C:\Users\Pedro Tomás\Documents\IST\logo_IST_A.emf

Programação

## Projeto de Programação – Entrega Final

Robot de Limpeza – iClean

*Pedro Gonçalo Bravo Mendes, 81046*

*Joao Guilherme Santos, 81126*

*Sexta-feira, 22 de maio de 2015*

*Lisboa*

**Código do projeto de programação iClean**

**MAIN.C**

/\*\*

\*PROJETO FINAL PROGRAMAÇÃO -

\* Robot de Limpeza - iClean

\*

\*AUTORES:

\* Pedro Gonçalo Mendes, ist\_81046

\* João Guilherme Santos, ist\_81126

\*

\*

\* MEEC 2015

\* Insti­tuto Superior Tecnico

\* Lisboa, 22 de maio de 2015

\*/

#include <SDL2/SDL.h>

#include <SDL2/SDL\_ttf.h>

#include <SDL2/SDL\_image.h>

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#include <math.h>

#include "map.h"

#include "robot\_list.h"

#include "graphic.h"

#include "move\_robot.h"

#define STRING\_SIZE 1000

#define MAXSTRSIZE 1000

#define OBSTACULO 2

#define MAP\_SIZE 1000

#define WINDOW\_POSX 500

#define WINDOW\_POSY 250

#define SQUARE\_SIZE 36

#define EXTRASPACE 150

#define MARGIN 5

#define SUJO 0

#define LIMPO 1

#define OBSTACULO 2

#define ROBOT 3

#define OBJETIVO 4

int main( int argc, char\* argv[] )

{

int nSquareW = 15;

int nSquareH = 10;

int num\_obst=0; // nº de obstáculos

SDL\_Event event;

SDL\_Window\* window = NULL;

SDL\_Renderer\* renderer = NULL;

int width = 0;

int height = 0;

int delay = 300;

int fecho=1;

int aux=0;

char linha[STRING\_SIZE];

int num\_rob;

int xi, yi;

int xf;

int yf;

int bateria=0;

char caracter;

float maxbateria;

int quad\_perc=0;

int pause=0;

int add=0;

int reset=0;

srand((unsigned) time (NULL));

statemap \*\*matriz;

robot\_list \*list;

robot\_list \*robot;

robot\_list \*auxi;

FILE \* map;

sscanf(argv[2], "%f", &maxbateria);/\*recebe o valor maximo que a bateria pode ter \*/

map= fopen ( argv[1], "r");

if(map==NULL){

printf("Erro na abertura do ficheiro");

exit(EXIT\_FAILURE);

}

analisa\_syntax (map, &nSquareW, &nSquareH, &num\_obst);/\*le a primeira linha do ficheiro \*/

matriz=carrega\_map(map, nSquareW, nSquareH, num\_obst);

/\*imprime os obstaculos na matriz \*/

if(matriz==NULL){

printf("ERRO ao escrever o array 2D");

}

list=begin();/\*Inicialização da lista\*/

fgets(linha, STRING\_SIZE, map);

sscanf(linha, "%d", &num\_rob);/\*le do ficheiro o numero de robots\*/

for(aux=0;aux<num\_rob;aux++){

robot=le\_linha(map);/\*le o nome, posição inicial e nivel de bateria de um robot\*/

newposition (nSquareH, nSquareW, &xf, &yf);/\*posição objectivo \*/

matriz[xf][yf].objetivo=OBJETIVO;

robot->x\_objective=xf;

robot->y\_objective=yf;

robot->squares\_covered=0;

list=add\_robot(list, robot);/\*escrita na lista dos parametros associados ao novo robot(nome, posiçao, bateria, etc)\*/

}

cria\_matrizrobot(list, nSquareW, nSquareH);/\*Cria um mapa da sala para cada robot\*/

fclose(map);

carrega\_robot(matriz, list);/\*imprime os robots na matriz\*/

// calculate the window size

width = SQUARE\_SIZE\*nSquareW;

height = SQUARE\_SIZE\*nSquareH;

// initialize graphics

if ( !InitEverything(width, height, &window, &renderer) )

return -1;

while( fecho==1 )

{

// while there's events to handle

while( SDL\_PollEvent( &event ) )

{

if( event.type == SDL\_QUIT )

{

fecho=0;

// quit the program

// TODO

}

else if ( event.type == SDL\_KEYDOWN )

{

if(add==2){ /\*ContinuaÃ§Ã£o da rotina para adicionar novo robot \*/

caracter=(char)event.key.keysym.sym;

if(caracter<'a' || caracter>'z'){/\*o caracter inserido tem de ser uma letra em minusculo\*/

printf("Caracter invalido\n");

add=1;

