DHBW Mannheim

Team „Theuseus“

DHBW Labyrinth

Softwareentwurf

25. Oktober 2015

Projektleitung: Arwed Mett

Projektmitglieder: Dominic Steinhauser, Tobias Dorra, Leon Mutschke, Philipp Pütz

Table of Contents

1 Ziele der Architektur

1. Das Spiel soll graphisch dargestellt werden.
2. Neue Charaktere, Objekte können einfach dem Spiel hinzugefügt werden.
3. Objekte im Spiel können miteinander kollidieren.
4. Verschiedene Level werde unabhängig voneinander hinzugefügt.
5. Objekte können miteinander interagieren können. Der Spieler soll z.B. in der Lage sein Texte im Spiel lesen zu können.
6. Das Spielfeld wird am Anfang des Spiels generiert werden.
7. Das Spiel kann pausiert werden.

2 Systemarchitektur

Die Anwendung kann in drei große Komponenten zerlegt werden: Die Spiellogik, die Engine und den Levelgenerator.

Die Spiellogik steuert das Verhalten der einzelnen Elemente, die ein Teil unseres Spieles sind.

Die Engine bildet eine Abstraktionsschicht zwischen der Spiellogik und der Benutzereingabe sowie der Grafikausgabe. Durch den höheren Abstraktionsgrad kann die Spiellogik schneller und einfacher implementiert werden. Außerdem stellt die Engine in der Spiellogik mehrfach benötigte Funktionalitäten auf eine wiederverwendbaren Art und Weise zur Verfügung.

Der Levelgenerator hat die Aufgabe, zufallsbasiert die Level zu generieren, sodass diese bei jedem Spieldurchlauf anders wirken.

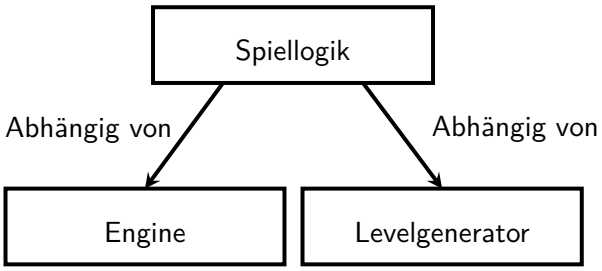


Abbildung 2.1: Abhängigkeiten zwischen den drei highlevel-Komponenten der Software.

2.1 Engine

Die Anwendung besteht aus einer Sammlung von Szenen. Jede Szene repräsentiert eine Bildschirmseite, wie zum Beispiel das Pausenmenü, das Hauptmenü, oder ein Level. Zu jedem Zeitpunkt im Spiel ist genau eine Szene aktiv.

Jede Szene beinhaltet verschiedene Spielobjekte. Ein Spielobjekt ist ein atomarer Bestandteil einer Szene wie zum Beispiel ein Knopf in den Menüs, eine Wand, ein Stück Fußboden, ein NPC oder eine Pflanze.

Die wiederverwendbaren Funktionalitäten der Spielobjekte werden in Komponenten ausgelagert. Aus diesen können neue Spielobjekte schnell zusammengesetzt werden.

