Otto-von-Guericke-Universität Magdeburg
Fakultät für Informatik
Acagamics e.V.

Technical Design Document

Veil of Death

Autoren:
Christoph Dollase
(Team leader)

Lars Wagner

Alexander Heck

8. September 2017

Betreuer:

Gerd Schmidt
Stefan Schwarz
Acagamics e.V.

Prof. Dr.-Ing. Holger Theisel
Institut für Simulation und Graphik

Inhaltsverzeichnis

1 Einleitung **3**
1.1 Das Projekt 3

2 Programmierung **4**
2.1 Programmiersprache 4
2.2 Coding Conventions 4
2.3 Verwendete Tools 5
2.4 Frameworks und Bibliotheken 6

3 Softwarearchitektur **7**
3.1 Beschreibung des Spielablaufes 7
3.2 UML-Diagramme 7
3.2.1 Klassendiagramme 7
3.2.2 Use-Cases 9
3.3 Softwarearchitektur Entscheidungen 9

4 Technische Mindestanforderungen **10**

5 Projektmanagement **10**
5.1 Meilensteinplanung - Arbeitspakete 11
5.1.1 Meilenstein 1: “technically it’s finished” (Deadline 04.05.) 11
5.1.2 Meilenstein 2: “Welcome to the Core” (Deadline 31.05.) . 11
5.1.3 Meilenstein 3: “Gimme some Feedback” (Deadline 30.06..) 11
5.1.4 Meilenstein 4: “Level 1 Completeted” (Deadline 31.07.) . 11
5.1.5 Meilenstein 5: “This won’t work” (Deadline 31.08.) . . . 12
5.1.6 Meilenstein 6: “It lives!” (Deadline 19.09.) 12

1 Einleitung

Das Technical Design Document stellt die technischen und strukturellen Anforderung an unser Spiel dar. Zuerst gehen wir auf unsere Coding Conventions ein. Danach geben wir eine Auflistung unserer verwendeten Tools, die für die Entwicklung des Spiels benötigt werden. Des Weiteren gehen wir auf die Struktur des Spiels ein durch Klassen- und Use Case Diagramme. Zum Schluss gibt dieses Dokument Auskunft über die Projektplanung mit ihren Arbeitspaketen und Meilensteinen.

1.1 Das Projekt

Veil of Death ist ein Level basierter Runner in dem der Spieler versucht mit dem Protagonisten aus einer Festung zu entkommen und entsprechenden Hindernissen auszuweichen ohne in den ihn verfolgenden Nebel zu geraten. Rennen tut der Protagonist selbstständig. Der Spieler muss den Hindernissen durch Springen, Rutschen und dadurch, dass er die Lane wechselt ausweichen. Man bekommt einen Score und eine Wertung am Ende jeden Levels, welche einem sagt, wie gut man abgeschnitten hat. Sollte der Spieler sterben muss er das jeweilige Level von vorne beginnen.

2 Programmierung

2.1 Programmiersprache

Die Programmiersprache für die wir uns entschieden haben ist C#. Einerseits erleichtert die Ähnlichkeit zu Java, die Arbeit im Team, da jeder im Team bereits Erfahrungen in der Programmierung mit Java gesammelt hat. Weiterhin wird C# uns als Programmiersprache von dem Monogame Framework vorgeschrieben.

2.2 Coding Conventions

Format-Vorgaben
• je Zeile nur 1 Anweisung (“{“ und “}“ zählen jeweils als eine Anweisung; “else“ ist eine anweisung, danach immer enter)
• wenn eine Klammerung {} länger als 10 Zeilen ist, wird die schließende Klammer kommentiert (z.B: “} // end if(true)“ , “} //end class(player)); das erleichtert die Lesbarkeit um einiges.
• zugehörige Codefragmente sind durch Einrücken deutlich zu machen (z.B.: nach “if(bedingung)“ oder “else“ ist in der nächsten Zeile ein zu rücken, gilt rekursiv -¿ Mehrfachverschachtelung)