}

sprintf(robot->name,"R%c",caracter);/\*guarda numa string o valor de R+o caracter inserido\*/

for(auxi=list; auxi!=NULL; auxi=auxi->next){

/\*percorre o lista e se o nome ja existe volta a começar a rotina de adicionar robots\*/

if(strcmp( auxi->name, robot->name)==0){

printf("Caracter invalido\n");

add=1;

}

}

if(add!=1){ /\*se o nome é valido segue para a proxima instrução\*/

printf("Introduza a posiçao\n");

add=3;

}

}

if(add==0)/\*desativa o switch quando se escreve o nome para o novo robot\*/

{

switch ( event.key.keysym.sym )

{

case 'a':

if(pause){

add=1;

}

break;

case 'q':

fecho=0;

printf("\nPrograma encerrado\n");

break;

case 'p':

pause=!pause;

break;

case 'i':

reset=1;

break;

case 'e':

print\_list(list, nSquareW, nSquareH);

/\*função que cria o ficheiro com as estatisticas \*/

break;

case SDLK\_DOWN:

if(delay<3000){

delay\*=1.1;

}

break;

case SDLK\_UP:

if(delay>50){

delay\*=0.9;

}

break;

default:

break;

}

}

}

else if( event.type == SDL\_MOUSEBUTTONDOWN)

{

if(pause && add==3){/\*continuação da rotina inserir uma posição inicial do robot\*/

SDL\_GetMouseState(&xi,&yi); /\*espera pelo evento do rato, ao clicar numa posiçao da matriz guarda a coordena x e y \*/

xi=xi/SQUARE\_SIZE;

yi=yi/SQUARE\_SIZE;

if((matriz[xi][yi].state==OBSTACULO) || (matriz[xi][yi].state==ROBOT)) /\*posições invalidas para inserir o novo robot (obstaculo e robot)\*/

{

printf("Posiçao inválida.\n");

}else{

do{

robot->bateria=rand()%100; /\*valor random da bateria entre 50 e 100\*/

}while(robot->bateria<50);

matriz[xi][yi].state=ROBOT;

robot->x=xi;

robot->y=yi;

robot->map\_robot=criar\_map(nSquareW, nSquareH);

robot->squares\_covered=0;

list=add\_robot(list, robot); /\*adiciona o robot na lista\*/

newposition (nSquareH, nSquareW, &xf, &yf); /\*nova posição objetivo\*/

matriz[xf][yf].objetivo=OBJETIVO;

robot->x\_objective=xf;

robot->y\_objective=yf;

num\_rob++;

add=0; /\*sai da rotina de adicionar robot\*/

carrega\_robot(matriz, list);

RenderMap(nSquareW, nSquareH, matriz, list, renderer);/\*insere o robot no ecra\*/

}

}

}

}

// add a delay

SDL\_Delay( delay );

if(add==1){

robot=new\_robot();

printf("Introduza o caracter do nome do robot\n");

add=2;

}

if(reset==1){/\*reinicializa o programa\*/

freemap(nSquareH, matriz); /\*apaga a matriz \*/

delete\_all\_list(list); /\*apaga a lista\*/

quad\_perc=0;

reset=0;

/\*volta a correr o codigo inicial de forma a ler novamente o ficheiro\*/

map= fopen ( argv[1], "r");

if(map==NULL){

printf("Erro na abertura do ficheiro");

exit(EXIT\_FAILURE);

}

analisa\_syntax (map, &nSquareW, &nSquareH, &num\_obst);

matriz=carrega\_map(map, nSquareW, nSquareH, num\_obst);

if(matriz==NULL){

printf("ERRO ao escrever o array 2D");

}

list=begin();

fgets(linha, STRING\_SIZE, map);

sscanf(linha, "%d", &num\_rob);

for(aux=0;aux<num\_rob;aux++){

robot=le\_linha(map);

newposition (nSquareH, nSquareW, &xf, &yf);

matriz[xf][yf].objetivo=OBJETIVO;

robot->x\_objective=xf;

robot->y\_objective=yf;

robot->squares\_covered=0;

list=add\_robot(list, robot);

}

cria\_matrizrobot(list, nSquareW, nSquareH);

fclose(map);

carrega\_robot(matriz, list);

RenderMap(nSquareW, nSquareH, matriz, list, renderer);

}

if(list==NULL){ /\*quando nao existem robot a limpar a simulação é colocada em pausa\*/

pause=1;

}

if(!pause){/\*se pause nao está ativo corre este codigo\*/

if(end\_move(matriz, nSquareW, nSquareH)==1){

/\*verifica quando a simulação acaba\*/

for(robot=list; robot!=NULL; robot=robot->next)

