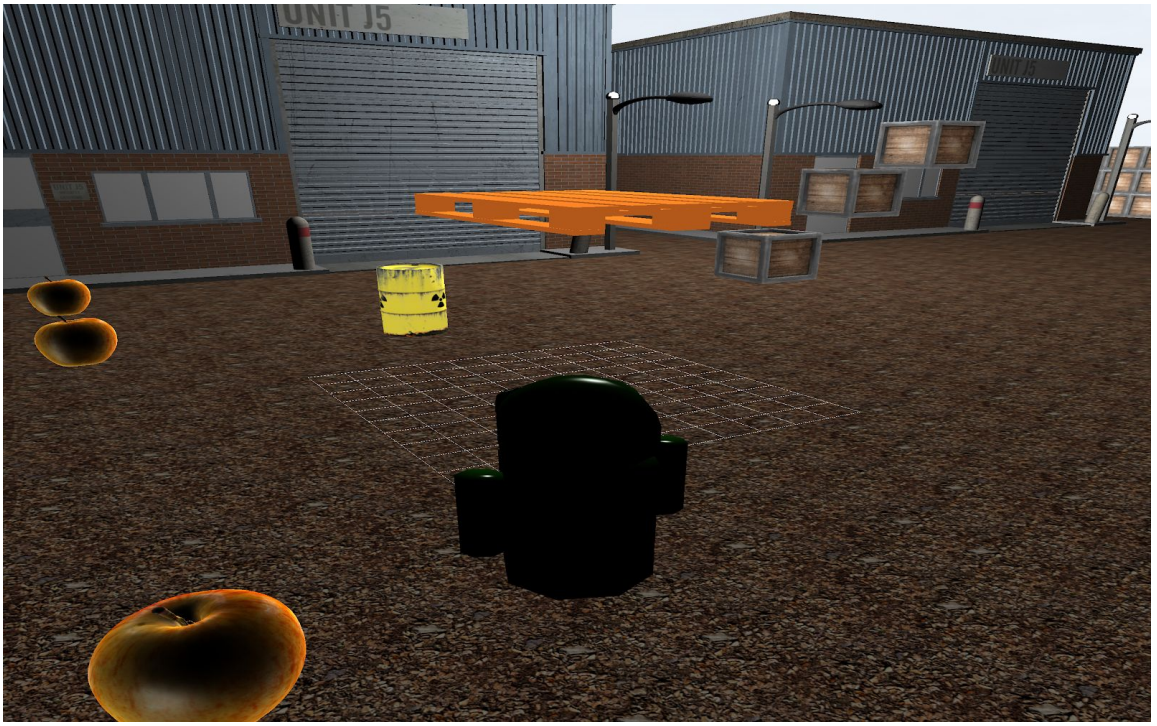
HOCHSCHULE OSNABRÜCK

UNIVERSITY OF APPLIED SCIENCES

Projektbericht Computergrafik



Sandra Tieben Mat-Nr.: 440471

sandra.tieben@hs-osnabrueck.de

Lukas Völler Mat-Nr.: 620445

lukas.voeller@hs-osnabrueck.de

Abgabedatum 05.03.2018

Inhalt

Spielidee

Bei dem Spiel handelt es sich um ein klassisches Jump ‘n’ Run in Third-Person-Perspektive (Verfolger-Ansicht), bei dem Äpfel eingesammelt werden. Das gesamte Spiel befindet sich in einer Ebene, auf der sich die Spielfigur durch Laufen, Drehen und Springen bewegen kann.

Neben den sammelbaren Gegenstände gibt es Giftfässer. Berührt die Spielfigur diese, stirbt sie und das Spiel kann neu gestartet werden.

Spannend wird das Spiel dadurch, dass die Spielfigur die Äpfel teilweise nur erreichen kann, wenn sie auf Elemente springt oder sich über schwebende Paletten zu diesen bewegt.

Das Spiel ist beendet, wenn alle Äpfel eingesammelt wurden.