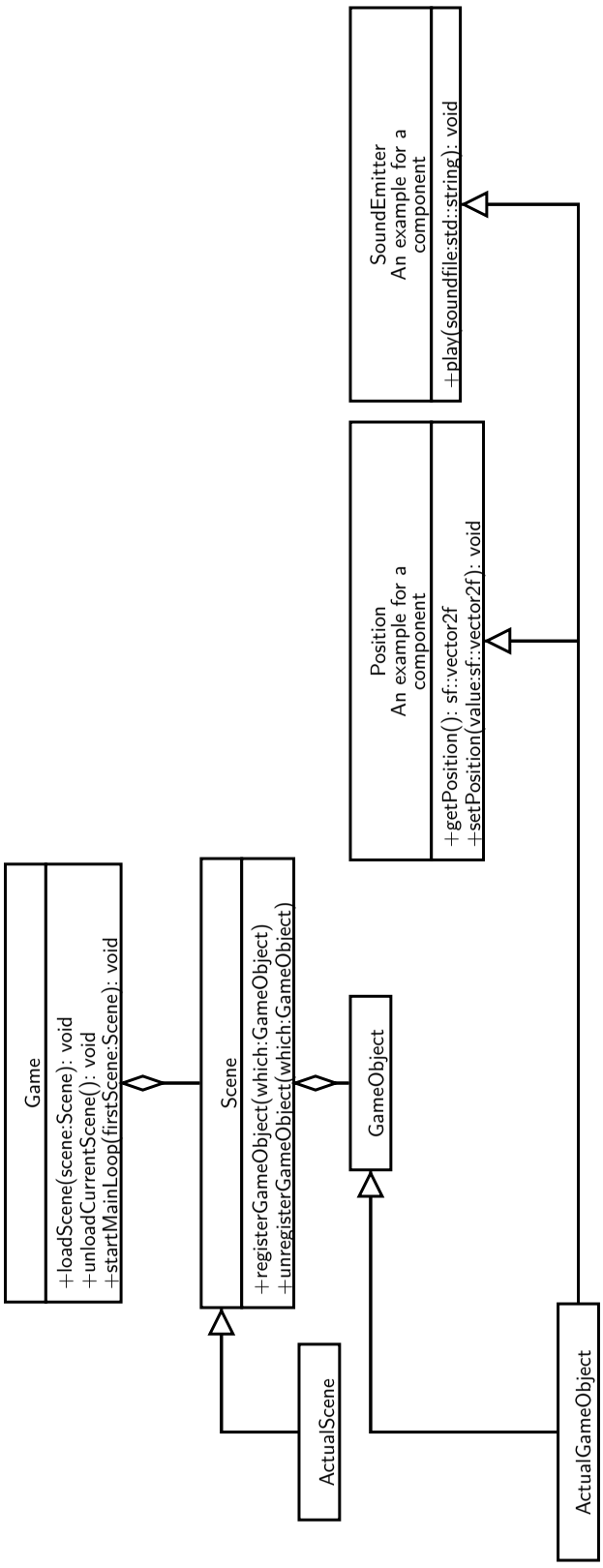


Abbildung 2.2: Zusammenfassendes Klassendiagramm der Spielarchitektur

2.1.1 Spielobjekte

Jeder Spielobjekt-Typ wird als eigene Klasse umgesetzt. Die Spielobjekte erben alle von einer gemeinsamen Vaterklasse „GameObject“.

Komponentensystem

Viele Funktionalitäten wiederholen sich bei den einzelnen Spielobjekten. Es werden jedoch nicht immer alle Funktionalitäten in jedem Spielobjekt benötigt. Dabei ist es wichtig, dass Funktionalitäten, die in einem Spielobjekt nicht benötigt wird, auch tatsächlich inaktiv oder gar nicht erst vorhanden sind. Bei einigen Funktionalitäten, wie beispielsweise der Kollisionserkennung, hat es nämlich starke negative Auswirkungen auf die Performance, wenn sie auf zu vielen Spielobjekten angewendet werden. Die unterschiedlichen Kombinationen dieser Funktionalitäten lassen sich jedoch nicht immer durch einen „klassischen Vererbungsbaum“ ausdrücken.

Die wiederverwendbaren Funktionalitäten von Spielobjekten werden in Komponenten ausgelagert. Jede Komponente wird als Klasse umgesetzt, die später ein Teil des Spielobjekts ist. Über virtuelle Mehrfachvererbung werden Abhängigkeiten zwischen den Komponenten ausgedrückt und Komponenten zu GameObject-Klassen hinzugefügt.

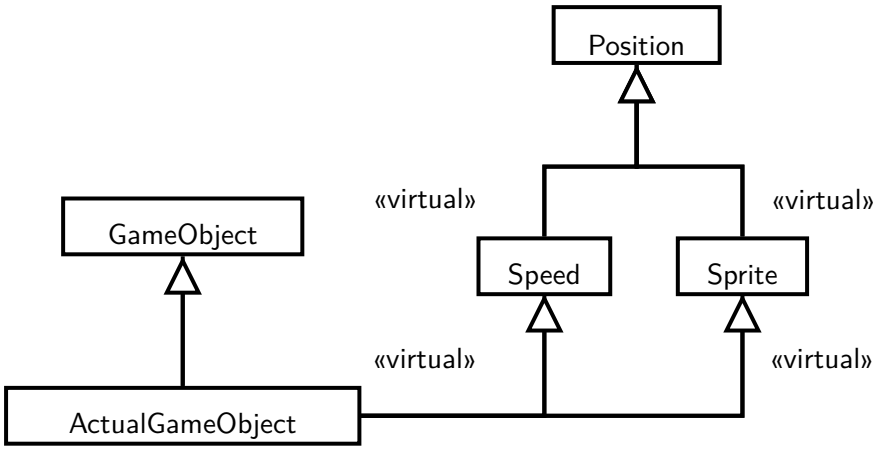


Abbildung 2.3: Komponentensystem

Durch die virtuelle Mehrfachvererbung wird das Diamond-Problem gelöst.
(https://de.wikipedia.org/wiki/Diamond-Problem)

Kommunikation zwischen Komponenten

Häufig ist es notwendig, dass Komponenten, die voneinander abhängig sind, sich untereinander über bestimmte Ereignisse informieren. So könnte beispielsweise die Kollisionserkennungs-Komponente von der Positions-Komponente benachrichtigt werden, dass sich die Position des Spielobjektes geändert hat. Da allerdings eventuell noch weitere Komponenten über die Änderung der Position benachrichtigt werden müssen, ist eine einfache Umsetzung über eine virtuelle Funktion nicht möglich. Stattdessen wird das Publish-Subscribe-Pattern zur Kommunikation zwischen den Komponenten verwendet.

