Instituto Superior Técnico

Engenharia Eletrotécnica e de Computadores

Programação

Blackjack

Projecto final de Programação

Docente: João Ascenso

Autores:

João Pinheiro	84086	joao.castro.pinheiro@tecnico.ulisboa.pt
João Freitas	84093	joao.m.freitas@tecnico.ulisboa.pt

Grupo 25

2015/2016 - Primeiro Ano - Segundo Semestre
Maio de 2016

```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
```

```
    srand(time(NULL));
    new_game(players, house, megadeck);

    // loads the cards images
    LoadCards(cards);

    // initialize graphics
    InitEverything(WIDTH_WINDOW, HEIGHT_WINDOW, &serif, imgs, &window, &renderer);

    List *aux = find_active_player(players);
    bool ea = false;
    while (!quit) {
        // while there's events to handle
        while (SDL_PollEvent(&event)) {
            if (event.type == SDL_QUIT) {
                // user killed the window
                quit = true;
            }
            else if (event.type == SDL_KEYDOWN) {
                switch (event.key.keysym.sym) {
                    case SDLK_q:
                        if (find_active_player(players) == NULL ||
                            find_ingame_player(players) == NULL)
                            quit = true;
                        break;

                    case SDLK_n:
                        new_game(players, house, megadeck);
                        break;

                    case SDLK_r:
                        if (find_active_human_player(players) != NULL)
                            surrender(players, house, megadeck);
                        break;

                    case SDLK_d:
                        if (find_active_human_player(players) != NULL)
                            double_bet(players, house, megadeck);
                        break;

                    case SDLK_b:
                        bet(players);
                        break;

                    case SDLK_a:
                        if (find_active_player(players) == NULL)
                            // sinalizar para mostrar as popups
                            add_player_key = true;
                        break;

                    case SDLK_s:
                        if (find_active_human_player(players) != NULL)
                            stand(players, house, megadeck);
                        break;

                    case SDLK_h:
                        if (find_active_human_player(players) != NULL)
                            player_hit(players, house, megadeck);
                        break;

                    case SDLK_UP:
                        ea_delay+=100;
                        break;

                    case SDLK_DOWN:
                        if (ea_delay > 100)
                            ea_delay-=100;
                        break;

```

```
                    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;
}
```

```
1  #ifndef MAIN_H
2  #define MAIN_H
3
4  #define MAX_DECK_SIZE 52       // number of max cards in the deck
5  #define MAX_CARD_HAND 11       // 11 cards max. that each player can hold
6  #define MAX_PLAYERS 4          // number of maximum players
7  #define MAX_STR_SIZE 100
8
9  #endif
```

```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);
```

```
    if (check[0] != '\n') {
        fprintf(stderr, "Erro: Ficheiro de estratégia das EAs mal formatado.\n
");
        exit(EXIT_FAILURE);
    }

    write_matrix(&strategy->soft, config_file, SOFT_LINES);

    fclose(config_file);

    return strategy;
}

/*
 * Decide que matriz utilizar: hard ou soft
 * Calcula a coluna e linha da matriz
 * Retorna a decisão a tomar
 */
Move get_decision(Player *player, Card *house_card, Strategy *strategy)
{
    bool ace = false;
    int line = 0, column = 0;
    Stack *aux = player->cards;

    //Verificar se a ases
    while(aux) {
        if (aux->card->id == 12)
            ace = true;
        aux = aux->next;
    }

    // Calcular a coluna da matriz
    if (house_card->id > 0 && house_card->id < 8)
        column = house_card->id;
    else if (house_card->id >=8 && house_card->id < 12)
        column = 8;
    else if (house_card->id == 12)
        column = 9;

    // Calcular a linha da matriz soft
    if (ace) {
        // Dois ases correspondem à primeira linha da matriz soft
        if (player->points == 12)
            line = 0;
        else if (player->points > 12 && player->points < 19)
            line = player->points - 12;
        else if (player->points >= 19)
            line = 7;

        return strategy->soft[line][column];
    }
    // Calcular a coluna da matriz hard
    else {
        if (player->points >= 4 && player->points <= 8)
            line = 0;
        else if (player->points > 8 && player->points < 17)
            line = player->points - 8;
        else if (player->points >= 17)
            line = 9;

        return strategy->hard[line][column];
    }
}


/*
 * Encontra proximo jogador
 * Usa o valor de retorno de get_decision para escolher a proxima ação
 */
void ea_make_decision(List *players, Player *house, Megadeck *megadeck, Strate
```

```
gy *strategy)
{
    bool can_double = false;
    List *aux = find_active_player(players);
    Player *cur_player = (Player *) aux->payload;

    Card *house_card = house->cards->next->card;
    Move decision = get_decision(cur_player, house_card, strategy);

    switch (decision) {
        case H:
            player_hit(players, house, megadeck);
            break;

        case S:
            stand(players, house, megadeck);
            break;

        case R:
            surrender(players, house, megadeck);
            break;


        case D:
            can_double = double_bet(players, house, megadeck);
            if (!can_double) {
                player_hit(players, house, megadeck);
            }
            break;
        case E:
            can_double = double_bet(players, house, megadeck);
            if (!can_double) {
                stand(players, house, megadeck);
            }
            break;

        default:
            // Isto nunca deverá acontecer
            fprintf(stderr, "Erro: Decisão de EA inesperada.\n");
            exit(EXIT_FAILURE);
            break;
    }
}


/*
 * Cada jogador começa com a contagem = 0
 * Em cada ronda são contadas as cartas e no fim da ronda ao count dos jogador
es
 * é somada a contagem da ronda
 */

//Conta cartas segundo a estrategia hi-lo
void count_cards(Card *new_card, Megadeck *megadeck)
{
    if (new_card->id < 5)
        megadeck->round_count++;
    else if (new_card->id > 7)
        megadeck->round_count--;
}

// soma player->count com megadeck->round_count
void update_count(List *players, Megadeck *megadeck)
{
    List *aux = players->next;
    Player *cur_player = NULL;

    while (aux) {
        cur_player = (Player *) aux->payload;
        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     }
```

```c
#ifndef EA_H
#define EA_H

#include <stdbool.h>
#include "logic.h"

#define COLUMNS 10
#define HARD_LINES 10
#define SOFT_LINES 8
#define HI_LO // opção que, se definida, ativa a estrátegia hi-lo das EAs

void write_matrix(Move ***matrix, FILE *file, int lines);
void destroy_matrix(Move **matrix, int lines);

Strategy *read_strategy(char *filename);
Move get_decision(Player *player, Card *house_card, Strategy *strategy);
void ea_make_decision(List *players, Player *house, Megadeck *megadeck, Strategy *strategy);

void count_cards(Card *new_card, Megadeck *megadeck);
void update_count(List *players, Megadeck *megadeck);
void hi_lo(Player *player, Megadeck *megadeck);

#endif
```

```c
#include <stdlib.h>
#include <stdio.h>

/*
 * Wrappers para o fopen() e o calloc() que tratam de erros automaticamente.