Bedienung

Element	Tastenzugriff	Beschreibung
Startbildschirm bzw. beim Tod der Spielfigur	ENTER	Spiel (neu) beginnen
Hilfe-Menü	ESC	Öffnen/Schließen des Hilfe-Menüs
Hilfe-Menü	P	Schließen des Spiels
“Gewonnen”	P	Schließen des Spiels
Spielfigur bewegen	W bzw. ↑	Vorwärts laufen
	A bzw. ←	Drehung gegen den Uhrzeigersinn
	D bzw. →	Drehung mit dem Uhrzeigersinn
	S bzw. ↓	Rückwärts laufen
	Leertaste	Springen
	W + SHIFT bzw. ↑ + SHIFT	Schnell vorwärts laufen
	S + SHIFT bzw. ↓ + SHIFT	Schnell rückwärts laufen

Stärken und Schwächen

Einige Punkte in der Arbeit sind nicht perfekt gelöst. Diese werden in diesem Abschnitt kurz dargestellt.

Szenenmanagement

Zur Speichereffizient sowie Aufbau der Spielszenerie liest die Klasse `scene.cpp` die verwendeten Modelle inklusive Lichter und beweglicher Objekte ein und positioniert diese in der gewünschten Skalierung in der Szene. Jedes Modell wird dabei als SceneNode angelegt. Die Straßenlaternen in der Szene werden durch Spotlichtquellen ergänzt.

Hierdurch ergibt sich einerseits ein übersichtlicher Aufbau der Szene in der Datei `scene.osh` sowie eine erhöhte Speichereffizienz, da mehrfach vorliegende Modelle nur einmal erzeugt werden.

In Zusammenhang mit den Szenenmanagement wird auch die BoundingBox der vorliegenden Objekte skaliert, was für eine funktionstüchtige Kollisionserkennung relevant ist.

Spielkonzept

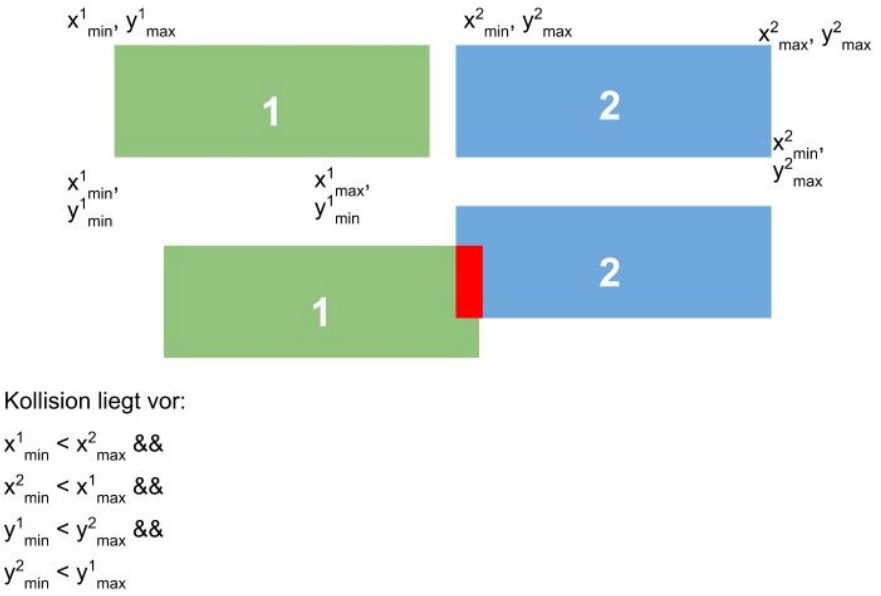
Auch gut gelöst ist das Spielkonzept. Es gibt ein Startmenü, ein Hilfemenü, welches während der Bedienung aufgerufen werden sowie eine Ansicht, welche beim Beenden des Spiels dargestellt wird. Das Spiel liegt als Vollbildversion vor und der Mauszeiger wird ausgeblendet. Somit liegt tatsächlich ein voll funktionstüchtiges Spiel vor.