Komponenten können ein „Publisher"-Objekt anbieten, um bestimmte Ereignisse mit anderen Komponenten zu „teilen". Die anderen Komponenten können sich bei dem Publisher-Objekt mit einem Funktionspointer registrieren. Wenn die anbietende Komponente das Ereignis auslöst, werden alle registrierten Funktionspointer aufgerufen und so die anderen Komponenten benachrichtigt.

Die wichtigsten Komponenten

Hier eine Übersicht über die Komponenten, die für die Umsetzung des Projektes essentiell sind:

ID	Name	Beschreibung	Abhängig von
C-10	Update	Erlaubt dem Spielobjekt, während der Update-Phase, also immer vor dem Zeichnen eines neuen Bildes, eine Aktion durchzuführen. Dies wird durch einen Publisher, der von dieser Komponente bereitgestellt wird, ermöglicht.	-
C-20	Position	Gibt dem Spielobjekt eine Position in Form einer X/Y-Koordinate.	-
C-30	Speed	Gibt dem Spielobjekt eine Geschwindigkeit in Form eines Richtungsvektors. Die Position des Spielobjektes wird entsprechend animiert.	Update, Position

C-40	Drawable	Ermöglicht es dem Spielobjekt, eine grafische Repräsentation jeglicher Art (Bitmapgrafik, Text, Rechteck, ...) zu haben.	Position
C-50	Text	Weist dem Spielobjekt einen Text, der angezeigt werden soll, zu.	Drawable
C-60	Sprite	Weist dem Spielobjekt eine (Bitmap-)Grafik zu.	Drawable
C-70	Animation	Weist dem Spielobjekt eine ständig wechselnde Grafik zu. So werden „Daumenkino-Animationen“ möglich.	Sprite, Update
C-80	Solide	Fügt dem Spielobjekt eine rechteckige Fläche hinzu, die zur Kollisionserkennung genutzt wird. Diese Fläche wird relativ zur eigenen Position angegeben.	Position
C-90	CollisionDetector	Benachrichtigt das Spielobjekt, wenn sich die eigene Kollisionsfläche aus der Solide-Komponente mit einer anderen Kollisionsfläche überschneidet.	Solide
C-100	CollisionResolver	Wenn das Spielobjekt mit einem anderen Spielobjekt kollidiert, löst diese Komponente die Kollision automatisch auf, indem sie die Position so anpasst, dass sich die beiden Objekte nicht mehr überschneiden.	CollisionDetector, Position
C-110	MouseInput	Ermöglicht dem Spielobjekt durch das Bereitstellen entsprechender Publisher-Objekte, auf Mauseingaben zu reagieren.	-

C-120	KeyboardInput	Ermöglicht dem Spielobjekt durch das Bereitstellen entsprechender Publisher-Objekte, auf Tastatureingaben zu reagieren.	-
C-130	MessageSender <MessageType>	(Klassentemplate) Erlaubt es dem Spielobjekt, Nachrichten an andere Spielobjekte zu verschicken. Eine Nachricht ist ein Objekt vom Typ MessageType. Die Spielobjekte, die die Nachricht erhalten, können durch eine maximale Entfernung beschränkt werden. Außerdem ist es möglich, die Nachricht nur an das nächste Spielobjekt zu senden, das die Nachricht empfangen kann. So ist eine Kommunikation zwischen Spielobjekten möglich, ohne dass große Abhängigkeiten zwischen diesen entstehen.	Position
C-140	MessageReceiver <MessageType>	Das Gegenstück zum MessageSender.	Position

2.1.2 Szene

Jede Szene im Spiel wird mittels einer eigenen Klasse umgesetzt, die von der abstrakten Klasse „Scene“ erbt.

Verwaltung der Spielobjekte

Die initialen Spielobjekte werden im Konstruktor von der (abgeleiteten) Szenen-Klasse erzeugt. Alle Spielobjekte werden in einem gemeinsamen Array verwaltet.

Spielobjekte, die Komponenten beinhalten, die Interaktionen mit der Szene erfordern, werden durch diese noch einmal extra bei der Szene registriert. Diese kann die Komponenten-Registrierungen dann getrennt voneinander verwalten.

Ein Beispiel dafür is die Update-Komponente. Sie muss sich bei der Szene für den Erhalt der Update-Ereignisse registrieren. So ist die Szene immer im Besitz einer vollständigen Liste aller Spielobjekte mit Update-Komponente. Nur an diese Spielobjekte muss die Scene das Update-Event weitergeben.

Die Verwaltung der Komponenten-Registrierungen erfolgt nicht generalisiert, sondern je nach Komponente individuell. So können immer die passenden Datenstrukturen verwendet werden. Bei der Update-Komponente ist es zum Beispiel am sinnvollsten, diese in einem einfachen Array zu verwalten. Bei den Komponenten zur Kollisionserkennung macht dagegen eine Verwaltung in einem QuadTree mehr Sinn.