/\*percorre todos os robots da lista\*/

{

xi=robot->x;

yi=robot->y;

bateria=robot->bateria;

quad\_perc=robot->squares\_covered;

matriz[xi][yi].state=LIMPO;

if( xi==robot->x\_objective && yi==robot->y\_objective )/\*se o robot atingiu a posiçao objetivo gera uma nova posiçao para a qual se tenta deslocar\*/

{

matriz[robot->x\_objective][robot->y\_objective].objetivo=0;

matriz[xi][yi].state=LIMPO;

do{

newposition (nSquareH, nSquareW, &xf, &yf);

}while(robot->map\_robot[xf][yf].state==LIMPO);

matriz[xf][yf].objetivo=OBJETIVO;

robot->x\_objective=xf;

robot->y\_objective=yf;

}

MoveRobot(&xi, &yi, robot->x\_objective, robot->y\_objective);/\*posição seguinte\*/

while((matriz[xi][yi].state==OBSTACULO) || (matriz[xi][yi].state==ROBOT))/\*enquanto a posição seguinte é um obstaculo ou um robot gera uma nova posição objetivo\*/

{

matriz[robot->x\_objective][robot->y\_objective].objetivo=0;

if(matriz[xi][yi].state==OBSTACULO){

robot->map\_robot[xi][yi].state=OBSTACULO;/\*guarda a posição do obstaculo no mapa que o robot conhece\*/

}

do{

newposition (nSquareH, nSquareW, &xf, &yf);

}while(robot->map\_robot[xf][yf].state==LIMPO || robot->map\_robot[xf][yf].state==OBSTACULO);

matriz[xf][yf].objetivo=OBJETIVO;

robot->x\_objective=xf;

robot->y\_objective=yf;

xi=robot->x;

yi=robot->y;

MoveRobot(&xi, &yi, robot->x\_objective, robot->y\_objective);

}

quad\_perc++;

addpoint\_list(xi, yi, bateria, quad\_perc, robot);/\*atualiza os valores de cada robot contidos na lista \*/

}

dec\_bat(list, maxbateria); /\*decrementa a bateria\*/

list=delete\_robotlist(list, matriz);/\*apaga o robot da lista se a bateria for menor ou igual a zero\*/

carrega\_robot(matriz, list); /\*escreve os robots da lista na matriz\*/

RenderMap(nSquareW, nSquareH, matriz, list, renderer);/\*atualiza o ecra com a nova matriz\*/

}

}

}

/\*no final da simulação\*/

freemap(nSquareW, matriz);/\*apaga a matriz\*/

delete\_all\_list(list);/\*apaga a lista\*/

SDL\_DestroyRenderer(renderer);

SDL\_DestroyWindow(window);

SDL\_Quit();

return 1;

}

**MAP.C**

#include <SDL2/SDL.h>

#include <SDL2/SDL\_ttf.h>

#include <SDL2/SDL\_image.h>

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#include <math.h>

#include "map.h"

#include "robot\_list.h"

#include "graphic.h"

#include "move\_robot.h"

#define STRING\_SIZE 1000

#define SUJO 0

#define LIMPO 1

#define OBSTACULO 2

#define ROBOT 3

#define OBJETIVO 4

statemap \*\*criar\_map(int nSquareW, int nSquareH)

/\*creates space in memory to allocate the array 2D (matriz/mapa)\*/

{

statemap \*\*mapa;

int n, x, y;

mapa=(statemap\*\*)calloc(nSquareW, sizeof(statemap\*));

for(n=0; n<(nSquareW); n++){

mapa[n]=(statemap\*)calloc(nSquareH, sizeof(statemap));

}

if(mapa==NULL){

printf("Erro ao alocar na memoria");

exit(0);

}

for(x=0; x<nSquareW; x++)

{

for(y=0; y<nSquareH; y++)

{

mapa[x][y].state=SUJO;

}

}

return mapa;

}

void freemap(int nSquareW, statemap \*\*mapa)

/\*clean the local of memory that save the array 2D\*/

{

int aux;

for(aux=0; aux<nSquareW; aux++){

free(mapa[aux]);

}

free(mapa);

}

statemap \*\*carrega\_map(FILE \*map, int nSquareW, int nSquareH, int num\_obst)

/\*reads the position of obstacles from the file and prints in the array 2D that contains the information about the room\*/

{

char linha[STRING\_SIZE];

int i;

statemap \*\*matriz;

int x, y;

matriz=criar\_map(nSquareW, nSquareH);

for(i=0; i<(num\_obst) ; i++){

fgets(linha, STRING\_SIZE, map);

sscanf(linha, "[%d, %d]", &x, &y);

matriz[x][y].state=OBSTACULO;

}

return matriz;

}

void analisa\_syntax (FILE \*map, int \*nSquareW, int \*nSquareH, int \*num\_obstaculos)

