Scrabble – Programmierer Dokumentation

Dieses Programm ist eine vereinfachte Version das originelle Brettspiel. Das Ziel der Spieler ist, je mehrere Wörter aus den Buchstaben auf das Brett legen, die in seiner Hand sind. Natürlich kann man die Steine (auf denen die Buchstaben mit ihren Werte sind) austauschen. In diesem Programm kann man entweder ein ausgewählter Stein, oder alle austauschen, die in seiner Hand sind. Man kann kein Stein auf ein anderen platzieren und darf man kein auf das Brett wegnehmen. Falls man sich umentscheiden hat, dann kann man den letzten Schritt zurücknehmen. Falls man den Schritt validieren möchte, dann presst man den Taste F, und das Programm prüft die Gültigkeit des Wortes. Falls ein Wort auf dem Brett nicht gültig ist, dann nimmt das Programm den letzten Schritt weg. Dazu benutzt das Programm auch ein „Wörterbuch“.

Das Programm hat ein GUI (Graphical User Interface), wozu benutzt es eine Bibliothek, das SDL.

```
#include <SDL2/SDL.h>
#include <SDL_image.h>
#include <SDL2/SDL2_gfxPrimitives.h>
#include <SDL2/SDL_ttf.h>
```

Die wichtigsten Datenstrukturen sind die folgenden:

```
struct eventTable{
    int centerError;
    int valid;
    int first;
    int connected;
    int illegal;
    int jokerInput;
    int currentJoker;
};
```

Diese Struktur ist ein Paket von solche Variablen, die wichtig für spezielle Events sind. „centerError“ ist 1, falls das erste Wort nicht durch die Mitte führt. „valid“, „connected“ und „illegal“ zeigen, falls der Spieler keinen richtigen Schritt macht. „first“ sagt natürlich, falls dieses Wort das erste ist oder nicht. „jokerInput“ und „currentJoker“ sind wichtig für die Joker-Behandlung. Diese Struktur wird im main() definiert.

```
typedef struct{
    char name;
    int pts;
    int x;
    int y;
}t_Stones;
```

Diese Struktur enthält Informationen über die Steine. „name“ enthält einen Charakter, was der Buchstabe auf dem Stein ist. „pts“ ist die Anzahl der Punkt, die man für die Benutzung dieses Buchstabens bekommt. „x“ und „y“ zeigen die Koordinaten des Steines im „stones_texture.png“. Diese Daten werden aus dem Datei „stones_en.latsa“ eingelesen. Diese ist eine binäre Datei. Die 100 Steine werden in einem t_Stones Array gespeichert.

```
struct HandItem{
    int no;

    int index;
    struct HandItem *next;
    struct HandItem *prev;
};

struct HandList{
    int listSize;
    struct HandItem *first;
    struct HandItem *last;
};
```

Diese Struktur ist eine der wichtigsten Strukturen in diesem Programm. Diese ist eine verkettete Liste und enthält die Steine, die im Hand der Spieler sind. Es gibt zahlreiche Funktionen auch, die diese Liste behandeln.
Im „HandItem“ gibt es zwei Daten: „no“ und „index“. „no“ ist die Nummer des Steines im Array des Steines. „index“ ist die Nummer des Steines im Hand des Spielers. Es gibt auch zwei Pointers, die auf den nächsten, bzw. den letzten Element der Liste zeigen.
Die Struktur „HandList“ besteht aus einem Integer „listSize“, der die Anzahl der Elemente enthält, und aus zwei Pointers, die auf den ersten und den letzten Element der Liste zeigen.

```
struct t_board{
    int x;
    int y;
    int Stone;
};
```

Diese ist auch eine wichtige Struktur. Sie enthält die x und die y Koordinaten der Elementen des Brettes. Das Brett ist ein 15x15 große Matrix. Diese Koordinaten zeigen auf die Koordinaten der Quadraten auf dem Bildscrirm, wo die Steinen sein können. „Stone" enthält die Nummer des Steines im Array.

```
struct HandPosition{
    int x;
    int y;
};
```

Falls man die Steine im Hand auf dem Bildschirm zeichnen möchte, dann muss man die Koordinaten wissen, wo diese Steine liegen sollen. Diese Struktur enthält die x- und y-Koordinaten zu einem HandItem. Aus diese Struktur wurde ein Array gemacht.

```
struct previous_step{
    int ii;
    int jj;
};
```

Diese Struktur ist wichtig für den Zurückkehr des Schrittes. Diese enthält die Zeilen- und die Spaltennummer des Brettes, wohin man in diesem Schritt Steine platziert hat. Es gibt zu dieser Struktur auch ein Array.

```
int no, index;
int score = 0;
char score_str[5] = "0";
bool holdingStone = false;
bool quit = false;
```

Die wichtigste Variablen sind diese. „holdingStone", „no" und „index" sind wichtig bei Operationen mit Steinen im Hand. „score" ist selbstverständlich. „score_str" ist das String-Form von „score", um es ausschreiben zu können. „quit" ist villeicht das Wichtigste. Bis es falsch ist, rennt das Programm.

