

Documentation RoomMaze

Das Spiel RoomMaze wird durch Maximilian Kinzel und Lukas Lidauer entwickelt. Anfänglich wurde das OpenGL Framework erstellt, wo simple Geometrien mit verschiedenen Texturen/Maps erstellt werden konnten und die Kamera frei im Raum bewegt werden konnte. Das Programm wurde um einen Wavefront Object Reader erweitert, da wir nicht von der Funktionalität von Assimp abhängig sein wollten. Die Funktionalität unseres Frameworks und des Object Reader ist laufend um diverse Maps gestiegen. Als das GrundFramework stand, wurde PhysX in das Projekt eingebunden und die Klassenstruktur dementsprechend etwas verändert. Zeitgleich wurde auch das HUD entwickelt (Implementierung des Font-Frameworks Freetype). Später wurde noch für den 3D Audio OpenAL eingebaut. Über den gesamten Zeitraum dieser LVA wurde ein Model in Blender entworfen woran wir insgesamt etwa 300 Stunden saßen. Alle im Spiel befindlichen Models sind eigenständig entwickelt worden. Die Texturen stammen zur Gänze aus der im TUWEL verlinkten Texturen-Datenbank.

How to start

Vor dem ersten Start auf einem PC bitte unbedingt OpenAL 1.1 installieren, indem die mitgelieferte Datei **oalinst.exe** ausgeführt und installiert wird. Einstellungen können über die Datei **assets/settings.ini** gesetzt werden.

Features (Gameplay)

- Playable: Das Spiel ist gänzlich spielbar. Es gibt keine Möglichkeit, im Spiel hängen zu bleiben.
- 3D Geometry: Es wurde eigene 3D Geometrie modelliert und in das Spiel importiert.
- Win/Loose Condition: Wenn man 5 Sekunden im Dunkeln wandert, wird man von einem Geist attackiert. Wenn man es schafft mit dem Aufzug an die Oberfläche zu kommen, ist das Spiel gewonnen.
- Intuitive Controls: Es gibt verschiedene Controls für das Bewegen (WASD), Laufen mit gedrückter Shift-Taste, Interagieren (Linke Maustaste), Alle Items erhalten (F10), Glow Effekt ein-/ausschalten (B), Batterie einlegen (Leertaste).
- Intuitive Camera: Die Kamera bewegt sich abhängig von der Maus und von der Bewegung des Spielers.
- Textures: Es gibt viele verschiedene Texturen mit vielen verschiedenen Maps (Specular, Normal, Alpha, Diffuse)
- Moving Objects: Türen, Aufzug, Schlüssel, Aufzugsknopf werden mit einem eigenen KeyFrame system linear animiert. Diese Animationen sind aktuell noch hardcoded.

- Adjustable Parameters: In der Datei „assets/settings.ini“ können alle gewünschten Einstellungen vorgenommen werden.
- Physics Engine: Es wurde PhysX 4.1 eingebunden und auch für viele Dinge benutzt (Raycast, Collision Detection, ..)
- Heads-Up Display: Zeigt Start, Ende und Gameover Screen an. Außerdem wird während des Spiels der Batteriestatus, das Inventar und ein Infotext des Fokusierte Gegenstandes angezeigt.

Features (Effekte)

- CPU Particle System: Das Partikel-System wurde nach dem Tutorial auf learnopengl.com implementiert. Partikel werden außerhalb des Shaders erstellt und simuliert und anschließend gerendert unter Verwendung von Instancing. Beim Interagieren mit Objekten werden Partikel gespawnt sowie im ersten Raum in einem Schaltkasten.
- Hierarchical Animation: Für die Animationen haben wir ein eigenes Keyframe-System entworfen, welches zwischen einzelnen Positionen interpoliert.
- Specular Map: Wurden wie üblich im Fragment-Shader anstatt einer Specular Color mit einberechnet.
- Simple Normal Mapping: Für diesen Effekt haben wir die Tangenten beim Laden der Meshes errechnet und diese anschließend in den Shader geladen. Dort wurde diese Information zusammen mit der jeweiligen Normal Map verwendet um eine realistische Licht-Simulation nach zu ahmen.
- Bloom/Glow: Dieser Effekt wurde nach einem Tutorial eines Youtubers entwickelt (ThinMatrix). Hierfür wurden Framebuffer verwendet. Zuerst wird die gesamte Szene auf einen Framebuffer gerendert. Der zugehörige "phong.frag"-Shader liefert 2 Outputs, einmal das üblich gerenderte Bild und ein Bild was sehr helle Bereiche enthält. Dieses zweite Bild wird durch einen Gauß-Filter bearbeitet und anschließend mit dem ersten Output kombiniert. Ein Beispiel für diesen Effekt befindet sich im Lift (später mehr dazu).