/\*reads the first line of the file which contains the weight, height and the number of obstacles \*/

{

char linha[STRING\_SIZE];

int val1, val2, val3;

if (fgets (linha, STRING\_SIZE, map)==NULL)

{

printf("ERRO!, houve problemas na leitura do ficheiro");

exit(1);

}

// Condição que verifica se o nº de elementos na primeira linha são 3, em caso contrário imprime uma mensagem de erro

if(sscanf( linha, "%d %d %d" , &val1, &val2, &val3) != 3)

{

printf ("ERRO!, ficheiro de mapa inválido (argumentos inválidos na primeira linha) ");

exit(1);

}

\*nSquareW = val1;

\*nSquareH = val2;

\*num\_obstaculos = val3;

}

**MAP.H**

#ifndef MAP\_H\_INCLUDE

#define MAP\_H\_INCLUDE

typedef struct statemap{

int state;

char robot[3];

int x;

int y;

int objetivo;

}statemap;

/\*structure of the type of array 2D(matriz), all array 2D are composed for this variables \*/

statemap \*\*criar\_map(int , int);

void freemap(int , statemap \*\*);

statemap \*\*carrega\_map(FILE \*, int , int , int );

void analisa\_syntax (FILE \*, int \*, int \*, int \*);

#endif // MAP\_H\_INCLUDE

**ROBOT\_LIST.C**

#include <SDL2/SDL.h>

#include <SDL2/SDL\_ttf.h>

#include <SDL2/SDL\_image.h>

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#include <math.h>

#include "map.h"

#include "robot\_list.h"

#include "graphic.h"

#include "move\_robot.h"

#define STRING\_SIZE 1000

#define SQUARE\_SIZE 36

#define MARGIN 5

#define SUJO 0

#define LIMPO 1

#define OBSTACULO 2

#define ROBOT 3

#define OBJETIVO 4

robot\_list \*begin(void)

/\*return the list=NULL; the list is empty\*/

{

return NULL;

}

robot\_list \*new\_robot(void)

/\*create a new register in the list to a new robot\*/

{

robot\_list \*aux = (robot\_list\*) malloc( sizeof(robot\_list) );

if ( aux == NULL )

{

printf("Erro ao reservar a memoria");

exit(0);

}

aux->next = NULL;

return aux;

}

robot\_list \*le\_linha(FILE \* map)

/\*read the line of the file and add it to a new structure\*/

{

char linha[STRING\_SIZE];

robot\_list \*robot=new\_robot();

fgets(linha, STRING\_SIZE ,map);

sscanf(linha, "%s [%d,%d] %f", robot->name, &robot->x, &robot->y, &robot->bateria);

if(sscanf(linha, "%s [%d,%d] %f", robot->name, &robot->x, &robot->y, &robot->bateria) != 4){

printf ("ERRO!, ficheiro de mapa invÃ¡lido ( argumentos invÃ¡lidos)");

exit(0);

}

return robot;

}

robot\_list \* add\_robot(robot\_list \*list, robot\_list \* novo)

/\*add a new robot to a list in alphabetic order\*/

{

robot\_list \*aux = list;

if( aux==NULL || (strcmp(aux->name, novo->name)>0)){/\*se a condiçao é verdadeira significa que insere no inicio da lista\*/

novo -> next = list;

return novo;

}

while( aux->next != NULL && (strcmp(aux->next->name, novo->name)<0)){/\*percorre a lista ate o nome do robot seguinte nao ser maior\*/

aux=aux->next;

}

/\*insere a estrutura do robot na lista na posição correta, criando os apontadores para ligar novamente a lista\*/

novo -> next = aux -> next ;

aux -> next = novo;

return list;

}

robot\_list \*cria\_matrizrobot(robot\_list \*list, int nSquareW, int nSquareH)

/\*allocate memory to create a array 2D for each robot\*/

{

robot\_list \*aux=list;

statemap \*\*map;

for(aux=list; aux!=NULL; aux=aux->next){

map=criar\_map(nSquareW, nSquareH);

aux->map\_robot=map;

}

return list;

}

void carrega\_robot(statemap \*\*matriz, robot\_list \*list)

/\*insert all robots in array 2D that represent the map of the room\*/

{

robot\_list \*aux;

int x;

int y;

aux=list;

while(aux!=NULL){

x=aux->x;

y=aux->y;

matriz[x][y].state=ROBOT;

strcpy(matriz[x][y].robot, aux->name);

aux=aux->next;

}

}

void addpoint\_list(int xi, int yi, int bateria, int quad\_perc, robot\_list \*robot)

/\*add the new value of x, y, battery, and increment the number of squares covered, and save in array 2D(matriz) of each robot that this positions is clean\*/