 */
void *ecalloc(size_t nmemb, size_t size)
{
    void *memory = calloc(nmemb, size);

    if (memory == NULL) {
        fprintf(stderr, "Erro: Impossível alocar memória.\n");
        exit(EXIT_FAILURE);
    }

    return memory;
}

FILE *efopen(const char *path, const char *mode)
{
    FILE *file = fopen(path, mode);

    if (file == NULL) {
        fprintf(stderr, "Erro: Impossível abrir ficheiro %s.\n", path);
        exit(EXIT_FAILURE);
    }

    return file;
}
```

```c
#include <stdio.h>

void *ecalloc(size_t nmemb, size_t size);
FILE *efopen(const char *path, const char *mode);
```

```c
/*
 * Leitura e escrita de ficheiros
 * Lê o ficheiro de configuração dos jogadores
 * Escreve o ficheiro de estatisticas
 * Lê a os parametros de um jogador quando é necessario inserir um jogador
 */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <limits.h>
#include "file.h"
#include "logic.h"
#include "error.h"

/* Lê o ficheiro de configuração dos jogadores
 * Numero de baralhos e jogadores
 */
Config *read_config(char *filename)
{
    char buffer[MAX_LINE_LEN];

    Config *config = NULL;
    config = (Config *) ecalloc((size_t) 1, sizeof(Config));

    FILE *config_file = efopen(filename, "r");

    // Parametros gerais de configuração:
    // Número de jogadores e numero de baralhos
    fgets(buffer, MAX_LINE_LEN, config_file);
    sscanf(buffer, "%d-%d", &(config->num_decks), &(config->num_players));

    if (config->num_decks > 8 || config->num_decks < 4){
        fprintf(stderr, "Erro: número de baralhos invalido.\n");
        exit(EXIT_FAILURE);
    }

    if (config->num_players > 4 || config->num_players < 1) {
        fprintf(stderr, "Erro: número de jogadores invalido.\n");
        exit(EXIT_FAILURE);
    }
    // Leitura dos parâmetros de configuracão de cada jogador
    for (int i=0; fgets(buffer, MAX_LINE_LEN, config_file) != NULL && i < conf
ig->num_players; i++)
        config = read_player(buffer, config, i);

    fclose(config_file);

    return config;
}

/*
 *  Leitura dos parâmetros de configuração de cada jogador
 *  Recebe uma string e separa os parâmetros de configuração utlizando strtok(
)
 */
Config *read_player(char *line, Config *config, int count)
{
    // strtok separa o buffer no caracter '-'
    char *str = strtok(line, "-");

    //Leitura do tipo do jogador
    if (strcmp(str, "HU") == 0)
        config->player_type[count] = HU;
    else if (strcmp(str, "EA") == 0)
        config->player_type[count] = EA;
    else {
        fprintf(stderr, "Erro: tipo de jogador inválido.\n");
        exit(EXIT_FAILURE);
    }
```

```
    69
    70      str = strtok(NULL, "-");
    71      if (strlen(str) > MAX_PLAYER_NAME) {
    72          fprintf(stderr, "Erro: nome do jogador demasiado grande (Máx. 8 caract
    eres).\n");
    73          exit(EXIT_FAILURE);
    74      }
    75      strcpy(config->player_names[count], str);
    76
    77      //Ultimo segmento da string
    78      str = strtok(NULL, "\0");
    79      sscanf(str, "%d-%d", &config->money[count], &config->bets[count]);
    80      if (config->money[count] < 10 || config->money[count] > 500) {
    81          fprintf(stderr, "Erro: valor inicial de dinheiro inválido.\n");
    82          exit(EXIT_FAILURE);
    83      }
    84
    85      if (config->bets[count] < 2 ||
    86          config->bets[count] > config->money[count] / 4) {
    87          fprintf(stderr, "Erro: valor da aposta invalido!\n");
    88          exit(EXIT_FAILURE);
    89      }
    90
    91      return config;
    92  }
    93
    94  /* Vai buscar uma linha a stdin.
    95   * Modifica o buffer por referência.
    96   * O buffer fica vazio se o fgets() der overflow ou se a input for vazia.
    97   * Senão, o buffer fica com a string de input, sem o \n.
    98   */
    99  void get_line(char buffer[MAX_PLAYER_NAME+2])
   100  {
   101      int newline = 0;
   102      int c = 0;
   103      fgets(buffer, MAX_PLAYER_NAME+2, stdin);
   104
   105      // localização do \n
   106      newline = (int) strcspn(buffer, "\n");
   107
   108      // se não existir (ou seja, newline é o comprimento da string inserida),
   109      // sabemos que a string de stdin é maior que o buffer pode conter.
   110      if (newline == MAX_PLAYER_NAME+1) {
   111          strcpy(buffer, "");
   112          // Consumir o resto do buffer de stdin
   113          while ((c = getchar()) != '\n' && c != EOF);
   114          return;
   115      }
   116      // se existir, substituir por \0.
   117      // neste caso se buffer estiver vazio, permanece vazio.
   118      else
   119          buffer[newline] = '\0';
   120  }
   121
   122  //  Ler novo valor da aposta a partir stdin
   123  void get_new_bet(List *players)
   124  {
   125      char buffer[MAX_PLAYER_NAME+2] = {0}; // newline + nullbyte
   126      bool correct = false;
   127      List *aux = players->next;
   128      Player *cur_player = NULL;
   129
   130      printf("Insira o nome do jogador a modificar a aposta: ");
   131      get_line(buffer);
   132
   133      if (buffer[0] == '\0') {
   134          puts("Jogador não encontrado. Tente novamente primindo a tecla <b>.");
   135          return;
   136      }
   137
```

```
138         correct = false;
139         //Verificar se o jogador existe
140         while (aux && !correct) {
141             cur_player = (Player *) aux->payload;
142             if (strcmp(buffer, cur_player->name) == 0 && !correct)
143                 correct = true;
144             else
145                 aux = aux->next;
146         }
147 
148         if (!aux) {
149             puts("Jogador não encontrado. Tente novamente primindo a tecla <b>.");
150             return;
151         }
152 
153         correct = false;
154         cur_player = (Player *) aux->payload;
155         long new_bet = 0;
156         do {
157             printf("Insira o novo valor da aposta do jogador %s: ", cur_player->na
    me);
158             get_line(buffer);
159 
160             if (buffer[0] == '\0')
161                 puts("Nova aposta inválida.");
162             else {
163                 new_bet = strtol(buffer, NULL, 10);
164                 // o dinheiro do jogador está (essencialmente) garantido
165                 // de estar abaixo de INT_MAX (a não ser que se jogue mesmo muito)
166                 // fazendo com bet pertencente a [1, money]
167                 if (new_bet > cur_player->money || new_bet < 1)
168                     printf("Nova aposta inválida [1-%d].\n", cur_player->money);
169                 else
170                     correct = true;
171             }
172         } while (!correct);
173 
174         cur_player->bet = (int) new_bet;
175     }
176 
177 
178 
179     /*
180      * Obter o valor do dinheiro, tipo, nome e aposta do jogador
181      * Pede ate obter um valor correto
182      */
183     Player *get_new_player(int pos)
184     {
185         char buffer[MAX_PLAYER_NAME+2] = {0};
186         bool correct = false;
187         Type type = HU;
188         char name[MAX_PLAYER_NAME+1] = {0};
189         int money = 0;
190         int bet = 0;
191         long money_tmp = 0;
192         long bet_tmp = 0;
193         Player *new_player = NULL;
194 
195         printf("Escolheu o %dº lugar.\n", pos);
196 
197         correct = false;
198         do {
199             printf("Introduza o tipo do jogador [HU ou EA]: ");
200             get_line(buffer);
201 
202             if (buffer[0] == '\0')
203                 puts("Tipo de jogador inválido [HU ou EA].");
204             else {
205                 if (strcmp(buffer, "HU") == 0) {
206                     type = HU;
```

```
207                 correct = true;
208             }
209             else if (strcmp(buffer, "EA") == 0) {
210                 type = EA;
211                 correct = true;
212             }
213             else
214                 puts("Tipo de jogador inválido (HU ou EA).");
215         }
216     } while (!correct);
217
218     correct = false;
219     do {
220         printf("Introduza o nome do jogador [máx. 8 carac.]: ");
221         get_line(buffer);
222
223         if (buffer[0] == '\0')
224             puts("Nome do jogador inválido. Este tem no máximo 8 caracteres.")
    ;
225         else {
226             strcpy(name, buffer);
227             correct = true;
228         }
229     } while (!correct);
230
231     correct = false;
232     do {
233         printf("Introduza o dinheiro do jogador: ");
234         get_line(buffer);
235
236
237         if (buffer[0] == '\0')
238             puts("Dinheiro inválido.");
239         else {
240             money_tmp = strtol(buffer, NULL, 10);
241             if (money_tmp <= 1 || money_tmp > INT_MAX)
242                 printf("Quantidade de dinheiro inválida [de 1 a %d].\n", INT_M
    AX);
243             else {
244                 correct = true;
245                 money = (int) money_tmp;
246             }
247         }
248     } while (!correct);
249
250
251     correct = false;
252     do {
253         printf("Introduza a aposta do jogador: ");
254         get_line(buffer);
255
256         if (buffer[0] == '\0')
257             puts("Aposta inválida.");
258         else {
259             bet_tmp = strtol(buffer, NULL, 10);
260             if (bet_tmp > money_tmp || bet_tmp <= 0)
261                 printf("Aposta inválida [de 1 a %d].\n", money);
262             else {
263                 correct = true;
264                 bet = (int) bet_tmp;
265             }
266         }
267     } while (!correct);
268
269     // Alocar espaço para o jogador e escrever a configuração
270     new_player = (Player *) ecalloc((size_t) 1, sizeof(Player));
271
272     new_player->ingame = true;
273     new_player->type = type;
274     strcpy(new_player->name, name);
```

```
275         new_player->money = money;
276         new_player->bet = bet;
277
278         return new_player;
279     }
280
281
282     // Escrever o ficheiro de estatisticas
283     void write_stats(List *players, Player *house, List *old_players)
284     {
285         FILE *stats = NULL;
286         stats = efopen("stats.txt" , "w");
287
288         fprintf(stats, "Jogador\t\tTipo\tJogos\tVitorias\tEmpates\tDerrotas\tDinhe
    iro\n");
289
290         write_stats_players(stats, players);
291         write_stats_players(stats, old_players);
292
293         // O dinheiro da casa esta em modulo. E indicado se a casa perdeu ou ganho
    u
294         // dinheiro
295         if (house->money < 0)
296             fprintf(stats, "A casa perdeu: %d ¿\n", -1*house->money);
297         else if (house->money > 0)
298             fprintf(stats, "A casa ganhou: %d ¿\n", house->money);
299         else if (house->money == 0)
300             fprintf(stats, "A casa não ganhou nem perdeu dinheiro.\n");
301
302         fclose(stats);
303     }
304
305     //  Escrever as estatisticas dos jogadores
306     void write_stats_players(FILE *stats, List *players)
307     {
308         List *aux = players->next;
309         Player *cur_player = NULL;
310         while (aux) {
311             cur_player = (Player *) aux->payload;
312             if (cur_player->type == VA) {
313                 aux = aux->next;
314                 continue;
315             }
316             fprintf(stats, "%s\t", cur_player->name);
317             if (strlen(cur_player->name) < 8)
318                 fprintf(stats, "\t");
319             if (cur_player->type == EA)
320                 fprintf(stats, "EA\t");
321             else if (cur_player->type == HU)
322                 fprintf(stats, "HU\t");
323             fprintf(stats, "%d\t", cur_player->wins+cur_player->losses+cur_player-
    >ties);
324             fprintf(stats, "%d\t\t", cur_player->wins);
325             fprintf(stats, "%d\t", cur_player->ties);
326             fprintf(stats, "%d\t\t", cur_player->losses);
327             fprintf(stats, "%d ¿\n", cur_player->money);
328             aux = aux->next;
329         }
330     }
```

```c
#ifndef FILE_H
#define FILE_H

#include "logic.h"
#define MAX_LINE_LEN 100

Config *read_player(char *line, Config *game_config, int count);
Config *read_config(char *filename);

void get_line(char buffer[MAX_PLAYER_NAME+2]);
void get_new_bet(List *players);

Player *get_new_player(int pos);

void write_stats(List *players, Player *house, List *old_players);
void write_stats_players(FILE *stats, List *players);

#endif
```

```c
/*
 * Implementação das listas que vamos usar no projeto.
