Hängebauchschwein-Simulator Infoblatt

# Spezifikation

* Platformer
* Punktesystem auf Basis von Gewicht / Futter, zu sammeln im Level
* Level (min 3)
* Gegner, die auch angreifen (KI)
* Menü
* Aufzeichnung von bisherigen Scores -> Rekord / Highscore
* 2D Grafik
* Animationen bei Gegnern und dem Hauptcharakter

Ziel unseres Projekt ist es ein Platformer-Spiel mit 2D-Grafik zu erstellen. Dazu gehört, dass der Spieler mit der Umwelt kollidieren kann. Das Spiel soll ein Punktesystem enthalten, wobei der Spieler mit seinem Charakter beim durchlaufen der Levels Futter und ähnliches sammeln kann, um so sein Hängebauchschwein schwerer zu machen. Am Ende eines jeden Levels soll dann das Gewicht des Hängebauchschweins ausgewertet werden.

Dazu soll das Spiel außerdem vergangene Ergebnisse bzw. Rekorde dauerhaft speichern, um so zu ermitteln ob der Spieler seinen bisherigen Rekord brechen konnte.

Außerdem wollen wir es uns zum Ziel setzen mindestens 3 verschiedene Level für das Spiel zu erstellen.

Daneben soll es ein Menü geben, in dem es möglich ist das Spiel zu starten, Level zu wählen und evtl. auch Einstellung zu ändern und Highscores einzusehen. Das Spiel sollte von dort aus auch wieder zu verlassen sein.

Wir streben zudem an, eine Gegner KI zu erstellen, die innerhalb des Levels in bestimmten Situationen z.B. auf Basis eines Triggers oder der Entfernung zu Spieler diesen angreift.

Auch Animationen, sowohl beim Charakter des Spielers, als auch bei den KI Gegner wollen wir umsetzen.

# Design

Der **Hängebauchschwein-Simulator** soll über seine Laufzeit hinweg in einer Schleife laufen. Innerhalb dieser Schleife soll es Funktionen für die jeweiligen **Game States** geben. Von diesen soll es in etwa für das Menü, das eigentliche Spiel,, den Startscreen, sowie den Highscore Bildschirm jeweils einen eigenen geben. Wir planen eventuell auch Stati auf einer darunter liegenden Ebene einzubauen, also z.B. menuStates, um die einzelnen Untermenüs darzustellen.

Jeder dieser States soll die Funktionen **Input()**, **Update()**, und **GraphicUpdate()** beinhalten. Input() soll dabei eventuelle Events, die aus den registrierten Inputs hervorgehen an Update zurückgeben, weshalb wir die Funktionen wahrscheinlich auch in der oben genannten Reihenfolge ausführen wollen. Diese könnten dann als Argumente an Update() gegeben werden. Alternativ würden wir das vielleicht aber auch global lösen.

Die Grafikumsetzung ist mit SDL geplant, evtl. könnte aber auch eine Alternative gewählt werden.

Folgende Datentype planen wir derzeit unter anderem umzusetzen(Angabe schematisch):

* Score {time, weight}
* Level {landscape, scores, unlocked}
* Player {position, weight, health}
* Enemy {name, position, strength, health, speed, viewDistance}
* Position {x, y}
* Food {name, position, weight}

Auch folgende wollen wir neben anderen schreiben:

* weight eat(food);
* void attack(\*player, \*enemy)