TU Hamburg-Harburg – Prozedurale Programmierung
Gruppe #37: Florian J******, Marc L******, Marcel H******, WS 2010/2011

Das C-Projekt:

c37 (bessere Idee?)

Entwurf!

Spezifikation

Das Programm *c37* ist ein kleines Abenteuerspiel. Ohne in einem weiteren Kontext zu stehen, übernimmt man die Kontrolle über eine Spielfigur, die sich durch eine zweidimensionale Ebene bewegen kann. Die Spielwelt wird dabei aus einer Perspektive beobachtet, die senkrecht zur Spielwelt steht. Zur Darstellung der Spielfigur und der Umwelt werden dabei Zeichen aus dem ASCII-Zeichensatz und für die angenehmere Betrachtung und Diversifikation der Spielelemente zusätzlich Farben verwendet. Damit orientiert sich *c37* an der Familie der *Rogue-like*-Spiele. Die Steuerung der Spielfigur erfolgt durch die Tastatur, detaillierte Informationen bezüglich der Interaktion des Spielers mit der Umwelt finden über eine Textausgabe am Spielfeldrand statt. Interaktionen sind beispielsweise das Öffnen einer Tür oder Aufheben eines Gegenstands. *c37* ist grundsätzlich nicht auf bestimmte Plattformen beschränkt,

verwendet jedoch für Tastaturevents und den graphischen Ausgabemodus die *Simple DirectMedia Layer*-Bibliothek, ist also auf deren Verfügbarkeit angewiesen. Das Format, in dem die Karten abgespeichert sind, ist die *JavaScript Object Notation* (JSON). Zum Parsen dieser wird eine frei verfügbare Fremdkomponente von *json.org* verwendet. Der Grund für die Verwendung von JSON liegt darin, dass dieses Format im Gegensatz zu etwa XML wenig Overhead besitzt, einfach per Hand editierbar ist und eine Implementation eines eigenen Text- oder Binärformats einen höheren Zeitaufwand und größere Risiken unentdeckter Sicherheitslücken darstellt.

Design

Programmarchitektur

Programmablauf

- Das Programm erhält als ersten Parameter beim Aufruf den Namen einer Karte, welche sich im Verzeichnis *maps* befindet.
- Fehlermeldungen werden nach *stderr* geschrieben.
- Ist diese nicht aufzufinden, startet das Programm nicht. Andernfalls wird die Kartendatei eingelesen: Die Karte wird im Speicher angelegt, die einzelnen Spielfelder erhalten jeweils die Kartenkomponente (Wände, Böden, Türen, ...), die in der Datei angegeben wurde. Weiterhin wird eine Spielfigur gesetzt und Gegenstände platziert.
- Sollte die Kartendatei nicht richtig ausgelesen werden können, bricht das Programm ab.
- Wenn die Karte komplett konstruiert worden ist, wird sichergesetellt, dass I/O-Operationen wie das Warten auf Tasturevents und die graphische Ausgabe funktionieren. Andernfalls bricht das Programm ab.
- Solange das Programm nicht per ESC-Taste abgebrochen wird, findet ein Loop über relevante Events statt, die Logik des Spiels wird auf vorhandene Komponenten angewendet und die Ausgabe aktualisiert.
- Mit dem Drücken der ESC-Taste wird der Loop abgebrochen und vom Programm genutzte Ressourcen werden freigegeben.

zentrale Komponenten

- Maploader - Das Konstruieren von nutzbaren Karten anhand der Vorgaben in der Kartendatei.
- Logikschleife - Interaktion des Spielers mit der Spielfigur und der Umwelt.
- Ausgabe - Darstellung eines vorgefertigten Feldes mit Zeichen- und Farbinformationen.

Ein- und Ausgabe

- Dateisystem (Eingabe) - Das Erstellen einer Karte anhand einer abgelegten Datei, Benutzung eines JSON-Parsers.
- Tastatur (Eingabe) - Steuerung des Programmablaufs sowie der Spielfigur, Benutzung von SDL.
- Bildschirm (Ausgabe) - Ausgabe des Spielgeschehens, Benutzung von SDL.
- Dateisystem/ostream (Ausgabe) - Fehlermeldungen, die in den *stderr*-Stream geschrieben werden.

