Programmiererdokumentation

Das Mainprogramm inkludiert 4 verschiedene Headerfiles(utils.h, scores.h, maps.h, graphics.h). Die Funktionalität ist durch .c-Files definiert. Das Hilfsprogramm benutzt seine eigene Funktions, aber es gibt auch eine Headerfile(map_gen.h).

utils.h

KONSTANTE:

R 25 -> Radius der ausgezeichnete Kreis

R2 625 -> Quadrat der Radius der ausgezeichnete Kreis

MAKROS:

MAX(X, Y) (((X) < (Y)) ? (Y) : (X)) -> Gibt den größere Element zurück

FUNKTIONS:

int dist(int x1, int y1, int x2, int y2)

Diese Funktion liefert das Entfernung von Punk(x1, y1) und Punkt(x2, y2) in ganze Zahl.

double distd(double x1, double y1, double x2, double y2)

Diese Funktion liefert pünktlich das Entfernung von Punk(x1, y1) und Punkt(x2, y2) in Gleitpunktzahl.

int ist_in_kreis(int x1, int y1, int x2, int y2)

Diese Funktion liefert ob die Entfernung von Punkt(x1, y1) und Punkt(x2, y2) ist kleiner als Radius R.

char* to_mapname(int target)

Diese Funktion konvertiert eine Zahl zu Zeichenkette der Filename.

Uint32 my_timer(Uint32 ms, void *param)

Diese Funktion macht einen Zähler mit ‘ms‘ Intervall, mit Type von ‘SDL_USEREVENT‘ und liefert die ID, damit es gelöscht werden kann.

scores.h

STRUCTS:

Score{int scores[16]}

Es speichert die Ergebnisse der Levels in Integer-Array mit konstanter Größe 16.

Kodiert so:

- 0-> Nicht gemacht
- 1-> 3. Beste
- 2-> 2.Beste
- 3-> 1. Beste

FUNKTIONS:

Score load_scores()

Es liest die Ergebnisse von den Text-File „scores.txt“ ein und liefert diese Score-Struktur.

void new_score(Score* scr, int map_i, int score_i)

Es verändert das Ergebnis der ‘map_i‘-ste Level zu ‘score_i‘ in scr Score-Struktur.

void save_score(Score* scr)

Es speichert die scr Score-Struktur in „scores.txt“.

int klick_menu(int x, int y, Score* scr)

Es liefert das Zahl des Levels, das an der Position(x,y) an der Menu ist. Falls dieses Level ist noch nicht erreichbar, oder es gibt keine Level an der Position (x,y), dann es liefert -1.

graphics.h

KONSTANTE

GRAY -> Uint32 Darstellung der grauen Farbe

RED -> -> Uint32 Darstellung der roten Farbe

YELLOW -> Uint32 Darstellung der gelben Farbe

GREEN -> Uint32 Darstellung der grünen Farbe

NIK_W -> Breite des Nikolaus

NIK_H -> Höhe des Nikolaus

KID_W -> Breite der Kinder

KID_H -> Höhe der Kinder

HAUS_W -> Breite der Häuser

HAUS_H -> Höhe der Häuser

KERET -> Die Entfernung von Menü Optionen

STRUCTS:

Graphics{kids, houses, nikolaus, nordpole}

Es speichert die SDL_Surface Pointers der Bilder

FUNKTIONS:

Graphics load_graphics()

Es liest die Bilder ein und liefert eine Graphics-Struktur.

void free_graphics(Graphics* g)

Es freit das Graphics Struktur in die Speicher auf.

void draw_nikolaus(Map* m, Graphics* g, SDL_Surface* screen)

void draw_house(int x, int y, Graphics* g, SDL_Surface* screen)

void draw_nordpole(int x, int y, Graphics* g, SDL_Surface* screen)

void draw_kid(int x, int y, Graphics* g, SDL_Surface* screen)

void draw_option(int x, int y, const char* lvl, Graphics* g, Uint32 color, SDL_Surface* screen)

Diese Funktionen zeichnen an den Bildschirm (screen) in Position(x,y).

void draw_menu(const char* info, Graphics* g, Score* scr, SDL_Surface* screen)

Diese Funktion zeichnet das Menü an den Bildschirm und auch das Zeichenkette ‚info' in der Mitte.

void draw_map(Map* m, Graphics* g, SDL_Surface* screen)

Diese Funktion zeichnet das Map an den Bildschirm.

maps.h

KONSTANTE

PLAYER_V-> Uint32 Darstellung der grauen Farbe

TIMER_SLEEP -> -> Uint32 Darstellung der roten Farbe

STRUCTS:

Map{

int N; -> Zahl der Häuser

int id; -> Id des Mapes

int loaded; -> Ist das Map eingeladet oder nein?

int **matrix; -> Es speichert die Entfernung von a und b in matrix[a][b]

int *x; -> X Koordinate der Häuser

int *y; -> Y Koordinate der Häuser

int *color; -> Ist Nikolaus schon hier oder nein?

int best[3]; -> 3 besten Wegen

int pos; -> Position des Nikolaus [welches Haus]

int weg; -> Der gereiste Weg des Nikolaus

//ANIMATION:

double nik_x;-> X Koordinate des Nikolaus

double nik_y;-> Y Koordinate des Nikolaus

int nik_n; -> die Reste Trete zu dem nächsten Haus

double nik_vx; ->X-Geschwindigkeit von Nikolaus

double nik_vy; ->Y-Geschwindigkeit von Nikolaus

int nik_wohin; ->Wohin geht Nikolaus?

int nik_richtung; ->Welche Richtung?

int nik_moving; ->Ist Nikolaus bewegen?

}

FUNKTIONS:

Map load_map(const char* map_name, int target)

Diese Funktion ladet das Level von einen File(map_name) mit Adresse von ‚target ein und liefiert das Level als Map Struktur.

void unload_map(Map* m);

Diese Funktion verändert das Map zu ‚unladet' Zustand, so wird es nicht mehr angezeigt wird. Achtung, diese Funktion freit den Speicher nicht!

void free_map(Map* g);

Diese Funktion freit das Speichermemory in g.

int bewegen(Map* m);

Diese Funktion bewegt Nikolaus wohin er gehen möchte und liefert ob er schon da ist.

int kann_bewegen(Map* m, int wohin);

Diese Funktion liefert ob Nikolaus kann zu dem ‚wohin' Haus gehen.

SDL_TimerID klick_map(Map* m, int x, int y);

Diese Funktion simulate ein Klick on Position(x,y), falls Nikolaus kann da gehen, dann er beginnt zu dahin gehen und die Funktion liefert das TimerID von die Bewegungevent. Falls er kann dahin nicht gehen, oder es gibt keine Haus in Position(x,y), dann liefert -1.

int get_node_at(Map* m, int x, int y);

Diese Funktion liefert welche Haus ist in Position(x,y), fall es gibt keine dann liefert -1.

SDL_TimerID gehen_zu(Map* m, int wohin);

Es liefert ein TimerID zu die Bewegung zum Haus(wohin)

int end_game(Map* m);

Diese Funktion entscheidet ob das Spiel enden soll.

- 0 falls es soll nicht enden
- 1 falls der Spieler verloren ist
- 2 falls der Spieler gewonnen hat

Das Hilfsprogramme ist in map_gen.h und map_gen.c verwirklicht. Dieses HIlfprogramm kalkuliert die 3 kürzesten Hamiltion-Kreis in der Graph und macht ein neues File, die Spielprogramm einlesen kann.

map_gen.h

STRUCTS:

Graph{

int **matrix;	-> Die Entfernung von a und b Knote in matrix[a][b]
int *x;	-> Die X-Koordinate der Knoten
int *y;	-> Die Y-Koordinate der Knoten
int *von_arr;	-> Von welchem Knote ist die Kanten gebunden
int *zu_arr;	-> Zu welchem Knote ist die Kanten gebunden
int N, M;	-> N: Anzahl von Knoten M: Anzahl von Kanten

};

FUNKTIONS:

int dist(int x1, int y1, int x2, int y2);

Diese Funktion liefert das Entfernung von Punk(x1, y1) und Punkt(x2, y2) in ganze Zahl.

int gut(Graph* g, int *m, int i);

Diese Funktion entscheidet ob die ‚m' Permutation möglich ist.

int* brute_force(Graph* g);

Diese Funktion liefert die 3 besten Wegen in einen Integer Pointer. Es löst das Problem mit zurücktretende Suchen. Also es probiert aus alles Permutation und falls es möglich ist, dann wird die besten gespeichert. Es durchführt bis wir alle mögliche Permutation gefunden haben. Diese Funktion durchführt nicht in polynomiale Zeit, also man soll es nur zu kleine Eingangsfile benutzen.

Graph load_graph(const char* file_name);

Diese Funktion einladet einen Graph-File und liefert in einen Graph-Struktur.

void free_graph(Graph* g);

Diese Funktion freit den Speicher in g Position.

void write_graph(const char* file_name, Graph* g);

Diese Funktion schreibt das gelöste Graph in File(file_name) aus. Dieses File ist schon spielbar und einlesbar durch das Mainprogramm.

void convert(const char* in, const char* out);

Es konvertiert den File(in) zu ein spielbar File(out).