```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
```

```
203   */
204  void RenderHouseCards(Player *house, SDL_Surface **_cards, TTF_Font *_font, SDL_Renderer* _renderer)
205  {
206      int x = 0, y = 0;
207      int div = WIDTH_WINDOW/CARD_WIDTH;
208      Card *cur_card = NULL;
209      int card_id = 0;
210      int num_cards = 0;
211      SDL_Color white = { 255, 255, 255, 255};
212      char status_str[STRING_SIZE] = {0};
213      char points_str[STRING_SIZE] = {0};
214
215      if (house->status == WW || house->status == ST)
216          sprintf(points_str, "%d", house->points);
217      else if (house->status == BJ)
218          sprintf(points_str, "BJ");
219      else if (house->status == BU)
220          sprintf(points_str, "BU");
221
222      sprintf(status_str, "dealer: %s points", points_str);
223      RenderText(20, 130, status_str, _font, &white, _renderer);
224
225      Stack *aux = house->cards;
226      Stack *tmp = NULL;
227      // drawing all house cards
228      while (tmp != house->cards) {
229          aux = house->cards;
230          while (aux->next != tmp)
231              aux = aux->next;
232
233          cur_card = aux->card;
234          card_id = cur_card->id + cur_card->suit * SUIT_SIZE;
235
236          // calculate its position
237          x = (div/2 - house->num_cards/2 + num_cards)*CARD_WIDTH + 15;
238          y = (int) (0.26f*HEIGHT_WINDOW);
239          RenderCard(x, y, card_id, _cards, _renderer);
240
241          num_cards++;
242          tmp = aux;
243      }
244
245      // If the dealer has only 2 cards and no blackjack, draw the second card face down
246      if (house->num_cards == 1 && house->status != BJ) {
247          x = (div/2-house->num_cards/2+1)*CARD_WIDTH + 15;
248          y = (int) (0.26f*HEIGHT_WINDOW);
249          RenderCard(x, y, MAX_DECK_SIZE, _cards, _renderer);
250      }
251  }
252
253  /**
254   * RenderPlayerCards: Renders the hand, i.e. the cards, for each player
255   * \param _player_cards 2D array with the player cards, 1st dimension is the player ID
256   * \param _pos_player_hand array with the positions of the valid card IDs for each player
257   * \param _cards vector with all loaded card images
258   * \param _renderer renderer to handle all rendering in a window
259   */
260  void RenderPlayerCards(List *players, SDL_Surface **_cards, SDL_Renderer* _renderer)
261  {
262      int pos = 0, x = 0, y = 0;
263      int num_player = 0;
264      int num_cards = 0;
265      int card_id = 0;
266
267      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     }
```