Ebenen

Die Szene steuert auch den Zeichenprozess aller Spielobjekte. Dabei stellt sie 5 verschiedene Ebenen zur Verfügung:

- **Ebene 0**: Fußboden, Sonstiger Hintergrund
- **Ebene 1**: Schatten
- **Ebene 2**: Wände, Gegenstände, NPCs und der Spielcharakter. Die Spielobjekte dieser Ebene werden vor dem Zeichnen nach der Y-Koordinate sortiert, damit sich die Gegenstände korrekt überlappen.
- **Ebene 3**: Alles, wo der Spieler drunter hindurch gehen kann. Wolken, Baumkronen
- **Ebene 4**: Knöpfe, Punkteanzeigen, sonstige Steuerelemente, die als Overlay unabhängig vom gezeigten Kartenausschnitt immer gleich angezeigt werden.

2.1.3 Game

Es existiert eine Klasse „Game“. Dies ist die Hauptklasse, in der die gestarteten Szenen und alle Ressourcen, die global verfügbar sein müssen, verwaltet werden. Ist die Hauptschleife des Spiels in dieser Klasse implementiert.

Organisation der Szenen

Neue Szenen können über eine Methode gestartet werden, der die entsprechende Szene als Parameter übergeben wird.

Alle gestarteten Szenen werden auf einem Stack verwaltet. Die oberste Szene ist dabei die aktive Szene, alle anderen Szenen sind pausiert. Wird eine neue Szene gestartet, so wird diese auf den Stack geschoben und ersetzt so die vorherige aktive Szene. Wird die aktive Szene beendet, so wird sie wieder vom Stack entfernt. Dadurch wird die vorherige Szene wieder aktiv. So ist es einfach, (Pausen-)Menüs oder Anzeigen zu erstellen, bei denen der Benutzer nach dem Schließen wieder zurück zum Spiel kommt.

Globale Ressourcen

Die globalen Ressourcen, die in der Game-Klasse verwaltet werden sind insbesondere:

1. Das Hauptfenster
2. Die Texturen

Hauptschleife

Nachdem die grundlegenden Bestandteile der Applikation initialisiert wurden, wird die Hauptschleife des Spiels durchlaufen. Diese wiederholt immer wieder die selben Aktionen, die zur kontinuierlichen Verarbeitung der Eingaben und Produktion der grafischen Ausgabe nötig sind.

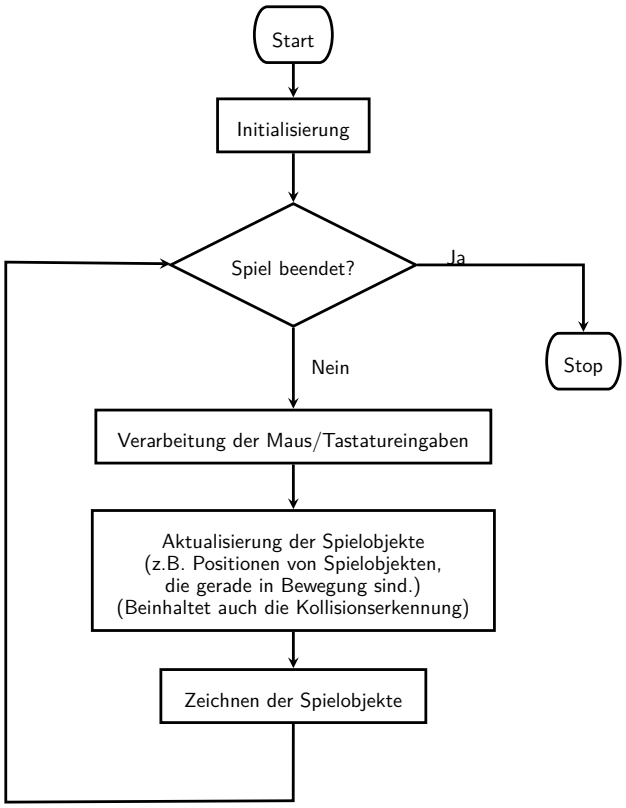


Abbildung 2.4: Ablauf der Hauptschleife

2.2 Generierung des Spielfelds

2.2.1 Objekte des Spielfelds

Das Spielfeld auf dem sich der Spieler mit seinem Charakter bewegen kann, besteht aus verschiedenen Räumen, Gängen, Gegenständen und NPCs. Alle zu platzierenden Objekte werden in der folgenden Übersicht dargestellt.