{

robot->x=xi;

robot->y=yi;

robot->bateria=bateria;

robot->map\_robot[xi][yi].state=LIMPO;

robot->squares\_covered=quad\_perc;

}

robot\_list \*delete\_robotlist(robot\_list \*list, statemap \*\*matriz)

/\* delete a robot of the list if the level of battery is less or equal to zero\*/

{

robot\_list \*aux1=list;

robot\_list \*aux2=NULL;

while(aux1!=NULL){

if(aux1->bateria<=0){

matriz[aux1->x\_objective][aux1->y\_objective].objetivo=0;

if(aux2==NULL){

list=aux1->next;

free(aux1);

aux1=list;

}else{

aux2->next=aux1->next;

free(aux1);

aux1=aux2->next;

}

}else{

aux2=aux1;

aux1=aux1->next;

}

}

return list;

}

robot\_list \*delete\_all\_list(robot\_list \*list)

/\*delete all structures of the list \*/

{

robot\_list \*aux=list;

robot\_list \*destroy=NULL;

while(aux!=NULL){

destroy=aux;

aux=aux->next;

free(destroy);

}

return list;

}

void print\_list(robot\_list \*list, int nSquareW, int nSquareH)

/\* writes the file "estatisticas.txt" that contains the name, the number of cleaned and the number of covered squares, written in the list\*/

{

int n=0, x, y;

FILE \*est;

robot\_list \*aux=list;

est=fopen("estatisticas.txt", "w");

if(aux==NULL){

fputs("\nNao existem Robots a efectuar a limpeza\n", est);

}

else{

while(aux!=NULL){

for(x=0; x<nSquareW; x++)

{

for(y=0; y<nSquareH; y++)

{

if((aux->map\_robot[x][y].state)==LIMPO){

n++;

}

}

}

fprintf(est, "\nNome do Robot: %s\n", aux->name);

fprintf(est, "\nO Robot limpou %d quadrados\n", n+1);

fprintf(est, "\nO Robot passou por %d quadrados\n", aux->squares\_covered+1);

fputs("\n==================================================\n", est);

aux=aux->next;

n=0;

}

}

fclose(est);

}

void dec\_bat(robot\_list \*list, float maxbateria)

/\*decreases a random value to the old value of battery when the robot does a movement\*/

{

robot\_list \*aux=list;

float n, dec, dif;

while(aux!=NULL){

n=(float) rand()/(float) RAND\_MAX;

dif=maxbateria-0.1;

dec=n\*dif;

aux->bateria=(aux->bateria) - (dec+0.1);

aux=aux->next;

}

}

int end\_move(statemap \*\*matriz, int nSquareW, int nSquareH)

/\*When all room is clean the simulation stop and the robots stop their movement \*/

{

int x, y;

for(x=0; x<nSquareW; x++)

{

for(y=0; y<nSquareH; y++)

{

if(matriz[x][y].state==SUJO){

return 1;

}

}

}

return 0;

}

**ROBOT\_LIST.H**

#ifndef ROBOT\_LIST\_H\_INCLUDE

#define ROBOT\_LIST\_H\_INCLUDE

typedef struct robot\_list{

char name[3];

float bateria;

int x;

int y;

int x\_objective;

int y\_objective;

int squares\_covered;

statemap \*\*map\_robot;

struct robot\_list \*next;

}robot\_list;

/\*linked list that contains all necessary data to the simulation\*/

robot\_list \*begin(void);

void apaga\_lista(robot\_list \*);

robot\_list \*new\_robot(void);

robot\_list \*le\_linha(FILE \*);

robot\_list \* add\_robot(robot\_list \*, robot\_list \*);

robot\_list \*cria\_matrizrobot(robot\_list \*, int , int );

void carrega\_robot(statemap \*\*, robot\_list \*);

void addpoint\_list(int, int, int, int, robot\_list \*);

robot\_list \*delete\_robotlist(robot\_list \*, statemap \*\*);

robot\_list \*delete\_all\_list(robot\_list \*);

void dec\_bat(robot\_list \*, float);

int end\_move(statemap \*\*, int, int);

void print\_list(robot\_list \*, int, int);

#endif // ROBOT\_LIST\_H\_INCLUDE

**MOVE\_ROBOT.C**

#include <SDL2/SDL.h>

#include <SDL2/SDL\_ttf.h>

#include <SDL2/SDL\_image.h>

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#include <math.h>

#include "map.h"

#include "robot\_list.h"

#include "graphic.h"

#include "move\_robot.h"

void newposition (int nSquareH, int nSquareW, int \*xf, int \*yf)

/\*Calculate a new objective position in the room to the robot \*/

{

int x;

int y;

x=rand()%nSquareW;

y=rand()%nSquareH;