 * Estas são listas de payload genérico, doubly-linked com dummy head node.
 *
 * A desvantagem de utilizar listas de payload genérico é que
 * obrigamos o utilizador a fazer cast do payload para o que ele
 * precisar, o que pode ser chato, adicionar complexidade ao código
 * e/ou criar bugs difíceis de perceber se nos esquecermos de
 * fazer cast da payload... (ou seja, fazia-se dereference dum
 * void pointer, o que é ilegal) _E_ o utilizador tem de fazer free()
 * da payload manualmente.
 *
 * Mas, por outro lado, o programa torna-se mais modular,
 * mais fácil de compreender e abstrai-se assim toda a parte
 * das listas.
 *
 */
#include <stdlib.h>
#include <stdio.h>
#include "list.h"
#include "error.h"

// Aceder a um nó numa posilão especifica da lista
List *list_follow(List *head, int pos)
{
    if (pos < 0) {
        fprintf(stderr, "Erro: tentou-se aceder a um nó não existente na lista
.\n");
        exit(EXIT_FAILURE);
    }

    List *aux = head;
    for (int i = 0; i < pos; i++) {
        if (aux != NULL) {
            aux = aux->next;
        }
        else {
            fprintf(stderr, "Erro: tentou-se aceder a um nó não existente na l
ista.\n");
            exit(EXIT_FAILURE);
        }
    }

    return aux;
}

// Inserir um nó numa posição especifica da lista
void list_insert_pos(List *head, int pos, void *payload)
{
    List *aux = list_follow(head, pos - 1);

    List *new = (List *) ecalloc((size_t) 1, sizeof(List));
    new->payload = payload;

    new->next = aux->next;
    if (aux->next != NULL)
        aux->next->prev = new;
    else {
        // inserting at the tail, no need to set aux->next->prev
    }

    new->prev = aux;
    aux->next = new;
}


// Inserir na tail
void list_append(List *head, void *payload)
{
    List *aux = head;
```

```
    while (aux->next != NULL)
        aux = aux->next;
    List *tail = aux;

    List *new_tail = (List *) ecalloc((size_t) 1, sizeof(List));

    new_tail->payload = payload;

    new_tail->next = NULL;
    new_tail->prev = tail;
    tail->next = new_tail;
}

// Remover um nó da lista
void *list_remove(List *node)
{
    List *to_rm = node;
    void *payload = to_rm->payload;

    if (node->next != NULL) {
        node->next->prev = to_rm->prev;
    }
    else {
        // removing tail, it has no next, skip.
    }

    if (node->prev != NULL) {
        node->prev->next = to_rm->next;
    }
    else {
        fprintf(stderr, "Erro: tentou-se remover o dummy head node da lista.\n
");
        exit(EXIT_FAILURE);
    }

    free(to_rm);
    return payload;
}


// Remover um nó especifico da lista
void *list_remove_pos(List *head, int pos)
{
    List *to_rm = list_follow(head, pos);
    void *payload = list_remove(to_rm);
    return payload;
}
```

```c
#ifndef LIST_H
#define LIST_H

// Doubly-linked list with dummy head nodes
struct List {
    void *payload; // payload genérica
    struct List *next;
    struct List *prev;
};
typedef struct List List;

List *list_follow(List *head, int pos);
void list_insert_pos(List *head, int pos, void *payload);
void list_append(List *head, void *payload);
void *list_remove(List *node);
void *list_remove_pos(List *head, int pos);

#endif
```

```c
#include <stdlib.h>
#include <stdio.h>
#include <time.h>
#include <string.h>
#include <stdbool.h>
#include "logic.h"
#include "list.h"
#include "file.h"
#include "sdl.h"
#include "ea.h"
#include "error.h"

// Definição das operações válidas nas pilhas de cartas
void stack_push(Stack **sp, Card *card)
{
    Stack *old_sp = *sp;
    Stack *new = (Stack *) ecalloc((size_t) 1, sizeof(Stack));

    new->card = card;

    new->next = old_sp;
    new->prev = NULL;
    if (old_sp != NULL)
        old_sp->prev = new;
    else {
        // estamos a puxar o primeiro elemento, sp tava a NULL
    }

    *sp = new;
}

Card *stack_pop(Stack **sp)
{
    if (*sp == NULL) {
        fprintf(stderr, "Erro: tentou-se fazer pop numa stack vazia.\n");
        exit(EXIT_FAILURE);
    }

    Stack *pop = *sp;
    Card *card = pop->card;
    *sp = pop->next;
    if (pop->next != NULL)
        pop->next->prev = NULL;

    free(pop);

    return card;
}

/*
 * Ler a estrutura de configuração vinda de file.c:read_config()
 * e inicializar os jogadores de acordo.
 */
int init_game(Config *config, List *players)
{
    Player *new_player = NULL;
    const int num_decks = config->num_decks;

    for (int i = 0; i < MAX_PLAYERS; i++) {
        new_player = (Player *) ecalloc((size_t) 1, sizeof(Player));
        if (i + 1 <= config->num_players) {
            // Jogadores efetivos
            new_player->type = config->player_type[i];
            new_player->ingame = true;
            strcpy(new_player->name, config->player_names[i]);
            new_player->money = config->money[i];
            new_player->orig_bet = config->bets[i];
            new_player->bet = new_player->orig_bet;
        }
        else {
```

```
71              // Lugar não especificado na configuração.
72              strcpy(new_player->name, "Empty");
73              new_player->type = VA;
74              new_player->ingame = false;
75          }
76          new_player->playing = false;
77          list_append(players, new_player);
78      }
79
80      free(config);
81
82      return num_decks;
83  }
84
85  /* Wrapper para dar uma carta a um jogador.
86   * Atualiza cards_left no Megadeck e insere novos baralhos quando
87   * este acaba.
88   */
89  void give_card(Player *player, Megadeck *megadeck)
90  {
91      int random = 0;
92
93      if (megadeck->cards_left == 0)
94          megadeck->cards_left = create_megadeck(megadeck);
95
96      // random: 1 - cards_left
97      /* random é o número de nós a seguir na lista, por isso,
98       * tem de ser pelo menos 1 (dummy head node),
99       * ou no máximo o número de nós (se seguirmos *cards_left
100      * nós a partir do dummy head node, chegamos à tail) */
101     random = rand() % megadeck->cards_left + 1;
102     List *random_node = megadeck->deck;
103     for (int i = 0; i < random; i++) {
104         if (random_node->next != NULL)
105             random_node = random_node->next;
106         else {
107             fprintf(stderr, "Erro: tentou-se dar uma carta não existente.\n");
108             exit(EXIT_FAILURE);
109         }
110     }
111
112     count_cards((Card *) random_node->payload, megadeck);
113
114     stack_push(&(player->cards), random_node->payload);
115     player->num_cards++;
116     list_remove(random_node);
117     megadeck->cards_left--;
118 }
119
120 int create_megadeck(Megadeck *megadeck)
121 {
122     int total_cards = 0;
123     Card *cur_card = NULL;
124
125     for (int i = 0; i < megadeck->num_decks; i++)
126         for (int j = 0; j < 4; j++)
127             for (int k = 0; k < SUIT_SIZE; k++) {
128                 cur_card = (Card *) ecalloc(1, sizeof(Card));
129                 cur_card->suit = j;
130                 cur_card->id = k;
131                 list_append(megadeck->deck, cur_card);
132             }
133
134     total_cards = megadeck->num_decks * DECK_SIZE;
135     return total_cards;
136 }
137
138 void new_game(List *players, Player *house, Megadeck *megadeck)
139 {
140     // só fazer new_game quando já toda a gente jogou
```

```
141         if (find_active_player(players) != NULL) {
142             return;
143         }
144 
145         // só fazer new_game quando houver jogadores para jogar
146         if (find_ingame_player(players) == NULL) {
147             return;
148         }
149 
150         // atualizar as contagens das EAs com os valores da ronda anterior,
151         // antes de qualquer carta ser distribuída na nova ronda.
