1 Planung und Konzeption

Um später die Spiellogik zu implementieren musste zunächst der grobe Spielablauf analysiert werden. Zum Spielablauf gehören: Das Spiel starten, pausieren, gewinnen, verlieren und das Spiel neu starten. Daraus ergibt sich der folgende Automat:

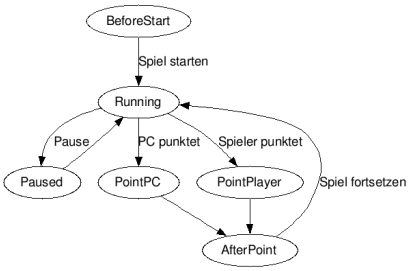

Abb. 1: Spielzustands-Automat

Zur Umsetzung des Spiels müssen im wesentlichen Die vier Wände, der Ball und die Schläger gezeichnet werden. Des weiteren müssen das aktuelle Level (In Form von Schädeln) und die aktuelle Anzahl der Extra-Leben (als Herzen) angezeigt werden. Wird das Spiel gestartet, kann der Schläger mit der Maus gesteuert und der Ball mit Enter gestartet werden. Berührt der Ball einen Schläger muss je nach Geschwindigkeit des Schlägers die Flugbahn des Balls angepasst werden. Ebenfalls wird bei jeder Kollision ein zufälliger Sound abgespielt.
Verlässt der Ball das Spielfeld auf der Seite des Gegners, gewinnt der Spieler und der Schwierigkeitsgrad erhöht sich. Dies zeigt sich an einem schnelleren Ball und einem besser reagierenden Computergegner.
Verlässt der Ball das Spielfeld auf der Seite des Spielers, so wird ein Extra-Leben angezogen. Erreicht die Anzahl der Leben null, so gilt das Spiel als verloren und beginnt von vorne.

2 Implementierung

Zunächst wurden für alle spielrelevanten Objekte Klassen erstellt. Dies sind: Game (das Spiel), Paddle (Schläger), Ball, Wall (Wände), Heart (Herzanzeige) und Skull (Levelanzeige). Die Klassen Wall, Paddle, Heart und Skull sind hier eher passiv einzuschätzen. Diese besitzen im wesentlichen nur eine Grafikroutine, um die Objekte im Spiel darzustellen. Die Klasse Paddle besitzt des weiteren noch die Methode AI_Act. Diese ist dafür verantwortlich, den Schläger automatisch zum Ball zu führen, was für den Computergegner benötigt wird.
Die Klasse Ball enthält zusätzlich zur Grafikroutine noch Methoden um Kollisionen zu simulieren. Die Funktionsweise des Balls lässt sich wie folgt erklären: Der Ball besitzt einen Vektor direction, in welchem die aktuelle Geschwindigkeit im dreidimensionalen Raum gespeichert wird. In jedem Frame wird diese Geschwindigkeit auf den Ball addiert, so dass dieser seine Position verändert. Anschließend wird geprüft, ob der Ball Wände oder Schläger berührt. Dabei wird der Ball wie eine Box behandelt, um so die Kollisionsabfrage trivial zu halten. Kollidiert der Ball mit etwas, so wird je nach getroffenem Objekt eine Komponente des Vektors direction negiert. Trifft der Ball zum Beispiel die linke oder rechte Wand, so wird die X-Komponente des Vektors negiert, so dass der Ball nun in die entgegengesetzte Richtung abprallt. Wird erkannt, dass der Ball das Spielfeld bei einem der Spieler verlässt, so wird in den jeweiligen Zustand PointPC oder PointPlayer gewechselt. Das Level wird dann erhöht bzw. die Leben verringert.

TODO:HIER FLUGBAHN

Eine weitere wichtige Klasse ist die Klasse GameUtils. In dieser statischen Klasse werden alle globalen

Variablen für diverse Spielinformationen gespeichert. Dort sind zum Beispiel alle Zustände des Automaten definiert. Ebenfalls finden sich dort diverse Hilfsfunktionen, welche in verschiedensten Bereichen des Spiels genutzt werden.
Zur Verwendung von Sounds wurde die quelloffene Bibliothek "Tinysound"[1] genutzt. Diese ist im Package kuusisto.tinysound zu finden. Tinysound stellt die beiden Klassen Music und Sound zur Verfügung, welche extrem einfachen Umgang mit diesen ermöglichen. Zur Implementierung der Spielsounds wurden zusätzlich noch zwei Threads erstellt, in welchen Sound und Musik abgespielt werden. Da das Programm bis zum Ende des Sounds wartet, bis es weiter ausgeführt wird, war dies unumgänglich. Ebenfalls wurde ein Soundboard erstellt. Bei diesem können mehrere Sounds mit verschieden Kontexten verbunden werden. Soll in einem gewissen Kontext (zum Beispiel die Kollision des Balles) ein Sound abgespielt werden, so wird zufällig zwischen allen zum Kontext passenden Sounds gewählt.

Das HUD wurde implementiert, indem die jeweiligen Objekte sichtbar oder unsichtbar gemacht werden. So sind zum Beispiel die Level auf zehn limitiert. Das heißt, dass sich im Array lvlGui in der Klasse GameUtils zehn Skull-Objekte befinden. Ändert sich das Level auf einen anderen Wert, so werden dementsprechend viele Skull-Objekte sichtbar gemacht. Dies wird in der statischen Methode setLevel umgesetzt. Die Lebensanzeige funktioniert hier analog mit drei Herz-Objekten.

[1] https://github.com/finnkuusisto/TinySound

3 Bedienung

- ESC - Spiel beenden
- R - Spiel zurücksetzen
- P - Spiel pausieren/fortsetzen
- Enter - Spiel starten
- Maus - Schläger bewegen[2]
- Pfeiltasten Links/Rechts - Spielfeld drehen
- M - Gitternetzmodell An/Aus
- Q - Leben um 1 erhöhen
- A - in AI-Only Modus zum Testen der AI wechseln
- E - Level um 1 erhöhen
- T - Texturen Aus-/Einblenden
- C - Backface culling aktivieren

[2] Alle folgenden Angaben sind ausschließlich zum Testen gedacht