\*xf=x;

\*yf=y;

}

void MoveRobot(int \*\_xc, int \*\_yc, int xt, int yt)

/\*calculate the next position of robot \*/

{

int xc, yc;

double angle;

// just copy the values

xc = \*\_xc;

yc = \*\_yc;

// calculate the angle

angle = atan2((double)(yc-yt), (double)(xt-xc));

// calculate the new position

\*\_xc = floor(xc + cos(angle)+0.5);

\*\_yc = floor(yc - sin(angle)+0.5);

}

**MOVE\_ROBOT.H**

#ifndef MOVE\_ROBOT\_H\_INCLUDE

#define MOVE\_ROBOT\_H\_INCLUDE

void MoveRobot(int \*\_xc, int \*\_yc, int xt, int yt);

void newposition (int nSquareH, int nSquareW, int \*xf, int \*yf);

#endif // MOVE\_ROBOT\_H\_INCLUDE

**GRAPHIC.C**

#include <SDL2/SDL.h>

#include <SDL2/SDL\_ttf.h>

#include <SDL2/SDL\_image.h>

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#include <math.h>

#include "map.h"

#include "robot\_list.h"

#include "graphic.h"

// pre-processor definitions

#define STRING\_SIZE 100

#define MAP\_SIZE 1000

#define WINDOW\_POSX 500

#define WINDOW\_POSY 250

#define SQUARE\_SIZE 36

#define EXTRASPACE 150

#define MARGIN 5

#define SUJO 0

#define LIMPO 1

#define OBSTACULO 2

#define ROBOT 3

#define OBJETIVO 4

// some global constant variables

const char myName[] = "Pedro Mendes";

const char myName2[]= "Joao Santos";

const char myNumber[] = "IST\_81046";

const char myNumber2[] = "IST\_81126";

/\*\*

\* RenderLogo function: Renders the IST Logo on the window screen

\* \param x X coordinate of the Logo

\* \param y Y coordinate of the Logo

\* \param \_renderer renderer to handle all rendering in a window

\*/

int RenderLogo(int x, int y, SDL\_Renderer\* \_renderer)

{

SDL\_Texture \*text\_IST;

SDL\_Surface \*img\_IST;

SDL\_Rect boardPos;

// renders IST Logo

img\_IST = SDL\_LoadBMP("ist\_logo.bmp");

if (img\_IST == NULL)

{

printf("Unable to load bitmap: %s\n", SDL\_GetError());

exit(-5);

}

// square where the Logo is placed

boardPos.x = x;

boardPos.y = y;

boardPos.w = img\_IST->w;

boardPos.h = img\_IST->h;

// creates a texture and renders it in the screen

text\_IST = SDL\_CreateTextureFromSurface(\_renderer, img\_IST);

SDL\_RenderCopy(\_renderer, text\_IST, NULL, &boardPos);

// destroy texture and surface

SDL\_DestroyTexture(text\_IST);

SDL\_FreeSurface(img\_IST);

return img\_IST->h;

}

/\*\*

\* RenderText function: Renders the IST Logo on the window screen

\* \param x X coordinate of the text

\* \param y Y coordinate of the text

\* \param text string where the text is written

\* \param font TTF font used to render the text

\* \param \_renderer renderer to handle all rendering in a window

\*/

int RenderText(int x, int y, const char \*text, TTF\_Font \*font, SDL\_Renderer\* \_renderer)

{

SDL\_Color color = { 0, 0, 0 };

SDL\_Surface \*text\_surface;

SDL\_Texture \*text\_texture;

SDL\_Rect solidRect;

solidRect.x = x;

solidRect.y = y;

// creates a surface with some text

text\_surface = TTF\_RenderText\_Blended(font,text,color);

if(!text\_surface)

{

printf("TTF\_RenderText\_Blended: %s\n", TTF\_GetError());

exit(-5);

}

// creates a texture from the surface and renders it !

text\_texture = SDL\_CreateTextureFromSurface(\_renderer, text\_surface);

SDL\_QueryTexture( text\_texture, NULL, NULL, &solidRect.w, &solidRect.h );

SDL\_RenderCopy(\_renderer, text\_texture, NULL, &solidRect);

// destroy texture and surface

SDL\_DestroyTexture(text\_texture);

SDL\_FreeSurface(text\_surface);

return solidRect.h;

}

/\*\*

\* RenderMap function: Renders the map on the window screen according to their size

\* \param nSquareW number of squares to render (width)

\* \param nSquareH number of squares to render (height)

\* \param pos\_robot position of the robot (ID of the square)

\* \param robot\_name name of the robot (max. 2 letters)