152         update_count(players, megadeck);
153 
154         // Limpar cartas e retirar apostas
155         clear_cards_take_bet(players, house, megadeck);
156         // Dar cartas
157         distribute_cards(players, house, megadeck);
158         // Encontrar qual o jogador que começa a jogar
159         find_playing(players, house);
160 
161         // Se a casa tiver blackjack...
162         if (house->status == BJ) {
163             // ...a ronda acaba logo.
164             pay_bets(players, house);
165             return;
166         }
167     }
168 
169     // limpar cartas e retirar apostas aos jogadores que possam jogar
170     void clear_cards_take_bet(List *players, Player *house, Megadeck *megadeck)
171     {
172         List *aux = players->next;
173         Player *cur_player = NULL;
174         while (aux != NULL) {
175             cur_player = (Player *) aux->payload;
176             if (cur_player->money < cur_player->bet)
177                 cur_player->ingame = false;
178 
179             // Se o jogador jogou na ronda antes
180             if (cur_player->ingame) {
181                 // Limpar cartas antigas
182                 destroy_stack(&cur_player->cards);
183                 cur_player->num_cards = 0;
184 
185                 // Verificar se o jogador pode jogar outra vez e
186                 // retirar as apostas a todos os jogadores
187                 // (apenas fazemos o cálculo dos dinheiros no final da ronda!)
188                 if (cur_player->type == HU) {
189                     if (cur_player->money < cur_player->bet)
190                         cur_player->ingame = false;
191                     else
192                         cur_player->money -= cur_player->bet;
193                 }
194                 else if (cur_player->type == EA) {
195                     // O jogador é EA, chamar hi_lo para modificar
196                     // a sua aposta antes de esta ser retirada,
197                     // de acordo com a estratégia hi-lo.
198                     #ifdef HI_LO
199                     hi_lo(cur_player, megadeck);
200                     #endif
201                     cur_player->money -= cur_player->bet;
202                 }
203             }
204             aux = aux->next;
205         }
206         destroy_stack(&house->cards);
207     }
208 
209     void distribute_cards(List *players, Player *house, Megadeck *megadeck)
210     {
```

```
211         List *aux = players->next;
212         Player *cur_player = NULL;
213         int public_house_points = 0;
214 
215         // Distribuir cartas realisticamente
216         for (int i = 0; i < 2; i++) {
217             aux = players->next;
218             cur_player = NULL;
219             while (aux != NULL) {
220                 cur_player = (Player *) aux->payload;
221                 // se puder jogar...
222                 if (cur_player->ingame)
223                     // dar uma carta
224                     give_card(cur_player, megadeck);
225                 aux = aux->next;
226             }
227             if (i == 1)
228                 // segunda vez, mostrar apenas os pontos da casa até agora
229                 public_house_points = count_points(house);
230             give_card(house, megadeck);
231         }
232 
233         house->num_cards = 1; // desenhar só uma carta
234         count_points(house);
235         if (house->points == 21)
236             house->status = BJ;
237         else {
238             house->points = public_house_points;
239             house->status = WW;
240         }
241     }
242 
243     /*
244      * Encontrar o primeiro jogador a jogar a seguir ao new_game()
245      */
246     void find_playing(List *players, Player *house)
247     {
248         bool found = false;
249         List *aux = players->next;
250         Player *cur_player = NULL;
251         while (aux != NULL) {
252             cur_player = (Player *) aux->payload;
253             if (cur_player->ingame) {
254                 // Colocar status a Waiting
255                 cur_player->status = WW;
256 
257                 // Contar pontos e verificar se tem blackjack
258                 count_points(cur_player);
259                 if (cur_player->points == 21)
260                     cur_player->status = BJ;
261 
262                 // Dar a vez ao primeiro jogador sem blackjack
263                 // E não dar a vez a ninguém se a casa tiver blackjack
264                 if (!(house->status == BJ) && !(cur_player->status == BJ) && !foun
    d) {
265                     cur_player->playing = true;
266                     found = true;
267                 }
268                 else
269                     cur_player->playing = false;
270 
271             }
272             aux = aux->next;
273         }
274     }
275 
276     /*
277      * Série de três funções auxiliares para encontrar certos jogadores na lista.
278      */
279     List *find_ingame_player(List *players)
```

```
280 {
281     List *aux = players->next; // dummy head
282     Player *cur_player = NULL;
283     while (aux != NULL) {
284         // iterar até ao jogador que está a jogar
285         cur_player = (Player *) aux->payload;
286         if (cur_player->ingame)
287             break;
288         else
289             aux = aux->next;
290     }
291 
292     return aux;
293 }
294 
295 List *find_active_player(List *players)
296 {
297     List *aux = players->next; // dummy head
298     Player *cur_player = NULL;
299     while (aux != NULL) {
300         // iterar até ao jogador que está a jogar
301         cur_player = (Player *) aux->payload;
302         if (cur_player->playing)
303             break;
304         else
305             aux = aux->next;
306     }
307 
308     return aux;
309 }
310 
311 /*
312  * Esta função é chamada na main apenas para
313  * registar as teclas de hit, stand, etc. se o jogador
314  * for humano (ou seja para as teclas não afetarem as
315  * jogadas dos jogadores EA mesmo se primidas)
316  */
317 List *find_active_human_player(List *players)
318 {
319     List *aux = players->next; // dummy head
320     Player *cur_player = NULL;
321     while (aux != NULL) {
322         // iterar até ao jogador que está a jogar
323         cur_player = (Player *) aux->payload;
324         if (cur_player->playing && cur_player->type == HU)
325             break;
326         else
327             aux = aux->next;
328     }
329 
330     return aux;
331 }
332 
333 void surrender(List *players, Player *house, Megadeck *megadeck)
334 {
335     List *aux = find_active_player(players);
336     if (aux == NULL) {
337         return;
338     }
339 
340     Player *cur_player = (Player *) aux->payload;
341 
342     cur_player->status = SU;
343     stand(players, house, megadeck);
344 }
345 
346 bool double_bet(List *players, Player *house, Megadeck *megadeck)
347 {
348     List *aux = find_active_player(players);
349     // não fazer nada se não for a vez dum jogador
```

```
350     if (aux == NULL) {
351         return false;
352     }
353
354     Player *cur_player = (Player *) aux->payload;
355
356     // não fazer nada se o jogador não pode fazer double
357     if (cur_player->money < cur_player->bet || cur_player->num_cards != 2) {
358         return false;
359     }
360
361     cur_player->money -= cur_player->bet;
362     cur_player->bet += cur_player->bet;
363
364     player_hit(players, house, megadeck);
365     if (!(cur_player->status == BU)) {
366         stand(players, house, megadeck);
367     }
368
369     return true;
370 }
371
372 void bet(List *players)
373 {
374     List *aux = find_active_player(players);
375     if (aux != NULL) {
376         return;
377     }
378
379     get_new_bet(players);
380 }
381
382 AddPlayerError add_player(List *players, List *old_players, Megadeck *megadeck
    , SDL_Window *window)
383 {
384     int pos = get_clicked_player();
385
386     if (pos == 0) {
387         // Sinalizar que o utilizador clicou fora da área de jogadores
388         return OUT;
389     }
390
391     List *aux = list_follow(players, pos);
392     Player *old_player = (Player *) aux->payload;
393     if (old_player->ingame) {
394         // Sinalizar que o utilizador clicou num jogador que ainda está a joga
    r
395         return NOTEMPTY;
396     }
397
398     show_add_player_input_message(window);
399
400     Player *new_player = get_new_player(pos);
401     // subtrair a count da ronda para, quando somarmos em new_game(),
402     // ficar a zero.
403     if (new_player->type == EA)
404         new_player->count -= megadeck->round_count;
405
406     old_player = (Player *) list_remove_pos(players, pos);
407     list_append(old_players, old_player);
408     list_insert_pos(players, pos, new_player);
409
410     return OK;
411 }
412
413 /*
414  * Esta função faz o stand dos jogadores, mas serve para
415  * encontrar o jogador que joga a seguir, por isso é chamada sempre
416  * no final da vez do jogador.