Zusätzlich hierzu wurde eine relativ weitläufige Spielwelt mit vielen Elementen angelegt.

Kollisionserkennung

Ein wichtiger Punkt in den Jump’n’Run ist eine funktionierende Kollisionserkennung sowie darauf aufbauende Kollisionsbehandlung, damit die Spielfigur sich durch das Spiel über Hindernisse bewegen und Gegenstände einsammeln kann.

Hierzu wird eine 3D-Kollisionserkennung benötigt, welche in der folgenden Abbildung vereinfacht als 2D-Kollisionserkennung dargestellt wird.

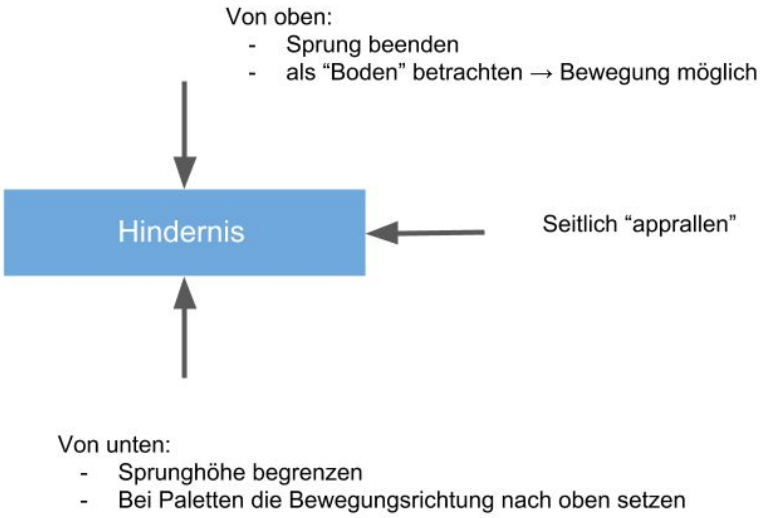


Kollisionserkennung (2D)

Nachdem eine Kollision erkannt wurde, erfolgt eine Kollisionsbehandlung. Diese ist individuell angepasst, je nach Art des Gegenstands.

Handelt es sich bei dem Kollisionselement um einen sammelbaren Gegenstand, so wird dieser mit einer kurzen Animation eingesammelt. Bei tödlichen Hindernissen stirbt die Spielfigur und ein neues Spiel kann erfolgen.

Interessanter sind die fliegenden Paletten sowie sonstige Hindernisse, da bei diesen die Kollisionsbehandlung gesondert nach Art der Kollision erfolgt. Je nachdem ob die Spielfigur von oben, unten oder seitlich in die Kollision eintritt, wird individuell reagiert, wie in der folgenden Abbildung dargestellt wird.



Kollisionsbehandlung

Nachteilig an der aktuellen Lösung ist die Ineffizient, da jedesmal alle Elemente auf Kollisionen geprüft werden. Liegt eine zu große Zeitspanne zwischen zwei Frames, so bewegt sich die Spielfigur zu tief in das Element, mit dem die jeweilige Kollision stattfindet.

Zur Behebung dieser Probleme kann einerseits das Kollisionsbehandlung angepasst werden. Zusätzlich kann die Zeitspanne zwischen zwei Frames verbessert werden, indem die Kollisionserkennung für weniger Elemente durchgeführt wird. In diesem Spiel können die sichtbaren Grenzen des Spielfelds bei der Kollisionserkennung ausgelassen werden, da diese Bereiche für die Spielfigur nicht zugänglich sind.