\* \param \_renderer renderer to handle all rendering in a window

\*/

void RenderMap(int nSquareW, int nSquareH, statemap \*\*matriz, robot\_list \*list, SDL\_Renderer\* \_renderer)

{

TTF\_Font \*sans;

TTF\_Font \*serif;

SDL\_Rect gridPos;

int i,j, height;

char robot\_name[3];

// set color of renderer to some color

SDL\_SetRenderDrawColor( \_renderer, 255, 255, 255, 255 );

// clear the window

SDL\_RenderClear( \_renderer );

// opens a font style and sets a size

sans = TTF\_OpenFont("FreeSans.ttf", 24);

serif = TTF\_OpenFont("FreeSerif.ttf", 16);

if(!sans || !serif)

{

printf("TTF\_OpenFont: %s\n", TTF\_GetError());

exit(-5);

}

// render the IST Logo

height = RenderLogo(nSquareW\*SQUARE\_SIZE, 0, \_renderer);

// render the student name

height += RenderText(nSquareW\*SQUARE\_SIZE+3\*MARGIN, height, myName, serif, \_renderer);

// render the student name

RenderText(nSquareW\*SQUARE\_SIZE+3\*MARGIN, height, myNumber, serif, \_renderer);

height += RenderText(nSquareW\*SQUARE\_SIZE+3\*MARGIN, height+30, myName2, serif, \_renderer);

RenderText(nSquareW\*SQUARE\_SIZE+3\*MARGIN, height+30, myNumber2, serif, \_renderer);

//inserts in screen the name and the level of batery of robot

batery\_ecra(list, height, nSquareW, serif, \_renderer);

// grid position

gridPos.w = SQUARE\_SIZE;

gridPos.h = SQUARE\_SIZE;

gridPos.y = 0;

// iterate over all squares

for (i = 0; i < nSquareH; i++)

{

gridPos.x = 0;

for (j = 0; j < nSquareW; j++)

{

// writes a dirty square

if(matriz[j][i].state==SUJO){

SDL\_SetRenderDrawColor( \_renderer, 0, 191, 255, 255 );

SDL\_RenderFillRect( \_renderer, &gridPos );

SDL\_SetRenderDrawColor( \_renderer, 0, 0, 0, 255 );

SDL\_RenderDrawRect( \_renderer, &gridPos );

}

// writes an obstacle in a square

if(matriz[j][i].state==OBSTACULO){

SDL\_SetRenderDrawColor( \_renderer, 181, 181, 181, 255 );

SDL\_RenderFillRect( \_renderer, &gridPos );

SDL\_SetRenderDrawColor( \_renderer, 0, 0, 0, 255 );

SDL\_RenderDrawRect( \_renderer, &gridPos );

}

//writes a clean square

if(matriz[j][i].state==LIMPO){

SDL\_SetRenderDrawColor( \_renderer, 255, 255, 255, 255 );

SDL\_RenderFillRect( \_renderer, &gridPos );

SDL\_SetRenderDrawColor( \_renderer, 0, 0, 0, 255 );

SDL\_RenderDrawRect( \_renderer, &gridPos );

}

//writes a objective position in a square to the robot

if(matriz[j][i].objetivo==OBJETIVO){

SDL\_SetRenderDrawColor( \_renderer, 255, 0, 0, 255 );

SDL\_RenderFillRect( \_renderer, &gridPos );

SDL\_SetRenderDrawColor( \_renderer, 0, 0, 0, 255 );

SDL\_RenderDrawRect( \_renderer, &gridPos );

}

// writes a robot and his name in a square

if(matriz[j][i].state==ROBOT){

SDL\_SetRenderDrawColor( \_renderer, 255, 255, 255, 255 );

SDL\_RenderFillRect( \_renderer, &gridPos );

SDL\_SetRenderDrawColor( \_renderer, 0, 0, 0, 255 );

SDL\_RenderDrawRect( \_renderer, &gridPos );

strcpy(robot\_name,matriz[j][i].robot);

RenderText(gridPos.x+MARGIN, gridPos.y, robot\_name, sans, \_renderer);

}

//writes a line between the robot position and objective position that indicates his movement

gridPos.x += SQUARE\_SIZE;

}

gridPos.y += SQUARE\_SIZE;

}

//Draws the line that represents the direction of movement

DrawLine(list, \_renderer);

// render the changes above

SDL\_RenderPresent( \_renderer);

// destroy everything

TTF\_CloseFont(sans);

TTF\_CloseFont(serif);

}

/\*\*

\* InitEverything: Initializes the SDL2 library and all graphical components: font, window, renderer

\* \param width width in px of the window

\* \param height height in px of the window

\* \param \_window represents the window of the application

\* \param \_renderer renderer to handle all rendering in a window

\*/

int InitEverything(int width, int height, SDL\_Window\*\* \_window, SDL\_Renderer\*\* \_renderer)