417  */
```

```c
void stand(List *players, Player *house, Megadeck *megadeck)
{
    List *aux = find_active_player(players);
    Player *cur_player = NULL;
    bool end_of_round = false;

    // Se não encontrarmos um jogador a jogar...
    if (aux == NULL) {
        // não fazer nada
        return;
    }

    // se encontrarmos, fazer-lhe stand, se este não tiver outros atributos
    cur_player = (Player *) aux->payload;
    if (cur_player->status == WW)
        cur_player->status = ST;
    cur_player->playing = false;

    // passar ao próximo jogador
    aux = aux->next;

    if (aux != NULL) {
        // se este próximo jogador existir,
        // procurar o próximo jogador válido a seguir
        while (aux != NULL) {
            cur_player = (Player *) aux->payload;
            if (cur_player->ingame && !(cur_player->status == BJ))
                break;
            else
                aux = aux->next;
        }
        // se ele existir, dar-lhe a vez
        if (aux != NULL) {
            cur_player->playing = true;
        }
        else {
            // não existe um próximo jogador válido para jogar
            end_of_round = true;
        }
    }
    else {
        // se não existir um próximo jogador, fizemos stand do último jogador
        end_of_round = true;
    }

    if (end_of_round) {
        // A ronda acabou, fazer o hit da casa e distribuir o dinheiro
        house_hit(house, megadeck);
        pay_bets(players, house);
    }
}

void player_hit(List *players, Player *house, Megadeck *megadeck)
{
    List *aux = find_active_player(players);
    Player *cur_player = NULL;

    if (aux != NULL) {
        cur_player = (Player *) aux->payload;
    }
    else {
        return;
    }

    give_card(cur_player, megadeck);
    count_points(cur_player);

    if (cur_player->points > 21) {
        cur_player->status = BU;
    }
```

```
488
489         if (cur_player->points >= 21) {
490             stand(players, house, megadeck);
491         }
492     }
493
494     void house_hit(Player *house, Megadeck *megadeck)
495     {
496         house->num_cards = 2;
497
498         while (house->points <= 16) {
499             give_card(house, megadeck);
500             count_points(house);
501         }
502
503         if (house->points > 21) {
504             house->status = BU;
505         }
506     }
507
508
509     /*
510      * Esta função distribui o dinheiro no final da ronda e é
511      * chamada sempre nessa altura.
512      */
513     void pay_bets(List *players, Player *house)
514     {
515         List *aux = players->next;
516         Player *cur_player = NULL;
517         while (aux != NULL) {
518             cur_player = ((Player *) aux->payload);
519
520             // not playing
521             if (!cur_player->ingame) {
522                 // skip this player
523                 aux = aux->next;
524                 continue;
525             }
526
527             // surrender
528             if (cur_player->status == SU) {
529                 house->money -= cur_player->bet / 2;
530                 cur_player->money += cur_player->bet / 2;
531                 cur_player->losses++;
532             }
533             // blackjack casa e do jogador: tie
534             else if (cur_player->status == BJ && house->status == BJ) {
535                 cur_player->money += cur_player->bet;
536                 cur_player->ties++;
537             }
538             // blackjack do jogador: win
539             else if (cur_player->status == BJ && !(house->status == BJ)) {
540                 cur_player->money += 2*cur_player->bet + cur_player->bet/2;
541                 house->money -= cur_player->bet + cur_player->bet/2;
542                 cur_player->wins++;
543             }
544             // blackjack da casa: loss
545             else if (!(cur_player->status == BJ) && house->status == BJ) {
546                 house->money += cur_player->bet;
547                 cur_player->losses++;
548             }
549             // bust da casa e do jogador: loss
550             else if (cur_player->status == BU) {
551                 house->money += cur_player->bet;
552                 cur_player->losses++;
553             }
554             // bust da casa: win
555             else if (!(cur_player->status == BU) && house->status == BU) {
556                 cur_player->money += 2*cur_player->bet;
557                 house->money -= cur_player->bet;
```

```
558                 cur_player->wins++;
559             }
560             // empate mesmos pontos: tie
561             else if (cur_player->points == house->points) {
562                 cur_player->money += cur_player->bet;
563                 cur_player->ties++;
564             }
565             // jogador ganha com mais pontos: win
566             else if (cur_player->points > house->points) {
567                 cur_player->money += 2*cur_player->bet;
568                 house->money -= cur_player->bet;
569                 cur_player->wins++;
570             }
571             // house ganha com mais pontos: loss
572             else if (cur_player->points < house->points) {
573                 house->money += cur_player->bet;
574                 cur_player->losses++;
575             }
576             else {
577                 // isto nunca pode acontecer
578                 fprintf(stderr, "Erro: estado de jogador desconhecido.\n");
579                 exit(EXIT_FAILURE);
580             }
581             aux = aux->next;
582         }
583     }
584
585
586     /*
587      * Conta os pontos do jogador.
588      * No primeiro loop atribui a todos os ases 11 pontos.
589      * No segundo loop se pontos > 21 remove 10 pontos ate não haverem ases.
590      */
591     int count_points(Player *player)
592     {
593         Stack *cards = player->cards;
594         int num_ace = 0;
595
596         player->points = 0;
597         while (cards != NULL) {
598             player->points += point_index(cards->card->id);
599             if (cards->card->id == 12)
600                 num_ace++;
601             cards = cards->next;
602         }
603
604         while (player->points > 21 && num_ace > 0) {
605             player->points -= 10;
606             --num_ace;
607         }
608
609         return player->points;
610     }
611
612
613     // Conversão do id da carta para pontos
614     int point_index(int id)
615     {
616         int points = 0;
617         id %= 13;
618         if (id == 12)
619             points = 11;
620         else if (id <12 && id>8)
621             points = 10;
622         else
623             points = id + 2;
624         return points;
625     }
626
627     /*
```

```c
 * Série de três funções que libertam a diversa memória alocada
 * ao longo do programa, como as pilhas dos jogadores e a lista dos jogadores
 */
void destroy_list(List *head)
{
    List *aux = head->next; // dummy head
    List *tmp = NULL;
    while (aux != NULL) {
        tmp = aux;
        aux = tmp->next;
        free(tmp->payload);
        free(tmp);
    }
    free(head);
}

void destroy_stack(Stack **cards)
{
    while (*cards != NULL)
        free(stack_pop(cards));
}

void destroy_players_list(List *players)
{
    List *aux = players->next;
    Player *cur_player = NULL;
    while (aux != NULL) {
        cur_player = (Player *) aux->payload;
        destroy_stack(&cur_player->cards);
        aux = aux->next;
    }
    destroy_list(players);
}
```

```c
#ifndef LOGIC_H
#define LOGIC_H

#include <stdbool.h>
#include <SDL2/SDL.h>
#include "types.h"
#include "list.h"

#define MAX_LINE_LENGTH 64
#define DECK_SIZE 52
#define SUIT_SIZE 13

int init_game(Config *config, List *players);

void stack_push(Stack **sp, Card *card);
Card *stack_pop(Stack **sp);

void give_card(Player *player, Megadeck *megadeck);
int create_megadeck(Megadeck *megadeck);

void new_game(List *players, Player *house, Megadeck *megadeck);
void clear_cards_take_bet(List *players, Player *house, Megadeck *megadeck);
void distribute_cards(List *players, Player *house, Megadeck *megadeck);
void find_playing(List *players, Player *house);

List *find_ingame_player(List *players);
List *find_active_human_player(List *players);
List *find_active_player(List *players);

void surrender(List *players, Player *house, Megadeck *megadeck);
bool double_bet(List *players, Player *house, Megadeck *megadeck);
void bet(List *players);
AddPlayerError add_player(List *players, List *old_players, Megadeck *megadeck
, SDL_Window *window);
void stand(List *players, Player *house, Megadeck *megadeck);
void player_hit(List *players, Player *house, Megadeck *megadeck);
void house_hit(Player *house, Megadeck *megadeck);

void pay_bets(List *players, Player *house);
int count_points(Player *player);
int point_index(int id);

void destroy_list(List *head);
void destroy_stack(Stack **cards);
void destroy_players_list(List *players);

#endif
```

```c
#include <SDL2/SDL.h>
#include <SDL2/SDL_ttf.h>
#include <SDL2/SDL_image.h>
#include <stdlib.h>
#include <stdio.h>
#include <stdbool.h>
#include <string.h>
#include "main.h"
#include "sdl.h"

const char myName1[] = "João Pinheiro 84086";
const char myName2[] = "João Freitas  84093";

/**
 * RenderTable: Draws the table where the game will be played, namely:
 * -  some texture for the background
 * -  the right part with the IST logo and the student name and number
 * -  squares to define the playing positions of each player
 * -  names and the available money for each player
 * \param _money amount of money of each player
 * \param _img surfaces where the table background and IST logo were loaded
 * \param _renderer renderer to handle all rendering in a window
 */
void RenderTable(List *players, TTF_Font *_font, SDL_Surface *_img[], SDL_Rend
erer *_renderer)
{
    SDL_Color black = {0, 0, 0, 255}; // black
    SDL_Texture *table_texture;
    SDL_Rect tableSrc, tableDest;
    int height;
    char money_str[STRING_SIZE];

    // set color of renderer to white
    SDL_SetRenderDrawColor(_renderer, 255, 255, 255, 255);

    // clear the window
    SDL_RenderClear(_renderer);

