

Prozedurale Programmierung WS 2015/16

Projekt, Woche 1

Teil 1: Spezifikation

a) Bewertungskriterien

(i) Spielregeln

Es gibt mindestens zwei und bis zu sechs Spieler. Jedem Spieler wird ein Paddle auf einer Kante des gleichseitigen n -Ecks zugewiesen, wobei n die Anzahl der Spieler ist; im Sonderfall von nur zwei Spielern werden zwei Kanten eines Rechtecks benutzt. Es gibt einen Spielball, und immer wenn der Spielball das Feld verlässt, bekommt der Spieler, dessen Paddle den Ball zuletzt berührt hat, einen Punkt, und der Ball wird auf die Mitte des Spielfelds zurückgesetzt. Das Spiel wird fortgesetzt, bis ein Spieler eine festgelegte Anzahl an Punkten erreicht hat.

(ii) Eingabe

Die Eingabe erfolgt über zwei Pfeiltasten der Computertastatur.

(iii) Grafikausgabe

Das Programm ist in der Lage, ein bildschirmfüllendes Bild auszugeben, das den momentanen Status des Spiels anschaulich als 2D-Grafik darstellt. Es gibt eine Anzeige, anhand derer die Punktzahl jedes Spielers abzulesen ist.

(iv) Menü

Das Spiel startet in ein Menü, in dem man (a) ein Spiel eröffnen oder einem Spiel beitreten, (b) das Spiel beenden und (c) die Eingabebezuweisungen ändern und die Lautstärke der Tonausgabe einstellen kann.

(v) Tonausgabe

Je nach Einstellung wird ein Ton abgespielt, wenn der Ball auf ein Paddle trifft, der Ball das Spielfeld verlässt oder das Spiel endet.

(vi) Multiplattformunterstützung

Das Programm sollte sowohl unter Microsoft Windows ab Version 7, Ubuntu 14.04 LTS als auch Mac OS X ab Version 10.6 funktionieren.

(vii) Netzwerkunterstützung

Mehrere Spieler können sich zum Spiel innerhalb eines LAN miteinander verbinden.

Tuesday 5th January, 2016

Teil 2: Design

a) Programmstruktur

Wenn das Programm ausgeführt wird, liegt die Kontrolle nach der **Programminitialisierung** in der **Menüschleife**, die auf die Komponenten **Eingabe** und **Ausgabe** zurückgreift. Von da aus kann das Programm entweder beendet werden, oder die Kontrolle wird nach der **Spielinitialisierung** an die **Spielschleife** abgegeben. In der Spielschleife wechselt die Ausführung regelmäßig zwischen den Komponenten **Eingabe**, **Netzwerk**, **Physik** und **Ausgabe**. Desweiteren gibt es eine **Debug**-Komponente, die wichtige Ereignisse in einer Debug-Logdatei hinterlegt.

b) Wichtigste Funktionen

(i) Hauptprogramm

```
int main( int argc, void **argv );
```

(ii) Programminitialisierung

```
int InitializeProgram( int argc, void **argv );  
void CloseProgram( void );
```

(iii) Menü

```
int MenuLoop( void );
```

(iv) Spiel

```
int InitializeGame( void );  
int GameLoop( void );  
void CloseGame( void );
```

(v) Netzwerk

```
int InitializeNetwork( void );  
int Connect( int server,  
             const char * remoteAddress,  
             unsigned short remotePort );  
int ProcessNetwork( void );  
void CloseNetwork( void );
```

(vi) Eingabe

```
int InitializeInput( void );  
const struct InputState GetInputState( void );  
void CloseInput( void );
```

(vii) Physik

```
int ProcessPhysics( const struct GameState *state,  
                   float deltaSeconds );  
struct Vector2D ScaleVector2D( struct Vector2D vector,
```

Tuesday 5th January, 2016

```
                                float          scaling );
struct Point2D AddVectorToPoint2D( struct Point2D point,
                                struct Vector2D vector );
```

(viii) Ausgabe

```
int InitializeDisplay( void );
int InitializeSound( void );
int DisplayGameState( const struct GameState *state );
void CloseDisplay( void );
void CloseSound( void );
```

(ix) Debug

```
int InitializeDebug( void );
void DebugPrint( const char *text );
void DebugPrintf( const char *format, ... );
void CloseDebug( void );
```

c) Zentrale Datentypen

(i) GameState

Stellt den aktuellen Status des Spiels dar.

```
struct GameState {
    int          numPlayers;
    Player *     players;
    Ball         ball;
};
```

(ii) Player

Beinhaltet Informationen zu einem Spieler.

```
struct Player {
    char *      name;
    float       position;
    int         score;
};
```

(iii) Ball

Speichert Position und Bewegungsvektor des Spielballs.

```
struct Ball {
    Point2D     position;
    Vector2D    direction;
};
```

Tuesday 5th January, 2016

(iv) **Point2D**

Beinhaltet einen einfachen 2D-Punkt.

```
struct Point2D {  
    float x;  
    float y;  
};
```