ID	**Beschreibung**
G-Item-10	**Vordefinierte Räume**
G-Item-10.1	Mensa
G-Item-10.2	Kursräume
G-Item-10.3	Parkplätze

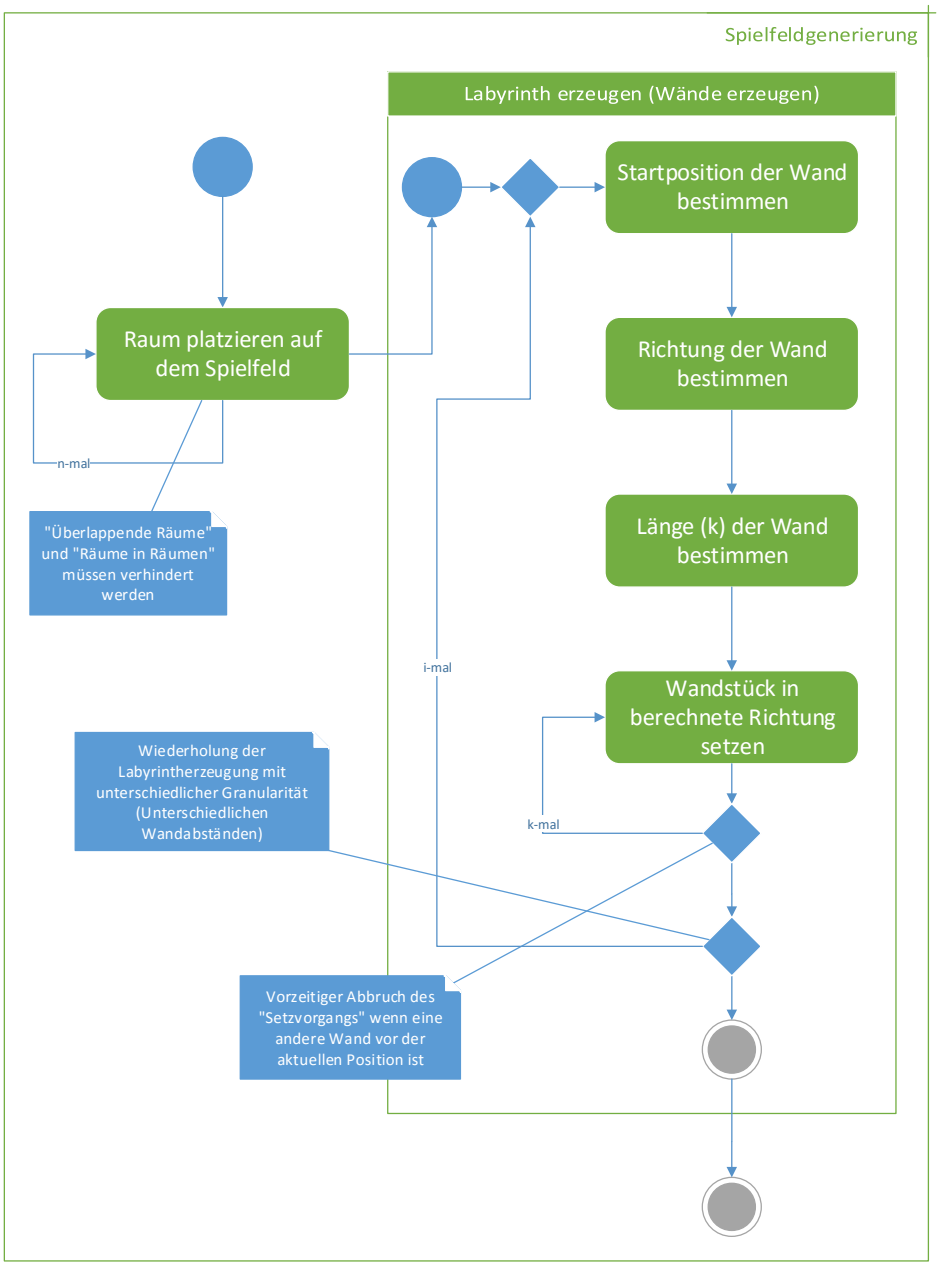
ID	Beschreibung
G-Item-20	**Labyrinthartige Gänge aus Wänden**
G-Item-30	**Gegenstände in Räumen**
G-Item-30.1	Tische
G-Item-30.2	Stühle
G-Item-30.3	Türen
G-Item-30.4	Tafeln
G-Item-40	**Gegenstände in Gängen**
G-Item-40.1	Plakate/Informationstafeln
G-Item-50	**Questitems** (siehe Level)
G-Item-60	**NPCs in Gängen und Räumen** (siehe Level)

2.2.2 Generierung des Spielfelds

Das Spielfeld auf dem sich der Spieler bewegt soll „Labyrinthartig“ aufgebaut sein und verschiedene Räume beinhalten. Dabei verbinden die Gänge die einzelnen Räume. Wichtig ist dabei, dass keine Fläche des Spielfeldes unerreichbar ist und auch alle Räume erreichbar bleiben. Weiterhin sollen keine Gänge in Räumen platziert werden und „Räume in Räumen“ entstehen.

