Teil 2: Design

1 Allgemeine Programmstruktur

i) Aufbau des Programmes

Sobald das Spiel initialisiert wurde, liegt die Kontrolle in der Schleife von `draw_Game`, der Funktion, die die Eingaben verwaltet und je nach aktuellem Zustand das Spiel zeichnet bzw. die zugehörigen Funktionen dazu aufruft. Das Spiel wird beendet, sobald sobald die Schleife durch den Benutzerinput unterbrochen wird. Innerhalb dieser Funktionen werden wiederum Funktionen aufgerufen, um die einzelnen Oberflächen des Spiels zu zeichnen, den Benutzerinput zu verwalten und die Spielregeln und die Physik des Balles umzusetzen.

- includes (`SDL.h`, `audio.h` etc., die Initialisierung der Structs und die Prototypen der Funktionen befinden sich in einer eigenen Library `main.h`)
- Initialisierung der Globals
- Definition der Funktionen
- `main`-Funktion

ii) Die wichtigsten Funktionen im Überblick

Displayfunktionen

- `void write_Text(SDL_Renderer *renderer, char *text, int posx, int posy, int size).`
- `char* format_Time(int t).`

Funktionen zur Darstellung der Menüs

- `void draw_Menu(void).` Zeichnet das Menü
- `void draw_PlayerSelection(void).` Spieleranzahl auswählen
- `void draw_ModeSelection(void).` Spielmodus auswählen, wirkt sich aus auf draw_Field (Punkteanzeige, Abbruchkriterium), set_Game_Vars (Leben der Spieler werden auf 0 oder 10 gesetzt), draw_GameOver, respawn_Ball (Leben der Spieler werden hier gezählt, je nach Modus auf- oder absteigend)
- `void draw_Rules(void).` Gibt eine Übersicht über die Spielregeln

Funktionen zur Spieldarstellung

- `void spawn_Perc(struct Perc *p).` Erzeugt Percs, zufällige Generierung eines Standortes
- `void despawn_Perc(struct Perc *p).` Ausgeführt bei Abschuss des Percs, Perc verschwindet vom Spielfeld (mit Ausnahme der Barriere), eine neue Spawnzeit wird festgelegt
- `void draw_GameOver(void).` Gewinnernotifikation (Abhängig vom Abbruchkriterium des Spielmodus)
- `void draw_Field(void).` Zeichnet das Spielfeld in Abhängigkeit des Spielmodus, den Ball, die gespawnten Percs, die Spieler ect. nach den in set_Game_Vars vorher festgelegten Positionen (bzw. Spawnzeiten für die Percs)

Physikalische Funktionen

- `float checkCollision(struct Object *o1, struct Object *o2, int flag).` Überprüft ob Ball und Spieler oder Ball und Barrier ect. zusammenstoßen. Parameter sind die beiden zu vergleichenden Objekte (Rechteck, Reckteck oder Rechteck, Linie) und ein Integer, welcher angibt worauf die Objekte miteinander verglichen werden sollen.
- `void change_Ball_Direction(float angle).` Verändert die Richtung des Balles nach Kollision mit einem Objekt
- `void move_Ball(void).` Updated die Position des Balles (Bewegung), ruft `checkCollision` für alle nötigen Objekte und die entsprechenden Funktionen für die verschiedenen Events auf (Respawn des Balles, Abspielen der Kommentare und Punkteverwaltung bei Tor eines Spielers, Aktivierung der Percs, Richtungsänderung des Balles bei Treffen auf Spielfeldgrenze o. Ä.)

Allgemeine Funktionen

- `void move_player(struct Player *p, int dirx, int diry).` Updated die Positon der Spieler
- `void respawn_Ball(struct Player *p).` Respawnt den Ball mit zufälliger Bewegungsrichtung, Punkteverwaltung (Welchem Spieler werden Punkte abgezogen bzw. angerechnet?), spielt die Kommentare beim Punktgewinn mit Hilfe der audio-Library ab
- `void updateEntities(void).` Positionen der Spieler und des Balles werden durch Aufruf der Funktionen `move_Player` bzw. `move_Ball` aktualisiert
- `void draw_Game(void).` Verwaltet die verschiedenen zu zeichnenden Ebenen des Spielmenüs, Tastatureingaben zum Navigieren werden eingelesen/verwaltet
- `void set_Game_Vars(void).` Stellt die Position, Geschwindigkeit und Leben der Spieler in Abhängigkeit von Modus und Spielerzahl ein, legt die anfängliche zufällige Spawnzeit der Percs fest
- `void init_Game(void).` Initialisiert das Spiel, startet die SDL engine, die audio engine, die Hintergrundmusik und die Funktion `draw_Game`, beendet SDL und audio engine
- `int main().` Das Gameobjekt und die verschiedenen Percobjekte werden hier initialisiert (Farbe, Typ)

iii) Zentrale (strukturierte) Datetypen

Idee: Das Programm pseudo-objektorientiert aufzubauen. D. h. wir arbeiten mit Objekten, aus denen wieder andere Objekte zusammengbaut werden, z. B. die Spieler (`struct Player`) und der Ball (`struct Ball`), welche im Kern aus `struct Object` aufgebaut werden. Dies gewährleistet, dass die Objekte miteinander verglichen werden können. Am Schluss wird alles in einem großen Objekt (`struct Game`) vereinigt.

- **struct Dim**
  Beinhaltet die Höhe und Breite des Spielfeldes

```
struct Dim{
  int width, height;
};
```

- **struct Object**
  Anonymes Objekt als Grundlage für Player, Ball und Percs

```
struct Object{
  float posx1, posy1, posx2, posy2;
};
```

- **struct Player**
  Beinhaltet die Informationen eines Spielers

```
struct Player{
  struct Object o;
  int size, lifes, speed;
};
```

- **struct Ball**
  Beinhaltet die Informationen des Balles

```
struct Ball{
  struct Object o;
  float speed, angle;
  struct Player *lastHit;
};
```

- **struct Perc**
  Beinhaltet die Informationen eines Percs

```
struct Perc{
  struct Object o;
  int spawnat,type,r,g,b;
};
```

- **struct Neck**
  Beinhaltet die Positionen der Grenzen des maximalen 8-Ecks (Oktagon)

```
struct Neck{       \\n-Eck
struct Object l1;
  struct Object l2;
  struct Object l3;
  struct Object l4;
  struct Object l5;
  struct Object l6;
  struct Object l7;
  struct Object l8;
};
```

- **struct Game**
  Fasst alle Structs zusammen und vereinigt sie in ein Gamestruct, der alle Informationen beinhaltet, die das Spiel benötigt

```
struct Game{
  SDL_Window *window;
  SDL_Renderer *renderer;
  struct Ball b;
  struct Player p1;
  struct Player p2;
  struct Player p3;
  struct Player p4;
  struct Dim dim;
  struct Neck neck;
  int state, player, mode, timestart, timeend, invert;
  struct Perc e1;
  struct Perc e2;
  struct Perc e3;
  struct Perc e4;
  struct Perc e5;
  struct Perc e6;
  struct Player *lastGoal
};
```