Klassen-Variablen
1-2 stelliges kleingeschriebenes Präfix zur Typbezeichnung + “_“ vor dem Variablennamen in CamelCase groß beginnend.
• t_VariablenName (Time)
• sp_VariablenName (Sprite)
• tx_VariablenName (Textur)
• m_VariablenName (Model)
• O_VariablenName (GameObject)
• x_VariablenName (Matrix)
• peVariablenName (PanelElement -Sprite)
• ptVariablenName (PanelElement -Text)

Standard-Variablen
Einstelliges klein geschriebenes Präfix zur Typbezeichnung + Variablennamen in CamelCase groß beginnend.
• iVariablenName (integer)
• fVariablenName (float)
• sVariablenName (String)

Jede Variabel vom Typ Bool ist immer mit Frage-Präfix zu Bennenen Bsp.: bool isCollactable, hasLoot, isActive, canRespawn;
Kommentieren
• Kommentare, die Funktionen und Methoden dokumentieren sind mit “summary“ einzupflegen
• übergebene Parameter sind mit “param“ zu erklären
• Summary wird in in Visual Studio mit 3 /// eingeleitet (Macro)
• Beispiel:
/// <summary>
/// kurze sinnvolle Beschreibung.
/// </summary>
/// <param name=“_uebergabeParameter“> Was soll übergeben werden </param>
private void doStuff(var _uebergabeParameter) {...}

2.3 Verwendete Tools

• Blender
Für die Modellierung des Charakters und der Gegner wird Blender verwendet. Die fertigen Modelle werden als .fbx exportiert um dann mit Monogame importiert zu werden. Alle Modelle im .stl Format werden importiert und zu .fbx konvertiert.
• Solid Edge
Für die Modellierung von Hindernissen und dem Rest der Spielwelt sowie Collectables wird Solid Edge verwendet. Die Dateien werden als .stl exportiert um dann mit Blender konvertiert zu werden.
• Audacity
Mit Audacity werden die entsprechenden Soundeffekte und auch die Hintergrundmusik kreiert und auch geschnitten sowie bearbeitet. Exportiert werden die Dateien in einem Format, welches von Audacity angenommen wird.
• PhotoShop CS5 (Sprites, Textures, GUI, Menü)
Photoshop benutzen wir zum Erstellen der Sprites sowie der Texturen und des Menüs. Die Benutzeroberfläche erstellen wir ebenfalls mit PhotoShop CS5.
• Git/ Turtoise Git
Als Versionierungssoftwares werden die Freewares Git und Turtoise Git benutzt, die das Projekt auf einen privaten Server laden. • Visual Studio 2015/ 2017
Visual Studio benutzen wir als Entwicklungsumgebung für die Software, da man mit Der Struktur der IDE aus früheren Veranstaltungen vertraut ist. Außerdem lässt sich Monogame als zusätzliches Plug-In installieren.
• Github (Projektmanagement)
Wir benutzen die Funktion von GitHub um unsere Arbeitspakete zu planen und uns Über den Fortschritt des Projektes zu versichern. Weiterhin können auch direkt kleine Tickets an die betreffende Person verschickt werden.

2.4 Frameworks und Bibliotheken

Wir benutzen zur Entwicklung des Spiels das Framework Monogame in der Version 3.6. Dieses Framework unterstützt die Programmiersprache C#, die an Java angelehnt ist und somit einen Teil der Gruppe einen einfachen Einstieg bereitet. Weiterhin ist das einbinden von Models, Fonts und weiteren grafischen Texturen mit dem vom Hause aus mitgebrachten Contentmanager ein Leichtes. Es hat weiterhin eine integriert Gameloop.