Um das beschriebene Problem zu lösen und das Labyrinth zu erzeugen, werden zunächst alle Räume auf dem Spielfeld platziert und dabei die Fläche der Räume als „unveränderlich“ markiert. Daraufhin werden die einzelnen Wände platziert, die das Labyrinth erzeugen. Ein Problem dabei ist, dass beim platzieren der Wände keine „Blöcke“ aus Wänden entstehen oder Wände übereinander lappen. Um dies zu verhindern, werden die Wände immer mit bestimmten „Granularität“ erzeugt, sprich die Wände werden mit einem festen Abstand voneinander über dem Spielfeld platziert. Um daraufhin das Labyrinth schwieriger zu gestalten, werden weitere Wände platziert und dabei die Granularität verringert. Dabei entsteht ein immer feiner werdendes Raster aus Wänden über dem Spielfeld.

Das folgende Diagramm beschreibt den Ablauf der Spielfeldgenerierung.


Spielfeldgenerierung
Labyrinth erzeugen (Wände erzeugen)
Startposition der Wand bestimmen
Raum platzieren auf dem Spielfeld
Richtung der Wand bestimmen
n-mal
"Überlappende Räume" und "Räume in Räumen" müssen verhindert werden
Länge (k) der Wand bestimmen
i-mal
Wandstück in berechnete Richtung setzen
Wiederholung der Labyrintherzeugung mit unterschiedlicher Granularität (Unterschiedlichen Wandabständen)
k-mal
Vorzeitiger Abbruch des "Setzvorgangs" wenn eine andere Wand vor der aktuellen Position ist

3 Technologien

Zum Entwickeln des Spiels verwenden wir C++ 11. Der verwendete Code-Editor ist jedem Teammitglied freigestellt.

3.1 SFML

Für Multimediafunktionalitäten verwenden wir die Bibliothek SFML. Sie bietet uns die folgenden Möglichkeiten:

1. Hardwarebeschleunigte grafische Ausgabe
2. Audioausgabe
3. Maus- und Tastatureingaben

3.2 Buildsystem

Da auf unterschiedlichen Betriebssystemen gearbeitet wird, wird CMake als Buildwerkzeug verwendet. So kann sichergestellt werden, dass jedes Teammitglied das Projekt einfach kompilieren kann.

3.3 Static analyzer

Um Fehler vorzubeugen, soll ein statisches Codeanalysetool zum Einsatz kommen. Dieses soll versuchen, typische Fehler automatisch zu erkennen, und entsprechende Warnungen erzeugen. Welches Tool dafür verwendet und wie es in unsere Infrastruktur eingebunden wird, steht zum aktuellen Zeitpunkt noch nicht fest. Mögliche Alternativen sind:

1. Cppcheck
2. Clang Static Analyzer

3.4 Grafikeditor

Zum Erstellen von Grafiken verwenden wir GIMP.

4 Leveldesign

4.1 Story

Ein Virus ist aus einem Computer ausgebrochen und hat sämtliche Professoren infiziert. Die Professoren haben ihre offene, nette Einstellung gegenüber den DHBW-Studenten verloren...
Die Professoren haben durch den Virus nur noch ein Ziel. Nämlich die Exmatrikulation aller Studenten. Doch die Studenten sehen das nicht ein und wissen, dass das Schicksal der DHBW nun in ihren Händen liegt. Denn die Studenten sind verantwortlich, die wild gewordenen Professoren wieder zu beruhigen und zur Vernunft zu bringen. Das geschieht aber nur, indem man den Professoren ihre geliebten, leider verschwundenen, Gegenstände, wie z.B. die Musterklausur, zurückbringt.

4.2 Level 1 - „Level-Runge“

4.2.1 Ziel des Levels

Ziel ist es, innerhalb der vorgegebenen Zeit, Herrn Runge seine „Bizagi Installations-CD“ wieder zu geben.

4.2.2 Handlung

1. Herr Runge muss geheilt werden.
2. Er ist wütend, weil er seine CD von seinem geliebten Programm nicht mehr findet.
3. Aus Wut verfolgt er den Spieler, falls dieser mit ihm in Kontakt treten will, ohne dass dieser ihm seine CD zurückgibt.
4. Ist der Spieler von Herrn Runge eingeholt und kann ihm seine CD nicht geben, dann wird dieser exmatrikuliert und muss das Level von vorne beginnen.

4.2.3 Level-spezifische Objekte

ID	Beschreibung
L-Item-10	**Level-spezifische Objekte**
L-Item-10.1	Bizagi-CD
L-NPC-10	**NPC's**
L-NPC-10.1	Dozent Herr Runge

4.3 Level 2 - „Level Glaser“

4.3.1 Ziel des Levels

Herr Glaser ist ebenfalls von dem ausgebrochenen Virus befallen. Da er pro Vorlesung 3 ganze Kreidestücke benötigt, um seinen Studenten den Vorlesungsstoff zu vermitteln, wird er wütend (exmatrikuliert die Studenten), wenn er keine Kreide mehr hat.