Funktionen

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

Natürlich main() ist die wichtigste Funktion. Es initialisiert alle wichtige Variablen und Strukturen. Der Kern ist aber ein while-Schleife, das lauft, bis das Window eingeschlossen wird. Da werden verschiedene Evente behandelt und die gebrauchte Funktionen beruft.

```
void Game_init(int width, int height, SDL_Window **pwindow, SDL_Renderer **prenderer,
               HandList *hand, TTF_Font **font,t_board board[15][15], t_Stones *Stones,
               previous_step *rst, int *RandTable, HandPosition *posOfHands, int *prev);
```

Diese Funktion befindet sich im init.c und wird am Anfang berufen. Initialisiert sie alle wichtigen Arrays und Variablen. Sie erstellt auch das Window und das Renderer. Deshalb braucht sie so viele Parameters.

```
void generateStones();
```

Diese importiert alle Dateien aus dem Datei stones_en.latsa im Array `t_Stones Stones[100];` Befindet sich im stones.c .

```
void generateBoard(SDL_Renderer *r, char* src);
void defineBoard(t_board board[15][15], previous_step *rst);
```

Diese Funktionen sind im board.c . Die Funktion generateBoard zeichnet das Brett auf den Bildschirm. Das Brett befindet sich im Datei Scrabbleboard.jpg. Dieses Bild ist modifiziert, das originelle befindet sich hier: https://www.flickr.com/photos/xouved/5371523488. Die Funktion defineBoard initialisiert den Matrix board und das Array rst.

```
void initFont(TTF_Font **font);
void drawText(SDL_Renderer *renderer, TTF_Font *font, int posX, int posY, int width, int height, char *text);
```

Diese Funktionen sind im text.c . Funktion initFont initialisiert das benutzte Font, was in diesem Programm Bauhaus 93 ist. Funktion drawText zeichnet den String im char *text in der bestimmten Position.

```
void drawHand(SDL_Renderer *renderer, HandList *hand, char* stn_src, t_Stones *Stones, HandPosition *posOfHands);
void initialHand(HandList *hand, int *RandTable, HandPosition *posOfHands, int *prev);
HandItem* addHand(HandList *hand, HandItem *prevent, int *RandTable);
HandItem* addHandReset(HandList *hand, HandItem *prevent, int *prev);
void freeHand(HandItem *first);
void delHandItem(HandItem *first, int index, HandList *hand);
void reindexList(HandItem *first);
void regenerateList(HandList *hand, int *RandTable, int *prev);
void resetList(HandList *hand, int *RandTable, int *prev);
```

Diese Funktionen sind im hand.c . Die meiste Funktionen behandeln die verkettete Liste. Natürlich drawHand zeichnet das Hand auf den Bildschirm, regenerateList wiedererfüllt das Hand mit Steinen und resetList stellt das Hand in dem letzten State.

```
void drawStone(SDL_Renderer *r, char *src, int no, int x, int y, int width, int height, t_Stones *Stones);
```

Diese Funktion ist im stone_img.c . Hier werden die Steine auf das Brett gezeichnet.

```
void refreshRenderer(SDL_Renderer *renderer, TTF_Font *font, HandList *hand, t_board board[15][15],
                     char *score_str, t_Stones *Stones, eventTable *events, HandPosition *posOfHands);
bool input_text(char *dest, size_t hossz, SDL_Rect teglalap, SDL_Color hatter, SDL_Color szoveg,
                TTF_Font *font, SDL_Renderer *renderer, t_Stones *Stones);
```

Diese sind im render.c . refreshRenderer macht alle Zeichnungen falls ein Event passiert. Hier werden die Joker-Algorithmen auch behandelt. Dazu wird input_text benutzt, was ich aus der Webseite infoc.eet.bme.hu herauskopiert habe. Natürlich habe es auch ein bisschen modifiziert.

```
void enterWord(HandList *hand, t_board board[15][15], int *score, char *score_str, t_Stones *Stones, previous_step *rst,
               eventTable *events, int *RandTable, int *prev);
void resetState(HandList *hand, t_board board[15][15], int *connected, t_Stones *Stones, previous_step *rst, int *RandTable, int *prev);
void mixLetters(HandList *hand, t_board board[15][15], int *score, char *score_str, t_Stones *Stones, previous_step *rst,
                eventTable *events, int *RandTable, int *prev);
void changeTile(HandList *hand, int *RandTable, int *prev, int index);
```

Diese sind im events.c . Sie behandeln die verschiedenen Events. Ihre Namen zeigen ihre Aufgaben. Im enterWord werden alle Zeilen und Spalten überprüft, ob sie richtige Wörter enthalten.

```
bool isThatInHand(SDL_Event event, int *no, HandList *hand, int *index, HandPosition *posOfHands);
bool isOnTable(SDL_Event event, HandList *hand, int index, int *drawX, int *drawY, t_board board[15][15],
               t_Stones *Stones, eventTable *events, previous_step *rst);
```

Diese sind im mouse.c . Sie prüfen, ob es auf die Elemente im Hand oder auf dem Brett geklickt wurde. Sie modifizieren die passenden Arrays auch.

```
bool checkWord(char *line, int vert, int linenumber, t_board board[15][15]);
bool wordCmp(char* buffer, char* thisWord);
bool checkDictionary(char *word);
```

Diese sind im word.c . Sie sind die wichtigsten Funktionen, weil sie prüfen, ob die „eingeschriebene“ Wörter wirkliche Wörter sind. checkDictionary läuft durch das ganze „Wörterbuch“ 2of4brif.txt und kompariert die Wörter.

```
void calculateScore(t_board board[15][15], int *score, char *score_str, t_Stones *Stone);
```

Schließlich diese Funktion zählt die Punkte. Sie läuft durch das ganze Brett und zählt die Punkte der da liegenden Steinen.