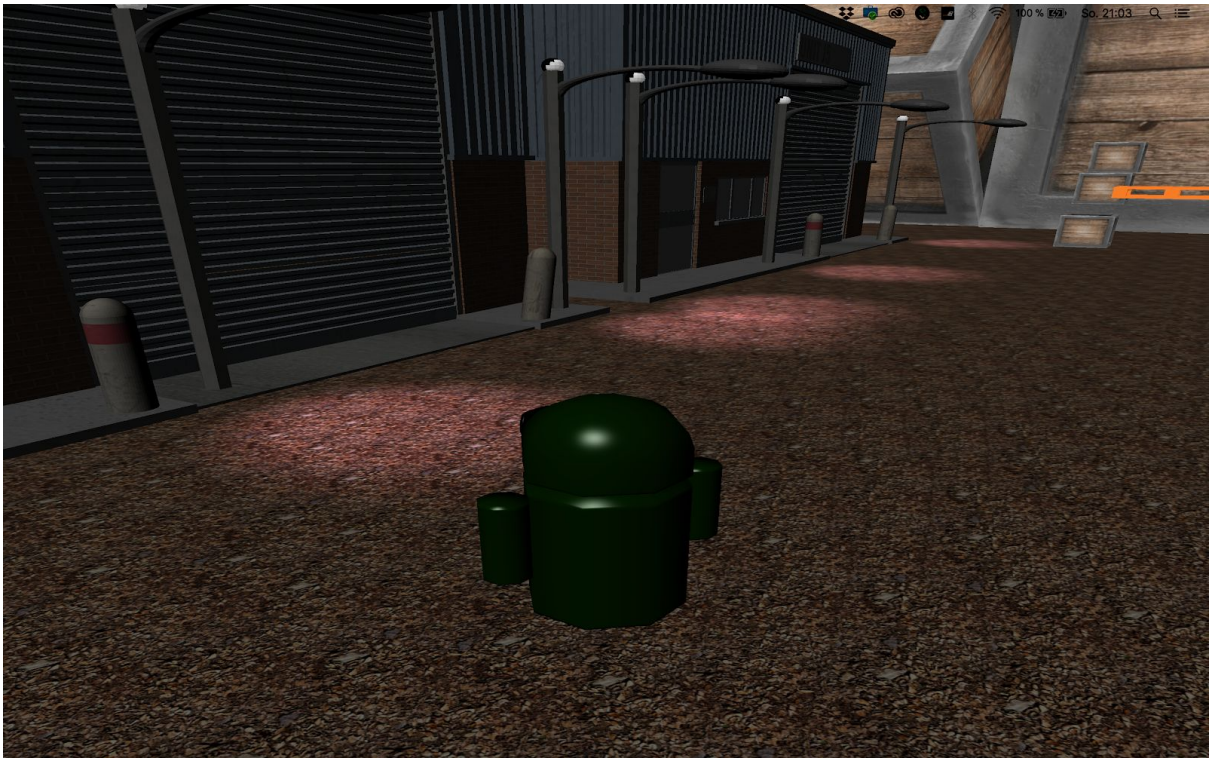
Ein weiterer großen Nachteil bei der Kollisionserkennung ist, dass stets nur auf eine Kollision geprüft wird, wodurch es bei mehreren Kollisionen gleichzeitig unerwünschte Effekte geben kann.

Bewegende Plattformen

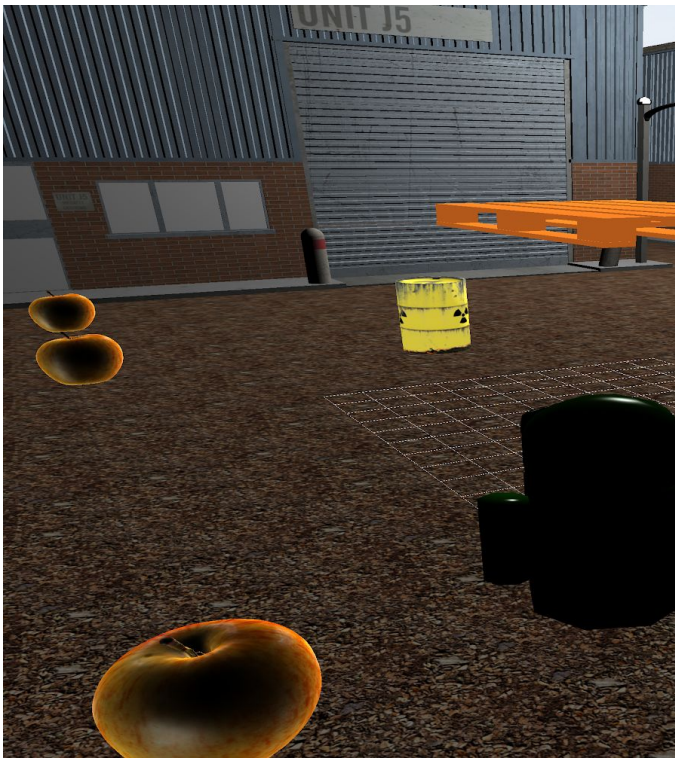
Damit es sich nicht um ein statisches Spiel handelt, wurden Paletten zugefügt, welche sich in vertikale Richtung bewegen. Erfolgt eine Kollision mit diesen Paletten, wobei die Spielfigur sich unterhalb der Palette befindet, bewegt sich die Palette (wieder) aufwärts.