4.3.2 Handlung

1. Herr Glaser muss geheilt werden.
2. Um seine Vorlesung komplett durchziehen zu können, benötigt er 3 Kreidestücke.
3. Kreidestücke befinden sich in unterschiedlichen Vorlesungsräumen.
4. Das Level dauert 3 min.
5. Pro Minute muss Herr Glaser ein Kreidestück gebracht werden, um das Level zu bestehen.
6. Läuft die Zeit ab, ohne das Herr Glaser ein Kreidestück besitzt, ist das Level nicht erfolgreich bestanden worden.

4.3.3 Level-spezifische Objekte

ID	Beschreibung

ID	Beschreibung
L-Item-20	**Level-spezifische Objekte**
L-Item-20.1	3 Kreidestücke
L-NPC-20	**NPC's**
L-NPC-20.1	Dozent Herr Glaser

4.4 Level 3 - „Level Hübl“

4.4.1 Ziel des Levels

Herr Hübl müssen 5 Lösungen zu seinen Aufgabenblättern gebracht werden.

4.4.2 Handlung

1. Herr Hübl muss geheilt werden.
2. Der Spieler muss bei anderen Kommilitonen Lösungen zu den von Herrn Hübl gestellten Arbeitsblättern einsammeln und diese an ihn übergeben.
3. Es müssen insgesamt 5 Dokumente beim Professor abgegeben werden.
4. Schafft dies der Spieler nicht in der vorgegebenen Zeit, so ist das Level nicht bestanden.
5. Doch der Spieler muss aufpassen. Gibt er 2 mal die gleiche Lösung eines Studenten ab, wird er exmatrikuliert, da Herr Hübl dies nicht duldet.

4.4.3 Level-spezifische Objekte

ID	Beschreibung
L-Item-30	**Level-spezifische Objekte**
L-Item-30.1	5 Lösungensblätter
L-NPC-30	**NPC's**
L-NPC-30.1	Professor Herr Hübl

4.5 Level 4 - „Level Hofmann“

4.5.1 Ziel des Levels

Herr Hofmann muss geheilt werden, indem man ihm sein Messgerät, zur Messung des Sauerstoffgehalts, zurückbringt.

4.5.2 Handlung

1. Herr Hofmann muss geheilt werden. Aufgrund des Virus, exmatrikuliert er planlos Studenten.
2. Ist ein Kommilitone exmatrikuliert(hat das Virus in sich), so darf der Spieler nicht mit diesem in Kontakt treten, weil er sonst von diesem aufgehalten wird. Dies äußert sich dadurch, dass der Spieler Zeit zum vollenden des Levels abgezogen bekommt.
3. Im DHBW-Gebäude verteilt, liegen UML-Diagramme.
4. Der Spieler weiß, dass Herr Hofmann solche Diagramme gern analysiert und deshalb kann er durch das Übergeben eines solchen Dokuments vom Exmatrikulierungsprozess abgehalten werden.
5. Wird das Messgerät innerhalb der geforderten Zeit Herrn Hofmann übergeben, so ist das Level bestanden.

4.5.3 Level-spezifische Objekte

ID	**Beschreibung**
L-Item-40	**Level-spezifische Objekte**
L-Item-40.1	Messgerät
L-Item-40.2	UML-Digramme
L-NPC-40	**NPC's**
L-NPC-40.1	Professor Herr Hofmann

4.6 Level 5 - „Level-Kruse“

4.6.1 Ziel des Levels

Ziel ist es Herrn Kruse zu heilen, indem die verschwundene C-Klausur gefunden und ihm übergeben wird, ohne dabei exmatrikuliert zu werden.

4.6.2 Handlung

1. Herr Kruse muss geheilt werden.
2. Es gibt eine bestimmte Zeitvorgabe, um ihm seine C-Klausur zu besorgen. Diese ist irgendwo in der DHBW verschwunden.
3. Da der Professor vom Virus infiziert ist und außerdem schlecht gelaunt ist, versucht er andere Kommilitonen und den Spieler zu exmatrikulieren.
4. Ist ein Student von Herrn Kruse exmatrikuliert, so hilft dieser Herrn Kruse. Zum Beispiel wird der Spieler beim zu nahen Herantreten an „exmatrikulierte“ Kommilitonen ein wenig verseucht und kann deshalb nur langsam laufen.
5. Um Herr Kruse vom exmatrikulieren abzuhalten, kann ihm eine Apfeltasche übergeben werden. Diese hilft, dass Herr Kruse 30 Sekunden keinem Studenten etwas antut.
6. Um an eine Apfeltasche zu gelangen, muss der Spieler diese in der Mensa kaufen.
7. Durch das Kaufen von Kaffee, kann der Spieler eine Geschwindigkeitsboost nutzen.
8. Sind alle Kommilitonen verseucht, versucht Herr Kruse den Spieler zu verfolgen und schließlich zu exmatrikulieren. Hierbei verfolgt er den Spieler durch die Gänge und möchte diesem auf die Schliche kommen.
9. Kann der Spieler aus einem Raum nicht mehr flüchten, so kann Herr Kruse nur durch eine Apfeltasche beruhigt werden. Falls der Spieler keine Apfeltasche mehr im Inventar hat, erfolgt der Exmatrikulierungsprozess.
10. Hat der Spieler die Klausur in einem Raum gefunden, so muss er diese aufnehmen und zu Herrn Kruse zurück bringen.
11. Schafft er dies in der vorgegebenen Zeit so hat er das Level bestanden und kommt in das nächste, da Herr Kruse wieder glücklich ist.