Story

In unserem Spiel startet der Protagonist in einem düsteren Raum. Anbei eine Taschenlampe, welche es überhaupt erst möglich macht etwas zu sehen. Das Ziel des Spieles ist es die Umgebung zu erkunden und irgendwie zu versuchen aus diesem heruntergekommenen Labyrinth aus Räumen zu entkommen. Gleichzeitig muss man darauf achten, dass man immer genug Batterien im Inventar hat. Diese sind random in der Gegend versteckt. Genauso wie Items, welche gefunden werden müssen um das Ziel des Spiels zu erreichen.

Erste Aufgabe ist es einen Schlüssel in den ersten 3 Räumen zu finden (dieser wird random versteckt). Wurde er gefunden, scheint er im Inventar auf und lässt sich an den beiden Stahltüren einsetzen. Sind diese geöffnet, ist die zweite Aufgabe einen Widerstand zu finden, welcher den Schaltkasten neben den geschlossenen Aufzugtüren repariert. Wurde dieser gefunden, kann er in jenem Schaltkasten eingesetzt werden und die Türen öffnen sich. Der Aufzug kann betreten werden und der Spieler erfährt, dass ein fehlender Knopf (welcher benötigt wird um den Aufzug in Betrieb zu nehmen) nicht weit von der aktuellen Position versteckt sein kann. Der Knopf kann in den nun erreichbaren Raum auf der gegenüberliegenden Seite des Aufzugs gefunden werden (auf dieser wird wieder per Zufall versteckt). Wurde dieser gefunden, kann er im Aufzug eingesetzt und dieser in Betrieb genommen werden.

Ein kleiner Cheat stellt die F10-Taste dar, welche einem alle nötigen Items ins Inventar setzt und das Ziel des Spiels, ohne erst nach diesen suchen zu müssen, erreichen lässt.

Additional Libraries

PhysX 4.1:

<https://gameworksdocs.nvidia.com/PhysX/4.1/documentation/physxguide/Index.html>

FreeType: <https://www.freetype.org/>

OpenAL 1.1: <https://www.openal.org/downloads/>

FreeALUT: <https://github.com/vancegroup/freealut>

Programming Tricks

Wir haben einen eigenen OBJ Loader programmiert, welcher sich verglichen mit herkömmlichen Loadern sehen lassen kann. Die Klasse ist OBJReader.cpp

Außerdem gibt es für die Animationen einen eigenen Animator (Animation.cpp), welcher ohne Fehler über die Zeit bestimmte KeyFrames animieren kann (aktuell nur lineare Animation). Wie die Animation funktioniert, kann z.B. in Door.cpp oder in ElevatorDoor.cpp eingesehen werden. Die KeyFrames werden aktuell noch hardcoded im Konstruktor definiert.

Illumination

Die ganze Szene wird durch ein Spotlight (Taschenlampe vom Spieler) beleuchtet. Außerdem gibt es noch 2 Pointlights, nämlich die vom Schaltkasten und die vom Aufzugslicht (dieses sieht man nicht besonders, weil hier Bloom/Glow implementiert wurde)

Andere spezielle Features

Das Spiel wurde mit 3D Sound ausgestattet, um die best mögliche Atmosphäre zu erzeugen.

Haben wir schon erwähnt, dass man ein Cheater sein kann? „F10“ gibt dir alle Items und Batterien, um nicht gegen die Dunkelheit zu verlieren.

Tools

Alle im Spiel befindlichen 3D Models wurden mit Blender erzeugt. Audio wurde mit Audacity editiert.