Animations-Bib von Mirko mit den dazu gehörigen Bibliotheken

3 Softwarearchitektur

3.1 Beschreibung des Spielablaufes

Der Spieler gerät nach Starten des Spiels ins Menü wo er entweder das Spiel starten kann oder es verlassen kann. Er kommt nach auswählen von „Spiel Starten“ direkt an den Anfang des Ersten Levels. Der Protagonist beginnt zu laufen und der Spieler erhält Kontrolle über seine Aktionen. Er muss das Level durch Vermeiden von Hindernisse, Debuffs und Gegnern abschließen. Als Unterstützung bekommt der Spieler durch einsammeln von Schild oder Schwert, welche ihm gegen die Hindernisse und Gegner helfen. Er sammelt auf dem Weg Coins ein. Das nächste Level kann nur betreten werden, wenn eine Mindestmenge an Coins eingesammelt wurde. Am Ende jedes Levels gibt es eine Wertung, welche einem sagt, wie gut man das Level abgeschlossen hat.

3.2 UML-Diagramme

3.2.1 Klassendiagramme

Die Game-Klasse ist die Hauptklasse unseres Spieles. Von Ihr aus geht der allgemeine Game-Loop, der in allen Zuständen des Spiels vorkommt. Die unterschiedlichen Zustände des Spiels werden mit Hilfe eines Interfaces (IGameStates) mit gemeinsamen Funktionen (z.B. Initialisierung, Game-Loop, Draw) gespeist. Für alle Objekte im Spiel die miteinander agieren können, haben wir eine übergeordnete abstrakte Klasse (AGameObject) erschaffen, von der die einzelnen Objektklassen erben. Hierzu gibt es noch 2 verschiedene Interfaces, die die Objekte in starre (IStaticEntity) und bewegliche (IMovingEntity) einteilen. Die Collision-Klasse berechnet alle Kollisionen des Spielers mit der Umgebung. Sie erhält Input aus der Level- und Player-Klasse. Die Player-Klasse, welche ebenfalls ein GameObject ist, wird durch die PlayerController-Klasse gesteuert. Hier wird vor Allem die Bewegung des Spielers und die Kommunikation mit dem Rest der Umgebung (Position im Grid, Animation) übernommen. Unsere Map-Klasse lädt die Spielwelt aus einer Bitmap-Grafik, welche die verschiedenen Blöcke kodiert. Die Block-Klasse holt sich die Informationen, welches Model, oder Textur geladen werden soll aus einem Dictionary, die Klasse LevelContent. Wir haben zwei zentrale Klassen für wichtige Steuerungs- und Speicherfunktionen. GameConstants ist eine statische Klasse, die sämtliche wichtigen Einstellungen und Standartwerte für das Spiel beinhaltet. Des weiteren haben wir einen Singelton-Klasse implementiert, die zwischen den verschiedenen Leveln und GameStates die wichtigsten Informationen überträgt und managed. Ein Gesamtüberblick über alle unserer Klassen sind der Abbildung 1 zu entnehmen.

