91      Stack *aux = player->cards;
92
93      //Verificar se a ases
94      while(aux) {
95          if (aux->card->id == 12)
96              ace = true;
97          aux = aux->next;
98      }
99
100     // Calcular a coluna da matriz
101     if (house_card->id > 0 && house_card->id < 8)
102         column = house_card->id;
103     else if (house_card->id >=8 && house_card->id < 12)
104         column = 8;
105     else if (house_card->id == 12)
106         column = 9;
107
108     // Calcular a linha da matriz soft
109     if (ace) {
110         // Dois ases correspondem à primeira linha da matriz soft
111         if (player->points == 12)
112             line = 0;
113         else if (player->points > 12 && player->points < 19)
114             line = player->points - 12;
115         else if (player->points >= 19)
116             line = 7;
117
118         return strategy->soft[line][column];
119     }
120     // Calcular a coluna da matriz hard
121     else {
122         if (player->points >= 4 && player->points <= 8)
123             line = 0;
124         else if (player->points > 8 && player->points < 17)
125             line = player->points - 8;
126         else if (player->points >= 17)
127             line = 9;
128
129         return strategy->hard[line][column];
130     }
131 }
132
133
134 /*
135  * Encontra proximo jogador
136  * Usa o valor de retorno de get_decision para escolher a proxima ação
137  */
138 void ea_make_decision(List *players, Player *house, Megadeck *megadeck, Strate
```

```
gy *strategy)
139 {
140     bool can_double = false;
141     List *aux = find_active_player(players);
142     Player *cur_player = (Player *) aux->payload;
143 
144     Card *house_card = house->cards->next->card;
145     Move decision = get_decision(cur_player, house_card, strategy);
146 
147     switch (decision) {
148         case H:
149             player_hit(players, house, megadeck);
150             break;
151 
152         case S:
153             stand(players, house, megadeck);
154             break;
155 
156         case R:
157             surrender(players, house, megadeck);
158             break;
159 
160 
161         case D:
162             can_double = double_bet(players, house, megadeck);
163             if (!can_double) {
164                 player_hit(players, house, megadeck);
165             }
166             break;
167         case E:
168             can_double = double_bet(players, house, megadeck);
169             if (!can_double) {
170                 stand(players, house, megadeck);
171             }
172             break;
173 
174         default:
175             // Isto nunca deverá acontecer
176             fprintf(stderr, "Erro: Decisão de EA inesperada.\n");
177             exit(EXIT_FAILURE);
178             break;
179     }
180 }
181 
182 
183 /*
184  * Cada jogador começa com a contagem = 0
185  * Em cada ronda são contadas as cartas e no fim da ronda ao count dos jogador
es
186  * é somada a contagem da ronda
187  */
188 
189 //Conta cartas segundo a estrategia hi-lo
190 void count_cards(Card *new_card, Megadeck *megadeck)
191 {
192     if (new_card->id < 5)
193         megadeck->round_count++;
194     else if (new_card->id > 7)
195         megadeck->round_count--;
196 }
197 
198 // soma player->count com megadeck->round_count
199 void update_count(List *players, Megadeck *megadeck)
200 {
201     List *aux = players->next;
202     Player *cur_player = NULL;
203 
204     while (aux) {
205         cur_player = (Player *) aux->payload;
206         if (cur_player->type == EA)
```

```
207                 cur_player->count += megadeck->round_count;
208
209             aux = aux->next;
210         }
211
212         megadeck->round_count = 0;
213     }
214
215     /* Altera a bet do jogador
216      * A aposta original do jogador é uma unidade
217      * true_count = contagem / numero de baralhos
218      * A nova aposta no jogador é igual a 2*true_count unidades
219      * Se true_count <= 0 a nova aposta e igual a 1 unidades
220      */
221     void hi_lo(Player *player, Megadeck *megadeck)
222     {
223         double new_bet = 0;
224         double decks_left = round(((double) megadeck->cards_left)/DECK_SIZE + 1);
225         int true_count = round(player->count/decks_left);
226
227         if (true_count <= 0)
228             new_bet = player->orig_bet;
229         else
230             new_bet = 2 * true_count * player->orig_bet;
231
232
233         if (player->type == EA) {
234             if (player->money > new_bet)
235                 player->bet = new_bet;
236             else
237                 player->bet = player->money;
238         }
239     }
```