Programm-Header

globals.h

Die Headerdatei *globals.h* übernimmt mehrere Aufgaben. Eine davon ist es, alle nötigen System-, Library- und Programm-Header zu laden. Andererseits werden hier interne Konstanten gesetzt, die technische Angelegenheiten betreffen als auch solche, die ein aufwendigeres, externes Konfigurationssystem für die Spiellogik ersetzen. Außerdem werden alle globalen Variablen hier angelegt, sodass diese unabhängig von einer bestimmten Objektdatei nutzbar sind.
Dies ist eine Übersicht von verwendeten internen Konfigurationsparametern:

Konstante	Wert	Zweck
OUTPUT_IN_GLYPHS_X	int	vertikale Anzahl an auszugebenden Zeichen
OUTPUT_IN_GLYPHS_Y	int	horizontale Anzahl an auszugebenden Zeichen
MESSAGE_STREAM_LIMIT	int	max. Anzahl an gespeicherten Nachrichten

Eine Übersicht von semantischen Konfigurationsparametern:

Konstante	Wert	Zweck
VISUAL_SQUARE	int	Kantenlänge des Quadrats (quasi Sichtweite), das bei der Erkundung von unbekannten Bereichen verwendet wird.

memory.h

In dieser Headerdatei werden essentielle Funktionen aus der *stdlib* zur Reservierung von Speicherbereichen gewrappt. Im Falle fehlgeschlagener Speicherallokation wird das Programm beendet!

Listing 1: ex_calloc

```
1 void* ex_calloc(size_t num, size_t size);
```

Verwendung: Reserviert *num* mal Speicher der Größe *size* und setzt alle Bytes auf 0. Schlägt dies fehl, beendet sich das Programm.

Parameter:

- num - Anzahl der Speichereinheiten
- size - Größe der Speichereinheit

Rückgabe: Zeiger eines unspezifizierten Typs auf den allozierten Speicherbereich.

Listing 2: ex_malloc

```
1 void* ex_malloc(size_t size);
```

Verwendung: Reserviert Speicher von der Größe *size*. Schlägt dies fehl, beendet sich das Programm.

Parameter:

- size - Größe des zu reservierenden Speicher in Bytes

Rückgabe: Zeiger eines unspezifizierten Typs auf den allozierten Speicherbereich.

Listing 3: ex_realloc

```
1 void* ex_realloc(void* ptr, size_t size);
```

Verwendung: Vergrößert oder verkleinert den Speicher von *ptr* auf *size* Bytes. Ist *ptr* NULL, verhält sich die Funktion wie *ex_malloc*. Schlägt dies fehl, beendet sich das Programm.

Parameter:

- ptr - Adresse des Speichers, dessen Größe verändert wird.
- size - neue Speichergröße in Bytes

Rückgabe: Zeiger eines unspezifizierten Typs auf den neu- oder reallozierten Speicherbereich.

output_buffer.h

Diese Header-Datei definiert die Kachel für den Ausgabepuffer. Diese ist quasi eine sehr beschnittene Spielkartenkachel, die nur noch Informationen enthält, die für die Ausgabe wichtig sind und unabhängig von Spiellogik und Ausgabegerät formuliert ist.

Listing 4: ex_realloc

```
1 typedef struct BufferTile {
2         char glyph;
3         unsigned long color;
4 } BufferTile;
```

Elemente der Struktur:

- glyph - Zeichen, das dargestellt werden soll.
- color - 32-bit Farbinformation im Format R8G8B8A8

item.h

Hier wird eine Struktur definiert, die Gegenstände in der Welt darstellen können, die entweder herumliegen oder aufgenommen sind. Dieser Header lädt zusätzlich Item-Property-Structs aus dem Verzeichnis *items*.

Listing 5: ex_realloc

```
typedef struct Item {
        char* id;
        unsigned long color;
        int weight;
        int value;
        char* type;
        void* properties;
} Item;
```

Elemente der Struktur:

- id - eindeutiger Name jedes Items in der gesamten Spielwelt
- color - 32-bit Farbinformation im Format R8G8B8A8.
- weight - Gewicht des Gegenstands
- value - Wert des Gegenstands
- type - Präzisierung des Typs, den dieses Item darstellt (z.B. Trank oder Waffe).
- properties - Zeiger auf ein *Property*-struct, das abhänging von *type* erstellt wird.

Listing 6: ex_realloc

```
void spawn_uses_item(Spawn* spawn, Item* item);
```

Verwendung: Führt abhänging vom Handelnden *spawn* und dem benutzten Gegenstand *item* einen Effekt aus.

Parameter:

- spawn - Benutzer des Items
- size - benutztes Item

Rückgabe: -

Aufgaben und Zeitplan

Wer?	Arbeitspaket	Tätigkeiten	Datum
Marcel	Planung	Demos (SDL_ttf, JSON)	18. + 19.12.10