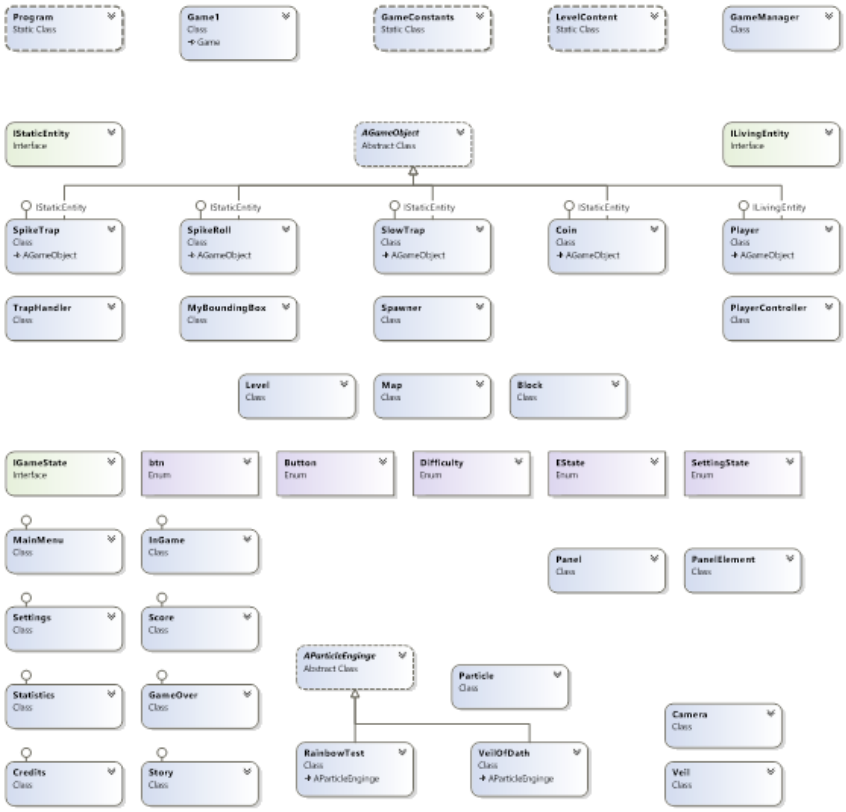

Abbildung 1: Klassendiagramm Veil of Death

3.2.2 Use-Cases

Hauptmenü
Das Hauptmenü zeigt eine Auswahl von fünf Buttons, die je nach Auswahl entsprechend hervorgehoben werden. Die Buttons sind:
• Spiel starten
Durch Spiel starten kommt man in den InGame Zustand des Spiels, in dem die einzelnen Level geladen und gestartet werden.
• Settings
Hier können vom Spieler verschiedene Einstellungen vorgenommen werden. z.B. Schwierigkeitsgrad, Lautstärke
• Statistics
In den Statistiken werden alle aktuellen Fortschritte angezeigt
• Credits
In den Credits gibt es eine kurze Nennung der am Spiel beteiligten Teammitglieder, mit ihren jeweiligen Betätigungsfeldern
• Exit
Exit beendet das Programm.

Ingame (Player vs. Environment)
Ingame kann der Spieler den Protagonisten verschiedene Aktionen ausführen lassen um Hindernissen aus dem Weg zu gehen:
• Lane wechseln
• Springen
• Dodgen
Er kann durch drüberlaufen verschiedene Collectables wie Coins oder Schwert aufnehmen. Er wechselt Lanes um Fallen, Hindernissen und Mobs aus dem Weg zu gehen, denn eine Kollision bedeutet im besten Fall einen Debuff aber meist den sofortigen Tod.

3.3 Softwarearchitektur Entscheidungen

Wir haben die Architektur der Software so gewählt, das wir die Struktur adaptiv halten. So halten wir uns die Möglichkeit offen, relativ einfach zusätzliche Komponenten in das Spiel zu integrieren. Durch die Separierung des Codes in seine logischen Bereiche erhoffen wir uns, dass durch die Kapselung die Wartbarkeit und Lesbarkeit sich stark verbessert und auch Erweiterungen oder Änderungen an ffremdenCodestellen kein Problem darstellen.

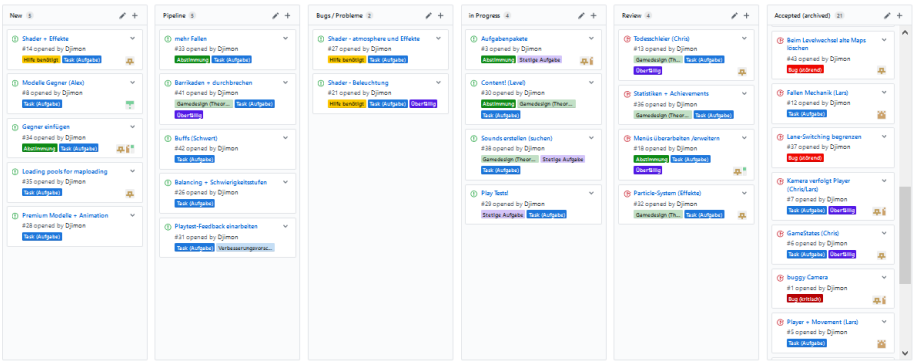

Abbildung 2: Kanban (Anfang September 2017)

4 Technische Mindestanforderungen

Als Betriebssystem wird mindestens Windows 7 vorausgesetzt. Die Grafikkarte sollte mindestens DirectX10 und eine Auflösung von mindestens 1200*700 unterstützen. Es sollte sowohl eine Soundkarte als auch eine Möglichkeit zur Soundausgabe in dem Gerät enthalten sein, da dies das Spielerlebnis deutlich verbessert. Es ist möglich das Spiel mit einem Controller oder per Tastatur zuspielen.