4.6.3 Level-spezifische Objekte

ID	Beschreibung
L-Item-50	**Level-spezifische Objekte**
L-Item-50.1	Apfeltasche
L-Item-50.2	C-Klausur
L-NPC-50	**NPC's**
L-NPC-50.1	Professor Herr Kruse

4.7 Level 6 - „Level Stroetmann“

4.7.1 Ziel des Levels

Das Ziel ist es Herrn Stroetmann zu heilen, indem seine verschwundene SetlX-Tasse gefunden und ihm übergeben wird, ohne dabei exmatrikuliert zu werden.

4.7.2 Handlung

1. Herr Stroetmann muss geheilt werden.
2. Der Spieler findet Herrn Stroetmann entweder in der Mensa oder in einem Klassenraum.
3. Herr Stroetmann wirft mit Tassen um sich, auf der Suche nach seiner Eigenen.
4. Wird der Spieler von einer Tasse getroffen, hat der Spieler verloren und er wird exmatrikuliert.
5. Man kann Herrn Stroetmann nicht ansprechen. Geht man zu nah an ihn heran, wird man auch exmatrikuliert.
6. Der Spieler weiß zu dem Punkt, wo sich Herr Stroetmann befindet.
7. Geht der Spieler aus dem Raum raus und läuft aus den Gang herunter, kann er auf einem Schwarzen Brett lesen: „Wer hat meine I ♡ SETLX Tasse gesehen? – K. Stroetmann“.
8. Nun bekommt der Spieler die Quest „Die Jagd nach der verschollenen Tasse“. Ziel der Quest ist es, Herrn Stroetmann seine geliebte Tasse zurück zu bringen.

9. Die Tasse befindet sich in dem Büro von Prof. Dr. Holger Hofmann, denn dieser ist neidisch, dass Herr Stroetmann eine so tolle Programmiersprache hat und er nicht.
10. Den Aufenthaltsort der Tasse kennt der Spieler nicht. Erfahren tut er es auch nicht.
11. Der Spieler kann den Schlüssel für das Büro in der DHBW finden. Diesen darf er behalten. In einem Schrank kann er die Tasse finden.
12. Findet der Spieler die Tasse, muss er die Tasse Herrn Stroetmann bringen.
13. Auf dem Weg zu seinem Aufenthaltsort kann der Spieler eine Flasche Möller Frutiv kaufen. Trinkt der Spieler dies und hat gleichzeitig die Tasse in seinem Besitz ist er für eine Minute unbesiegbar, d.h. er bekommt keinen Schaden, wenn er von einer Tasse getroffen wird.
14. Kauft der Spieler das Getränk nicht beziehungsweise trinkt er es nicht, kann er immer noch von einer Tasse getroffen werden und dadurch exmatrikuliert werden.
15. Egal ob der Spieler das Getränk trinkt oder nicht, steht er vor Herrn Stroetmann wird er nicht exmatrikuliert und kann auch nicht mehr von einer Tasse getroffen werden, denn Herr Stroetmann erkennt seine Tasse und wird geheilt.
16. Nun bekommt der Spieler eine Aufgabe von Herrn Stroetmann gestellt.
17. Der Spieler bekommt drei verschiedene „Hello-World“ Code Beispiele und muss die von Herrn Stroetmann geforderte Programmiersprache sagen.
18. Die möglichen Programmiersprachen und die Codebeispiele sind in C, Java und SeltX geschrieben.
19. Löst der Spieler die Aufgabe richtig, bekommt er von Herrn Stroetmann eine weitere Möller Frutiv Flasche. Ist die Antwort falsch, hat er keinen weiteren Versuch.

4.7.3 Level-spezifische Objekte

ID	**Beschreibung**
L-Item-60	**Level-spezifische Objekte**
L-Item-60.1	Tasse
L-Item-60.2	Schwarzes Brett
L-Item-60.3	SetlX-Tasse

ID	Beschreibung
L-Item-60.4	Schlüssel zu Hofmanns Büro
L-Item-60.5	Flasche Möller Frutiv
L-NPC-60	**NPC's**
L-NPC-60.1	Professor Herr Stroetmann

4.8 Allgemeine Objekte pro Level

ID	Beschreibung
L-Item-70	**Objekte jedes Levels**
L-Item-70.1	Kaffee (kann in der Mensa gekauft werden)
L-Item-70.2	Tische
L-Item-70.3	Stühle
L-Item-70.4	Tafel
L-NPC-70	**NPC's**
L-NPC-70.1	Studenten/Kommilitonen