{

SDL\_Window\* window = NULL;

SDL\_Renderer\* renderer = NULL;

if ( !InitSDL() )

return 0;

if ( !InitFont() )

return 0;

window = CreateWindow(width, height);

if ( window == NULL )

return 0;

renderer = CreateRenderer(width, height, window);

if ( renderer == NULL )

return 0;

\*\_window = window;

\*\_renderer = renderer;

return 1;

}

/\*\*

\* InitSDL: Initializes the SDL2 graphic library

\*/

int InitSDL()

{

// init SDL library

if ( SDL\_Init( SDL\_INIT\_EVERYTHING ) == -1 )

{

printf(" Failed to initialize SDL : %s\n", SDL\_GetError());

return 0;

}

return 1;

}

/\*\*

\* InitFont: Initializes the SDL2\_ttf font library

\*/

int InitFont()

{

// init font library

if(TTF\_Init()==-1)

{

printf("TTF\_Init: %s\n", TTF\_GetError());

return 0;

}

return 1;

}

/\*\*

\* CreateWindow: Creates a window for the application

\* \param width width in px of the window

\* \param height height in px of the window

\* \return pointer to the window created

\*/

SDL\_Window\* CreateWindow(int width, int height)

{

SDL\_Window\* window = NULL;

// init window

window = SDL\_CreateWindow( "iClean", WINDOW\_POSX, WINDOW\_POSY, width+EXTRASPACE, height, 0 );

if ( window == NULL )

{

printf("Failed to create window : %s\n", SDL\_GetError());

return NULL;

}

return window;

}

/\*\*

\* CreateRenderer: Creates a renderer for the application

\* \param width width in px of the window

\* \param height height in px of the window

\* \param \_window represents the window for which the renderer is associated

\* \return pointer to the renderer created

\*/

SDL\_Renderer\* CreateRenderer(int width, int height, SDL\_Window\* \_window)

{

SDL\_Renderer\* renderer;

// init renderer

renderer = SDL\_CreateRenderer( \_window, -1, 0 );

if ( renderer == NULL )

{

printf("Failed to create renderer : %s", SDL\_GetError());

return NULL;

}

// set size of renderer to the same as window

SDL\_RenderSetLogicalSize( renderer, width+EXTRASPACE, height );

return renderer;

}

void DrawLine(robot\_list \*list, SDL\_Renderer\* \_renderer)

/\*draws a line between the robot position and objective position that indicates his movement\*/

{

robot\_list \*aux=list;

for(aux=list; aux!=NULL;aux=aux->next){

SDL\_RenderDrawLine(\_renderer, ((aux->x)\*SQUARE\_SIZE+(SQUARE\_SIZE/2)), ((aux->y)\*SQUARE\_SIZE+(SQUARE\_SIZE/2)), ((aux->x\_objective)\*SQUARE\_SIZE+(SQUARE\_SIZE/2)), ((aux->y\_objective)\*SQUARE\_SIZE+(SQUARE\_SIZE/2)));

}

}

void batery\_ecra(robot\_list \*list, int height, int nSquareW, TTF\_Font \*\_font, SDL\_Renderer \*\_renderer)

/\*writes in the screen the name of robots and the level of battery\*/

{

robot\_list \*aux=list;

float bat;

char batery[6];

char nome[3];

while(aux!=NULL){

bat=aux->bateria;

strcpy(nome, aux->name);

snprintf(batery, 6, "%f", bat);

RenderText(nSquareW\*SQUARE\_SIZE+3\*MARGIN, height+60, nome, \_font, \_renderer);

height += RenderText(nSquareW\*SQUARE\_SIZE+3\*MARGIN+50, height+60, batery, \_font, \_renderer);

aux=aux->next;

}

}

**GRAPHIC.H**

#ifndef GRAPHIC\_H\_INCLUDE

#define GRAPHIC\_H\_INCLUDE

//declaration of the graphic functions of SDL2

int InitEverything(int , int , SDL\_Window\*\* , SDL\_Renderer\*\* );

int InitSDL();

int InitFont();

SDL\_Window\* CreateWindow(int , int );

SDL\_Renderer\* CreateRenderer(int , int , SDL\_Window\* );

void RenderMap(int , int , statemap \*\*, robot\_list \*, SDL\_Renderer\* );

int RenderText(int , int , const char\* , TTF\_Font \*, SDL\_Renderer\* );

int RenderLogo(int , int , SDL\_Renderer\* );

void DrawLine(robot\_list \*, SDL\_Renderer\* );

void batery\_ecra(robot\_list \*, int, int, TTF\_Font \*, SDL\_Renderer \*);

#endif // GRAPHIC\_H\_INCLUDE