5 Projektmanagement

Github unterstützt inzwischen das Projektmanagement mit einer weiten Palette an Tools und UIs, sodass wir uns entschieden haben GitHub neben der Versionsverwaltung auch als zentrales Werkzeug für die Projektverwaltung zu benutzen. Neben der Einteilung der verschiedenen Arbeitspakete in Meilensteine haben wir uns auch dazu entschlossen mit dem Kanban zu arbeiten. So hat jedes Teammitglied immer die aktuellen Status des Projektes vor Augen. Um den Informationsfluss des Kanbans noch zu verstärken haben wir Gebrauch von sogenannten Labels gemacht, die mit farblichen Markierungen den einzelnen Aufgabenpakete zusätzliche Attribute verleihen. So war es uns möglich auf den ersten Blick zu sehen, welcher Task gerade bearbeitet wird und wo es Probleme oder Rückfragen gibt. Auch Bugs und andere Fehler haben wir als Tasks erfasst und mit auffälligen Roten Labels markiert. Dies ermöglichte eine sehr nutzerfreundliche Arbeit an einer einzigen LLeinwandËin Momentaufnahme des Kanbans in der späteren Projektphase ist der Abbildung 2 zu entnehmen.

5.1 Meilensteinplanung - Arbeitspakete

5.1.1 Meilenstein 1: "technically it's finished" (Deadline 04.05.)

Game-Summary (Einleitung): Alex
Coding Conventions: Chris
Bibliotheken und Frameworks: Lars
Verwendete Tools: Chris
UML- und Klassendiagramme: Lars
Projektmanagement, Aufgabenpakete: Chris
Technische Mindestanforderung: Lars
(Als Aufgaben nehmen wir jeweils die Überschriften und wer sie bearbeitet hat)

5.1.2 Meilenstein 2: "Welcome to the Core" (Deadline 31.05.)

Gamestates: Chris
Player: Lars
Movement: Lars MapGeneration (Bitmap): Chris
StartMenü: Alex
Modell Player: Alex
Modelle Wände, Boden: Alex
Texturen: Wände und Boden: Chris, Alex
Model Collactables: Alex

5.1.3 Meilenstein 3: "Gimme some Feedback" (Deadline 30.06..)

GUI + Scoresystem: Lars, Chris
SoundSystem: Lars
VisualEffects: Chris, Alex
Prototyp Shader: Chris
Fallen Mechaniken: Lars
Mechanik des Todes Schleier: Chris
Model Fallen: Alex
Model Gegner: Alex
Refactoring: Chris

5.1.4 Meilenstein 4: "Level 1 Completeted" (Deadline 31.07.)

Ein Komplettes Level mit allen mechaniken Spielbar: Lars
Prototyp Proceduale mapgeneration: Chris
Animationen Laufen, Springen, Rutschen: Alex
Weitere Shader für Effekte und Postprocessing: Chris

Balancing Belohnung und Score: Lars
Levelübersicht: Chris

5.1.5 Meilenstein 5: "This won't work" (Deadline 31.08.)

Level 2,3 (4,5): Lars
Procedural Mapgeneration: Chris
Playtests, Feedback: Alle
Balancing Belohnung, Bestrafung: Lars
Polishing Models und Animationen: Alex
Polishing Menüs: Alex
Achievements: Alle
Refactoring: Lars, Chris

5.1.6 Meilenstein 6: "It lives!" (Deadline 19.09.)

Debugging: Alle
Polishing: Alle
...