    tableDest.x = tableSrc.x = 0;
    tableDest.y = tableSrc.y = 0;
    tableSrc.w = _img[0]->w;
    tableSrc.h = _img[0]->h;

    tableDest.w = SEP;
    tableDest.h = HEIGHT_WINDOW;

    table_texture = SDL_CreateTextureFromSurface(_renderer, _img[0]);
    SDL_RenderCopy(_renderer, table_texture, &tableSrc, &tableDest);

    // render the IST Logo
    height = RenderLogo(SEP, 0, _img[1], _renderer);

    // render the student name
    height += RenderText(SEP+3*MARGIN, height, myName1, _font, &black, _render
er);

    // this renders the student number
    height += RenderText(SEP+3*MARGIN, height, myName2, _font, &black, _render
er);

    // 2xnewline
    height += 2*RenderText(SEP+3*MARGIN, height, " ", _font, &black, _renderer
);

    List *aux = players->next;
    Player *cur_player = NULL;
    while (aux) {
        cur_player = (Player *) aux->payload;
        if (cur_player->ingame) {
            sprintf(money_str, "%s (%s): %d euros",
```

```
67                      cur_player->name, cur_player->type == HU ? "HU" : "EA", cu
  r_player->money);
68              height += RenderText(SEP+3*MARGIN, height, money_str, _font, &blac
  k, _renderer);
69          }
70          aux = aux->next;
71      }
72
73      RenderPlayerArea(players, _renderer, _font);
74
75      // destroy everything
76      SDL_DestroyTexture(table_texture);
77  }
78
79
80  /* Desenhar a area do jogador
81   * Nome, aposta, estado e pontos
82   * Quadrado de cor diferente para o jogador que esta a jogar
83   */
84  void RenderPlayerArea(List *players, SDL_Renderer* _renderer, TTF_Font *_font)
85  {
86      SDL_Color white = {255, 255, 255, 255};
87      SDL_Rect playerRect;
88      char points_str[STRING_SIZE];
89      char status_str[STRING_SIZE];
90      List *aux = players->next;
91      Player *cur_player = NULL;
92      int num_player = 0;
93
94      while (aux) {
95          cur_player = (Player *) aux->payload;
96          if (cur_player->ingame) {
97              if (cur_player->playing)
98                  SDL_SetRenderDrawColor(_renderer, 255, 0, 0, 255);
99              else
100                 SDL_SetRenderDrawColor(_renderer, 255, 255, 255, 255);
101
102             playerRect.x = num_player*PLAYER_RECT_X;
103             playerRect.y = PLAYER_RECT_Y;
104             playerRect.w = PLAYER_RECT_W;
105             playerRect.h = PLAYER_RECT_H;
106
107             if (cur_player->status == WW || cur_player->status == ST)
108                 sprintf(points_str, "%d", cur_player->points);
109             else if (cur_player->status == BJ)
110                 sprintf(points_str, "BJ");
111             else if (cur_player->status == BU)
112                 sprintf(points_str, "BU");
113             else if (cur_player->status == SU)
114                 sprintf(points_str, "SU");
115
116             sprintf(status_str, "%s -- bet: %d, points: %s",
117                     cur_player->name, cur_player->bet, points_str);
118             RenderText(playerRect.x, playerRect.y-30, status_str, _font, &whit
  e, _renderer);
119             SDL_RenderDrawRect(_renderer, &playerRect);
120         }
121         aux = aux->next;
122         num_player++;
123     }
124 }
125
126 /*
127  * Série de três funções que mostram janelas popup quando
128  * inserimos um jogador (opção da tecla <a>)
129  */
130 void show_add_player_message(SDL_Window *window)
131 {
132     SDL_ShowSimpleMessageBox(SDL_MESSAGEBOX_INFORMATION,
133                              "Adicionar Jogador",
```

```
134                                 "Clique num lugar vazio para inserir um novo joga
    dor.",
135                                 window);
136 }
137
138 void show_add_player_error_message(SDL_Window *window, AddPlayerError error)
139 {
140     char error_msg[MAX_STR_SIZE] = {0};
141
142     switch(error) {
143         case OUT:
144             strcpy(error_msg, "Nao clicou dentro da area dos jogadores.\n"
145                     "Tente novamente primindo a tecla <a>.");
146             break;
147
148         case NOTEMPTY:
149             strcpy(error_msg, "Nao selecionou um lugar vazio.\n"
150                     "Tente novamente primindo a tecla <a>.");
151             break;
152
153         default:
154             break;
155     }
156     SDL_ShowSimpleMessageBox(SDL_MESSAGEBOX_INFORMATION,
157                                 "Adicionar Jogador",
158                                 error_msg,
159                                 window);
160 }
161
162 void show_add_player_input_message(SDL_Window *window)
163 {
164     SDL_ShowSimpleMessageBox(SDL_MESSAGEBOX_INFORMATION,
165                                 "Adicionar Jogador",
166                                 "Insira os dados do jogador no terminal.",
167                                 window);
168 }
169
170
171 // Obter posição para inserir o novo jogador na lista
172 int get_clicked_player()
173 {
174     SDL_Event event;
175     int i = 0;
176
177     while (1) {
178         SDL_PollEvent(&event);
179         if (event.type == SDL_MOUSEBUTTONDOWN)
180             break;
181     }
182
183     int mouse_x = event.button.x;
184     int mouse_y = event.button.y;
185
186     if (mouse_y >= PLAYER_RECT_Y && mouse_y <= PLAYER_RECT_Y + PLAYER_RECT_H)
187         while (mouse_x >= 0*PLAYER_RECT_X) {
188             mouse_x -= PLAYER_RECT_W;
189             i++;
190         }
191     else
192         i = 0;
193
194     return i;
195 }
196
197 /**
198  * RenderHouseCards: Renders cards of the house
199  * \param _house vector with the house cards
200  * \param _pos_house_hand position of the vector _house with valid card IDs
201  * \param _cards vector with all loaded card images
202  * \param _renderer renderer to handle all rendering in a window
```

```
 */
void RenderHouseCards(Player *house, SDL_Surface **_cards, TTF_Font *_font, SDL_Renderer* _renderer)
{
    int x = 0, y = 0;
    int div = WIDTH_WINDOW/CARD_WIDTH;
    Card *cur_card = NULL;
    int card_id = 0;
    int num_cards = 0;
    SDL_Color white = { 255, 255, 255, 255};
    char status_str[STRING_SIZE] = {0};
    char points_str[STRING_SIZE] = {0};

    if (house->status == WW || house->status == ST)
        sprintf(points_str, "%d", house->points);
    else if (house->status == BJ)
        sprintf(points_str, "BJ");
    else if (house->status == BU)
        sprintf(points_str, "BU");

    sprintf(status_str, "dealer: %s points", points_str);
    RenderText(20, 130, status_str, _font, &white, _renderer);

    Stack *aux = house->cards;
    Stack *tmp = NULL;
    // drawing all house cards
    while (tmp != house->cards) {
        aux = house->cards;
        while (aux->next != tmp)
            aux = aux->next;

        cur_card = aux->card;
        card_id = cur_card->id + cur_card->suit * SUIT_SIZE;

        // calculate its position
        x = (div/2 - house->num_cards/2 + num_cards)*CARD_WIDTH + 15;
        y = (int) (0.26f*HEIGHT_WINDOW);
        RenderCard(x, y, card_id, _cards, _renderer);

        num_cards++;
        tmp = aux;
    }

    // If the dealer has only 2 cards and no blackjack, draw the second card face down
    if (house->num_cards == 1 && house->status != BJ) {
        x = (div/2-house->num_cards/2+1)*CARD_WIDTH + 15;
        y = (int) (0.26f*HEIGHT_WINDOW);
        RenderCard(x, y, MAX_DECK_SIZE, _cards, _renderer);
    }
}

/**
 * RenderPlayerCards: Renders the hand, i.e. the cards, for each player
 * \param _player_cards 2D array with the player cards, 1st dimension is the player ID
 * \param _pos_player_hand array with the positions of the valid card IDs for each player
 * \param _cards vector with all loaded card images
 * \param _renderer renderer to handle all rendering in a window
 */
void RenderPlayerCards(List *players, SDL_Surface **_cards, SDL_Renderer* _renderer)
{
    int pos = 0, x = 0, y = 0;
    int num_player = 0;
    int num_cards = 0;
    int card_id = 0;

    List *aux = players->next; // dummy head
```

```
268     Player *cur_player = NULL;
269     Card *cur_card = 0;
270     Stack *aux_cards = NULL;
271     // Iterate over all players
272     while (aux) {
273         cur_player = (Player *) aux->payload;
274         if (cur_player->ingame) {
275             // Iterate over the stack backwards
276             aux_cards = cur_player->cards;
277             if (aux_cards)
278                 while (aux_cards->next)
279                     aux_cards = aux_cards->next;
280 
281             // agora aux_cards aponta para o último elemento da stack
282             while (aux_cards) {
283                 // get the card
284                 cur_card = aux_cards->card;
285                 card_id = cur_card->id + cur_card->suit * SUIT_SIZE;
286 
287                 // draw the card
288                 pos = num_cards % 4;
289                 x = (int) num_player * (SEP/4-5)+(num_cards/4)*12+15;
290                 y = (int) PLAYER_RECT_Y+10;
291                 if ( pos == 1 || pos == 3) x += CARD_WIDTH + 30;
292                 if ( pos == 2 || pos == 3) y += CARD_HEIGHT+ 10;
293                 RenderCard(x, y, card_id, _cards, _renderer);
294 
295                 num_cards++;
296                 aux_cards = aux_cards->prev;
297             }
298             num_cards = 0;
299         }
300         aux = aux->next;
301         num_player++;
302     }
303 }
304 
305 /**
306  * RenderCard: Draws one card at a certain position of the window, based on th
    e card code
307  * \param _x X coordinate of the card position in the window
308  * \param _y Y coordinate of the card position in the window
309  * \param _num_card card code that identifies each card
310  * \param _cards vector with all loaded card images
311  * \param _renderer renderer to handle all rendering in a window
312  */
313 void RenderCard(int _x, int _y, int _num_card, SDL_Surface **_cards, SDL_Rende
    rer* _renderer)
314 {
315     SDL_Texture *card_text;
316     SDL_Rect boardPos;
317 
318     // area that will be occupied by each card
319     boardPos.x = _x;
320     boardPos.y = _y;
321     boardPos.w = CARD_WIDTH;
322     boardPos.h = CARD_HEIGHT;
323 
324     // render it !