(v) **Vector2D**

Beinhaltet einen einfachen 2D-Bewegungsvektor.

```
struct Vector2D {  
    float dx;  
    float dy;  
};
```

(vi) **InputState**

Speichert Informationen zum aktuellen Zustand von Maus und Tastatur. Um den Tasten eine Bedeutung zu geben, werden in der Eingabekomponente Variablen definiert, die die Indizes der Steuerungstasten im **keys**-Array beinhalten.

```
struct InputState {  
    int      numKeys;  
    int *    keys;  
    Point2D mouse;  
};
```

d) **Gemeinsamer Codestil**

(i) **Einrückung**

Es werden Tabs mit einer Breite von 4 Leerzeichen benutzt.

(ii) **Klammern**

Wo möglich, werden geschweifte Klammern gesetzt (**if**, **else**, Funktionen, Strukturen, **typedefs**, etc.)

```
if ( x ) {  
  
}
```

Das **else**-Statement beginnt in derselben Zeile wie die vorhergehende schließende geschweifte Klammer.

```
if ( x ) {  
} else {  
}
```

Tuesday 5th January, 2016

Ausdrücke in Klammern werden mit Leerzeichen umgeben.

```
if ( x ) {
```

```
}
```

statt

```
if (x) {
```

```
}
```

und

```
x = ( y * 0.5f );
```

statt

```
x = (y * 0.5f);
```

(iii) Gleitkommaliterale

Immer die Genauigkeit eines Literals mit angeben, es sei denn, **double** wird explizit benötigt.

```
float f = 0.5f;
```

statt

```
float f = 0.5;
```

und

```
float f = 1.0f;
```

statt

```
float f = 1.f;
```

(iv) Funktionsnamen

Funktionen beginnen mit einem Großbuchstaben (Einzige Ausnahme ist die **main**-Funktion):

```
void Function( void );
```

In Funktionen, die mehrere Wörter enthalten, beginnt jedes Wort mit einem Großbuchstaben:

```
void ThisFunctionDoesSomething( void );
```

Die Standard-Kopfzeile für Funktionen ist:

```
/*
```

```
=====
```

```
 Funktionsname
```

```
 Zweck
```

```
=====
```

```
*/
```

Tuesday 5th January, 2016

(v) Variablen

Variablennamen beginnen mit einem Kleinbuchstaben.

```
float x;
```

In Variablennamen mit mehreren Wörtern beginnt jedes Wort bis auf das erste mit einem Großbuchstaben.

```
float maxDistanceFromPlane;
```

(vi) typedef

Namen von **typedefs** folgen derselben Konvention wie **variablen**, wobei ihnen ein `_t` an den Namen angehängt wird.

```
typedef int fileHandle_t;
```

(vii) struct-Namen

In Namen von **structs** beginnt jedes Wort mit einem Großbuchstaben.

```
struct RenderEntity;
```

(viii) enum

Namen von **enums** folgen ebenfalls derselben Konvention wie **structs**. Die **enum**-Konstanten werden komplett in Großbuchstaben geschrieben. Mehrere Wörter werden untereinander durch Unterstriche getrennt.

```
enum Contact {  
    CONTACT_NONE,  
    CONTACT_EDGE,  
    CONTACT_MODELVERTEX,  
    CONTACT_TRMVERTEX  
};
```

(ix) Rekursive Funktionen

Namen von rekursiven Funktionen enden in `_r`.

```
void WalkBSP_r( int node );
```

(x) Makros

Makros werden komplett in Großbuchstaben geschrieben. Wörter werden untereinander durch Unterstriche getrennt.

```
#define SIDE_FRONT 0
```

Tuesday 5th January, 2016

(xi) **const**

const soll wo immer möglich genutzt werden:

```
const int *p; // pointer to const int
int * const p; // const pointer to int
const int * const p; // const pointer to const int
```

Nicht benutzen:

```
int const *p;
```

(xii) **struct**

Die Standardkopfzeile für eine **struct** ist:

```
/*
=====

Beschreibung

=====
*/
```

Felder von **structs** folgen derselben Benennungskonvention wie Variablen. Felder sollten eingerückt sein, um ein tabellenähnliches Aussehen zu erreichen.

```
struct Player {
    char *      name;
    Texture2D   texture;
    float       position;
    int         points;
};
```

Das * eines Zeigers gehört zum Typ des Feldes und sollte daher aus Gründen der Lesbarkeit links bleiben.

(xiii) **Dateinamen**

Jede **struct** mit ihren dazugehörigen Funktionen sollte in einer eigenen Quellcodedatei sein, außer es ergibt Sinn, mehrere kleinere **structs** in eine Datei zu gruppieren. Der Dateiname sollte derselbe sein wie der Name der **struct**. Die Dateien für **struct** Winding wären dementsprechend Winding.h und Winding.c.

(xiv) **Sprache**

Der Code wird mit englischen Bezeichnern und Kommentaren geschrieben.