```c
/*
 * Projecto de Programação
 * Implementação em C do jogo de casino Blackjack
 * Autores:
 * João Pinheiro <joao.castro.pinheiro@tecnico.ulisboa.pt>
 * João Freitas  <joao.m.freitas@tecnico.ulisboa.pt>
 */

#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <stdbool.h>
#include <SDL2/SDL.h>
#include <SDL2/SDL_ttf.h>
#include <SDL2/SDL_image.h>
#include <time.h>
#include "main.h"
#include "logic.h"
#include "file.h"
#include "sdl.h"
#include "ea.h"
#include "error.h"

int main(int argc, char *argv[])
{
    SDL_Window *window = NULL;
    SDL_Renderer *renderer = NULL;
    TTF_Font *serif = NULL;
    SDL_Surface *cards[MAX_DECK_SIZE+1] = {0};
    SDL_Surface *imgs[2] = {0};
    SDL_Event event;
    int delay = 300;
    int ea_delay = 1500;
    bool quit = false;
    bool add_player_key = false;
    AddPlayerError add_player_error = OK;

    if (argc != 3) {
        fprintf(stderr, "Erro: número inválido de argumentos.\n");
        puts("Utilização:");
        printf("%s <ficheiro de config. do jogo> <ficheiro de config. das EAs>
\n", argv[0]);
        exit(EXIT_FAILURE);
    }

    // Ler ficheiro de configuração dos jogadores
    Config *config = read_config(argv[1]);

    // Ler ficheiro de estrategia das EAs
    Strategy *strategy = read_strategy(argv[2]);

    // Declarar a lista de jogadores
    List *players = (List *) ecalloc((size_t) 1, sizeof(List));
    // enchê-la com dados do ficheiro de configuração
    const int num_decks = init_game(config, players);

    // Declarar a lista de jogadores velhos
    List *old_players = (List *) ecalloc((size_t) 1, sizeof(List));

    // Inicializar o megabaralho
    // é uma struct que contém a lista das cartas em si
    int cards_left = 0;
    List *deck = (List *) ecalloc((size_t) 1, sizeof(List));
    Megadeck megadeck_real = {cards_left, num_decks, deck, 0};
    Megadeck *megadeck = &megadeck_real;
    megadeck->cards_left = create_megadeck(megadeck);

    // Inicializar a casa
    Player *house = (Player *) ecalloc((size_t) 1, sizeof(Player));
    // Inicializar um novo jogo
```

```
70      srand(time(NULL));
71      new_game(players, house, megadeck);
72
73      // loads the cards images
74      LoadCards(cards);
75
76      // initialize graphics
77      InitEverything(WIDTH_WINDOW, HEIGHT_WINDOW, &serif, imgs, &window, &render
    er);
78
79      List *aux = find_active_player(players);
80      bool ea = false;
81      while (!quit) {
82          // while there's events to handle
83          while (SDL_PollEvent(&event)) {
84              if (event.type == SDL_QUIT) {
85                  // user killed the window
86                  quit = true;
87              }
88              else if (event.type == SDL_KEYDOWN) {
89                  switch (event.key.keysym.sym) {
90                      case SDLK_q:
91                          if (find_active_player(players) == NULL ||
92                              find_ingame_player(players) == NULL)
93                              quit = true;
94                          break;
95
96                      case SDLK_n:
97                          new_game(players, house, megadeck);
98                          break;
99
100                     case SDLK_r:
101                         if (find_active_human_player(players) != NULL)
102                             surrender(players, house, megadeck);
103                         break;
104
105                     case SDLK_d:
106                         if (find_active_human_player(players) != NULL)
107                             double_bet(players, house, megadeck);
108                         break;
109
110                     case SDLK_b:
111                         bet(players);
112                         break;
113
114                     case SDLK_a:
115                         if (find_active_player(players) == NULL)
116                             // sinalizar para mostrar as popups
117                             add_player_key = true;
118                         break;
119
120                     case SDLK_s:
121                         if (find_active_human_player(players) != NULL)
122                             stand(players, house, megadeck);
123                         break;
124
125                     case SDLK_h:
126                         if (find_active_human_player(players) != NULL)
127                             player_hit(players, house, megadeck);
128                         break;
129
130                     case SDLK_UP:
131                         ea_delay+=100;
132                         break;
133
134                     case SDLK_DOWN:
135                         if (ea_delay > 100)
136                             ea_delay-=100;
137                         break;
138
```

```
                    default:
                        break;
                }
            }
        }

        if (add_player_key)
            show_add_player_message(window);
        // render game table
        RenderTable(players, serif, imgs, renderer);
        // render house cards
        RenderHouseCards(house, cards, serif, renderer);
        // render player cards
        RenderPlayerCards(players, cards, renderer);
        // render colorful status rects above player
        render_status(players, serif, renderer);
        // render in the screen all changes above
        SDL_RenderPresent(renderer);
        // add a delay
        SDL_Delay(delay);

        aux = find_active_player(players);
        ea = false;
        if (aux != NULL)
            if (((Player * ) aux->payload)->type == EA)
                ea = true;
        // Se for a vez dum EA, decidir a sua jogada, com delay
        if (ea) {
            ea_make_decision(players, house, megadeck, strategy);
            SDL_Delay(ea_delay);
        }

        else if (add_player_key) {
            add_player_error = add_player(players, old_players, megadeck, wind
ow);
            if (add_player_error != OK)
                // mostrar popup de erro se este existir
                show_add_player_error_message(window, add_player_error);
            add_player_key = false;
        }
    }

    write_stats(players, house, old_players);

    destroy_players_list(players);
    destroy_players_list(old_players);
    destroy_stack(&house->cards);
    free(house);
    destroy_list(megadeck->deck);
    destroy_matrix(strategy->hard, HARD_LINES);
    destroy_matrix(strategy->soft, SOFT_LINES);
    free(strategy);

    UnLoadCards(cards);
    TTF_CloseFont(serif);
    SDL_FreeSurface(imgs[0]);
    SDL_FreeSurface(imgs[1]);
    SDL_DestroyRenderer(renderer);
    SDL_DestroyWindow(window);
    SDL_Quit();

    return EXIT_SUCCESS;
}
```