325     card_text = SDL_CreateTextureFromSurface(_renderer, _cards[_num_card]);
326     SDL_RenderCopy(_renderer, card_text, NULL, &boardPos);
327 
328     // destroy everything
329     SDL_DestroyTexture(card_text);
330 }
331 
332 /**
333  * LoadCards: Loads all images of the cards
334  * \param _cards vector with all loaded card images
335  */
```

```
336 void LoadCards(SDL_Surface **_cards)
337 {
338     int i = 0;
339     char filename[STRING_SIZE] = {0};
340 
341      // loads all cards to an array
342     for (i = 0; i < MAX_DECK_SIZE; i++) {
343         // create the filename !
344         sprintf(filename, ".//assets//cartas//carta_%02d.png", i+1);
345         // loads the image !
346         _cards[i] = IMG_Load(filename);
347         // check for errors: deleted files ?
348         if (_cards[i] == NULL) {
349             fprintf(stderr, "Unable to load image: %s\n", SDL_GetError());
350             exit(EXIT_FAILURE);
351         }
352     }
353     // loads the card back
354     _cards[i] = IMG_Load(".//assets//cartas//carta_back.jpg");
355     if (_cards[i] == NULL) {
356         fprintf(stderr, "Unable to load image: %s\n", SDL_GetError());
357         exit(EXIT_FAILURE);
358     }
359 }
360 
361 
362 /**
363  * UnLoadCards: unloads all card images of the memory
364  * \param _cards vector with all loaded card images
365  */
366 void UnLoadCards(SDL_Surface **_array_of_cards)
367 {
368     // unload all cards of the memory: +1 for the card back
369     for (int i = 0 ; i < MAX_DECK_SIZE + 1; i++ )
370     {
371         SDL_FreeSurface(_array_of_cards[i]);
372     }
373 }
374 
375 // Desenhar o estado do jogador
376 // Blackjack, Bust e Surrender
377 void render_status(List *players, TTF_Font *_font, SDL_Renderer *renderer)
378 {
379     SDL_Rect rect;
380 
381     char bust[] = "BUST";
382     char blackjack[] = "BLACKJACK";
383     char surrender[] = "SURRENDER";
384 
385     List *aux = players->next;
386     Player *cur_player = NULL;
387     SDL_Color white = { 255, 255, 255, 255};
388     for (int i=0; aux; i++) {
389         cur_player = (Player *) aux->payload;
390         rect.y = 380;
391         rect.h = 30;
392         if (cur_player->ingame) {
393             if (cur_player->status == BJ) {
394                 rect.x = 55 + 208*i;
395                 rect.w = 115;
396                 SDL_SetRenderDrawColor(renderer, 0, 0, 0, 255 );
397                 SDL_RenderFillRect(renderer, &rect);
398                 SDL_SetRenderDrawColor(renderer, 0, 0, 0, 255 );
399                 SDL_RenderDrawRect(renderer, &rect);
400                 RenderText(64+208*i, 382, blackjack, _font, &white, renderer);
401             }
402             else if (cur_player->status == BU) {
403                 rect.x = 80 + 208*i;
404                 rect.w = 70;
405                 SDL_SetRenderDrawColor(renderer, 255, 0, 0, 255 );
```

```
406                 SDL_RenderFillRect(renderer, &rect);
407                 SDL_SetRenderDrawColor(renderer, 255, 0, 0, 255 );
408                 SDL_RenderDrawRect(renderer, &rect);
409                 RenderText(94+(208*i), 382, bust, _font, &white, renderer);
410             }
411             else if (cur_player->status == SU) {
412                 rect.x = 55 + 208*i;
413                 rect.w = 115;
414                 SDL_SetRenderDrawColor(renderer, 255, 200, 0, 255 );
415                 SDL_RenderFillRect(renderer, &rect);
416                 SDL_SetRenderDrawColor(renderer, 255, 200, 0, 255 );
417                 SDL_RenderDrawRect(renderer, &rect);
418                 RenderText(64+208*i, 382, surrender, _font, &white, renderer);
419             }
420 
421         }
422         aux = aux->next;
423     }
424 }
425 
426 /**
427  * RenderLogo function: Renders the IST Logo on the window screen
428  * \param x X coordinate of the Logo
429  * \param y Y coordinate of the Logo
430  * \param _logoIST surface with the IST logo image to render
431  * \param _renderer renderer to handle all rendering in a window
432  */
433 int RenderLogo(int x, int y, SDL_Surface *_logoIST, SDL_Renderer* _renderer)
434 {
435     SDL_Texture *text_IST;
436     SDL_Rect boardPos;
437 
438     // space occupied by the logo
439     boardPos.x = x;
440     boardPos.y = y;
441     boardPos.w = _logoIST->w;
442     boardPos.h = _logoIST->h;
443 
444     // render it
445     text_IST = SDL_CreateTextureFromSurface(_renderer, _logoIST);
446     SDL_RenderCopy(_renderer, text_IST, NULL, &boardPos);
447 
448     // destroy associated texture !
449     SDL_DestroyTexture(text_IST);
450     return _logoIST->h;
451 }
452 
453 /**
454  * RenderText function: Renders the IST Logo on the window screen
455  * \param x X coordinate of the text
456  * \param y Y coordinate of the text
457  * \param text string where the text is written
458  * \param font TTF font used to render the text
459  * \param _renderer renderer to handle all rendering in a window
460  */
461 int RenderText(int x, int y, const char *text, TTF_Font *_font, SDL_Color *_co
    lor, SDL_Renderer* _renderer)
462 {
463     SDL_Surface *text_surface;
464     SDL_Texture *text_texture;
465     SDL_Rect solidRect;
466 
467     solidRect.x = x;
468     solidRect.y = y;
469     // create a surface from the string text with a predefined font
470     text_surface = TTF_RenderUTF8_Blended(_font,text,*_color);
471     if (text_surface == NULL)
472     {
473         fprintf(stderr, "TTF_RenderText_Blended: %s\n", TTF_GetError());
474         exit(EXIT_FAILURE);
```

```
475         }
476         // create texture
477         text_texture = SDL_CreateTextureFromSurface(_renderer, text_surface);
478         // obtain size
479         SDL_QueryTexture( text_texture, NULL, NULL, &solidRect.w, &solidRect.h );
480         // render it !
