Fabian Schmidt <fa.schmidt@tuhh.de>
Fin Töter <f.toeter@gmail.com>
Thomas Pohl <thomas.pohl@tuhh.de>
Nicolai Frech <nicolai.frech@tuhh.de>

Monday 4th January, 2016

# 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

Die Grafikausgabe ist in der Lage, ein bildschirmfüllendes Bild auszugeben, das den momentanen Status des Spiels für einen Menschen verständlich und anschaulich als 2D-Grafik darstellt. Es gibt eine Anzeige, anhand derer die Punktanzahl 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 Eingabezuweisungen 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 Spiel sollte sowohl unter Microsoft Windows ab Version 7, Ubuntu 14.04 LTS als auch Mac OS X ab Version 10.6.

#### (vii) Netzwerkunterstützung

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

# Teil 2: Design

## a) Programmstruktur

## b) Wichtigste Funktionen

## c) Zentrale Datentypen

## 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, **typedef**s, etc.)

```
if ( x ) {

}
```

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

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

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:

```
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
====================
*/
```

### (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 **`typedef`**s folgen derselben Konvention wie variablen, wobei ihnen ein _t an den Namen angehängt wird.

```
typedef int fileHandle_t;
```

### (vii) `struct`-Namen

Namen von **`struct`**s folgen derselben Konvention wie **`typedef`**s.

```
struct renderEntity_t;
```

### (viii) `enum`

Namen von **`enum`**s folgen ebenfalls derselben Konvention wie **`typedef`**s und **`struct`**s. Die **`enum`**-Konstanten werden komplett in Großbuchstaben geschrieben. Mehrer Wörter werden untereinander durch Unterstriche getrennt.

```
enum contact_t {
    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
```

### (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 **`struct`**s folgen derselben Benennungskonvention wie Variablen. Felder sollten eingerückt sein, um ein tabellenähnliches Aussehen zu erreichen.

```
struct player_t {
    char *      name;
    texture2D_t 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`** mit einem Großbuchstaben am Anfang, aber ohne das `_t`. Die Dateien für **`struct`** `winding_t` wären dementsprechend Winding.h und Winding.c.

### (xiv) Sprache

Der Code wird mit englischen Bezeichnern und Kommentaren geschrieben.