Shader

In der Szenen wird mit insgesamt sieben Lichtquellen gearbeitet. Neben einer direktionen Lichtquelle, welche die Grundhelligkeit erzeugt, werden vorliegende Straßenlaternen durch Spotlichtquellen ergänzt.



Die Äpfel, die in dem Spiel eingesammelt werden, werden über den Outline-Shader hervorgehoben, wie auf der folgenden Abbildung erkennbar. Statt eines eigenen Fragment-Shaders wäre für diese Hervorhebung wäre es sinnvoll die bisherigen Shader durch Post-Processing-Effekte im letzten Schritt der Rendering-Pipeline (Rasterization) zu ergänzen.



Aufbau und Strukturierung

Es wurde eine Strukturierung des Quellcodes in verschiedene Module durchgeführt, wodurch der Quellcode lesbarer und besser wartbar wird. Bei dem Versuch eine sinnvolle Aufteilung zu erzielen, wurden leider unnötige Abhängigkeiten zwischen verschiedenen Klassen erkannt, welche nicht komplett aufgehoben wurden. Hier sind noch Verbesserungen möglich.

Hinweise zur Umsetzung

Zusätzlich zu den im vorherigen Abschnitt beschriebenen Stärken und Schwächen des Spiels werden in diesem Abschnitt einige zusätzliche Hinweise zur Umsetzung gegeben.

Kamera

Bei der Kamera handelt es sich um eine 3rd-Person-Kamera, welche sich mit der Spielfigur mitbewegt (Translation und Rotation).

1. Matrix der Spielfigur
2. Verschiebung der Matrix, damit Figur sichtbar
3. Rotation der Matrix, damit die Figur von hinten gezeigt wird

Aufbau

Zum vereinfachten Verständnis wurden die in dem Spiel verwendeten Klassen gruppiert (jeweils .cpp und .h).

```
Buffer
      VertexBuffer
      FrameBuffer
      IndexBuffer
Entity
      Character
      Coin
      MovingItem
SceneManagement
      Scene
      SceneNode
Graphic
      AABB
      Camera
      DebugRenderer
      EgoCam
      Lights
      Texture
Math
      Matrix
      Vector
Model
      Model
```

```
      …
Shader
      PhongShader
      OutlineShader
      …
Util
      ..
Game
      ..
main
Application
Constants.h
```

Quellen

1. Android: https://www.turbosquid.com/FullPreview/Index.cfm/ID/757160
2. Apple: https://www.turbosquid.com/FullPreview/Index.cfm/ID/549455
3. Factory: https://www.turbosquid.com/FullPreview/Index.cfm/ID/762121
4. Pallet: https://www.turbosquid.com/FullPreview/Index.cfm/ID/751772
5. Plane: Praktikum
6. Skycylinder: Praktikum
7. Woodcube: https://www.turbosquid.com/FullPreview/Index.cfm/ID/861657
8. Barrel: https://www.turbosquid.com/FullPreview/Index.cfm/ID/1014400
9. Streetlamp: https://free3d.com/3d-model/street-light-lamp-61903.html