481         SDL_RenderCopy(_renderer, text_texture, NULL, &solidRect);
482 
483         SDL_DestroyTexture(text_texture);
484         SDL_FreeSurface(text_surface);
485         return solidRect.h;
486     }
487 
488     /**
489      * InitEverything: Initializes the SDL2 library and all graphical components:
    font, window, renderer
490      * \param width width in px of the window
491      * \param height height in px of the window
492      * \param _img surface to be created with the table background and IST logo
493      * \param _window represents the window of the application
494      * \param _renderer renderer to handle all rendering in a window
495      */
496     void InitEverything(int width, int height, TTF_Font **_font, SDL_Surface *_img
    [], SDL_Window** _window, SDL_Renderer** _renderer)
497     {
498         InitSDL();
499         InitFont();
500         *_window = CreateWindow(width, height);
501         *_renderer = CreateRenderer(width, height, *_window);
502 
503         // load the table texture
504         _img[0] = IMG_Load("assets//table_texture.png");
505         if (_img[0] == NULL) {
506             fprintf(stderr, "Unable to load image: %s\n", SDL_GetError());
507             exit(EXIT_FAILURE);
508         }
509 
510         // load IST logo
511         _img[1] = SDL_LoadBMP("assets//ist_logo.bmp");
512         if (_img[1] == NULL) {
513             fprintf(stderr, "Unable to load bitmap: %s\n", SDL_GetError());
514             exit(EXIT_FAILURE);
515         }
516 
517         // this opens (loads) a font file and sets a size
518         *_font = TTF_OpenFont("assets//FreeSerif.ttf", 16);
519         if(*_font == NULL) {
520             fprintf(stderr, "TTF_OpenFont: %s\n", TTF_GetError());
521             exit(EXIT_FAILURE);
522         }
523     }
524 
525     /**
526      * InitSDL: Initializes the SDL2 graphic library
527      */
528     void InitSDL()
529     {
530         // init SDL library
531         if (SDL_Init(SDL_INIT_EVERYTHING) != 0) {
532             fprintf(stderr, "Failed to initialize SDL: %s\n", SDL_GetError());
533             exit(EXIT_FAILURE);
534         }
535     }
536 
537     /**
538      * InitFont: Initializes the SDL2_ttf font library
539      */
540     void InitFont()
541     {
542         // Init font library
```

```
543         if (TTF_Init() == -1) {
544             fprintf(stderr, "TTF_Init: %s\n", TTF_GetError());
545             exit(EXIT_FAILURE);
546         }
547     }
548 
549     /**
550      * CreateWindow: Creates a window for the application
551      * \param width width in px of the window
552      * \param height height in px of the window
553      * \return pointer to the window created
554      */
555     SDL_Window *CreateWindow(int width, int height)
556     {
557         SDL_Window *window;
558         // init window
559         window = SDL_CreateWindow("Blackjack", WINDOW_POSX, WINDOW_POSY, width+EXT
    RASPACE, height, 0);
560         // check for error !
561         if (window == NULL) {
562             fprintf(stderr, "Failed to create window : %s\n", SDL_GetError());
563             exit(EXIT_FAILURE);
564         }
565 
566         return window;
567     }
568 
569     /**
570      * CreateRenderer: Creates a renderer for the application
571      * \param width width in px of the window
572      * \param height height in px of the window
573      * \param _window represents the window for which the renderer is associated
574      * \return pointer to the renderer created
575      */
576     SDL_Renderer *CreateRenderer(int width, int height, SDL_Window *_window)
577     {
578         SDL_Renderer *renderer;
579         // init renderer
580         renderer = SDL_CreateRenderer(_window, -1, 0);
581 
582         if (renderer == NULL) {
583             fprintf(stderr, "Failed to create renderer : %s", SDL_GetError());
584             exit(EXIT_FAILURE);
585         }
586 
587         // set size of renderer to the same as window
588         SDL_RenderSetLogicalSize(renderer, width+EXTRASPACE, height);
589 
590         return renderer;
591     }
```

```c
#ifndef SDL_H
#define SDL_H

#include <SDL2/SDL.h>
#include <SDL2/SDL_ttf.h>
#include <SDL2/SDL_image.h>
#include "main.h"
#include "logic.h"
#include "list.h"

#define STRING_SIZE 100       // max size for some strings
#define CARD_WIDTH 67         // card width
#define CARD_HEIGHT 97        // card height
#define WINDOW_POSX 100       // initial position of the window: x
#define WINDOW_POSY 100       // initial position of the window: y
#define WIDTH_WINDOW 900      // window width
#define HEIGHT_WINDOW 525     // window height
#define EXTRASPACE 150
#define MARGIN 5
#define SEP ((int) (0.95f*WIDTH_WINDOW))
// as quatro coordenadas dos retângulos dos jogadores
#define PLAYER_RECT_X (SEP/4 - 5) + 10
#define PLAYER_RECT_Y ((int) (0.55f*HEIGHT_WINDOW))
#define PLAYER_RECT_W SEP/4 - 5
#define PLAYER_RECT_H ((int) (0.42f*HEIGHT_WINDOW))

void InitEverything(int width, int height, TTF_Font **_font, SDL_Surface *_img
[], SDL_Window** _window, SDL_Renderer** _renderer);
void InitSDL();
void InitFont();

SDL_Window* CreateWindow(int width, int height);
SDL_Renderer* CreateRenderer(int width, int height, SDL_Window *_window);

int RenderText(int x, int y, const char *text, TTF_Font *_font, SDL_Color *_co
lor, SDL_Renderer* _renderer);
void RenderPlayerArea(List *players, SDL_Renderer* _renderer, TTF_Font *_font)
;

void show_add_player_message(SDL_Window *window);
void show_add_player_error_message(SDL_Window *window, AddPlayerError error);
void show_add_player_input_message(SDL_Window *window);

int get_clicked_player();

int RenderLogo(int, int, SDL_Surface *, SDL_Renderer *);

void RenderTable(List *players, TTF_Font *_font, SDL_Surface *_img[], SDL_Rend
erer *_renderer);

void RenderHouseCards(Player *house, SDL_Surface **_cards, TTF_Font *_font, SD
L_Renderer* _renderer);
void RenderPlayerCards(List *players, SDL_Surface **_cards, SDL_Renderer* _ren
derer);
void RenderCard(int _x, int _y, int _num_card, SDL_Surface **_cards, SDL_Rende
rer* _renderer);

void LoadCards(SDL_Surface **_cards);
void UnLoadCards(SDL_Surface **_array_of_cards);

void render_status(List *players, TTF_Font *_font, SDL_Renderer *renderer);

// definition of some strings: they cannot be changed when the program is exec
uted !
extern const char myName[];
extern const char myNumber[];

#endif
```

```c
#define MAX_PLAYER_NAME 8
#define MAX_PLAYERS 4
#include "list.h"

// Tipo de jogador: Entidade Artificial, Humano, Vazio
typedef enum {EA, HU, VA} Type;

// Estado do jogador: waiting, blackjack, busted, stand'ed', surrender
typedef enum {WW, BJ, BU, ST, SU} Status;

// Addplayer: OK (sem erro), OUT: o utilizador clicou fora da área dos jogadores
// NOTEMPTY: o utilizador clicou num jogador que ainda está a jogar
typedef enum {OK, OUT, NOTEMPTY} AddPlayerError;

// Codificação das ações das EAs:
// hit, stand, surrender, double->hit, double->stand
typedef enum {H='H', S='S', R='R', D='D', E='E'} Move;

typedef struct Card {
    int suit;
    int id;
} Card;

typedef struct Stack {
    Card *card;
    struct Stack *next;
    struct Stack *prev;
} Stack;

typedef struct Player {
    Type type;
    char name[MAX_PLAYER_NAME+1];
    bool ingame;
    bool playing;
    Status status;
    int money;
    int bet;
    int orig_bet;
    Stack *cards;
    int num_cards;
    int points;
    int wins;
    int losses;
    int ties;
    // guarda a contagem das cartas para o hi-lo
    // (apenas usada se o jogador for EA)
    int count;
} Player;

typedef struct Config {
    int num_decks;
    int num_players;
    int player_type[MAX_PLAYERS];
    char player_names[MAX_PLAYERS][MAX_PLAYER_NAME+1];
    int money[MAX_PLAYERS];
    int bets[MAX_PLAYERS];
} Config;

typedef struct Megadeck {
    int cards_left;
    const int num_decks;
    List *deck;
    // guarda a contagem das cartas duma ronda apenas para o hi-lo das EAs
    int round_count;
} Megadeck;

typedef struct Strategy {
    Move **hard;
    Move **soft;
```

```
} Strategy;
```

```
HHHHHHHHHH
HDDDDHHHHH
DDDDDDDDHH
DDDDDDDDDH
HHSSSHHHHH
SSSSSHHHHH
SSSSSHHHHH
SSSSSHHHRH
SSSSSHHRRR
SSSSSSSSSS

HHHHHHHHHH
HHHDDHHHHH
HHHDDHHHHH
HHDDDHHHHH
HHDDDHHHHH
HDDDDHHHHH
SEEEESSHHH
SSSSSSSSSS
```