Machbarkeitsanalyse

Projekttitel: **Laserwürfel**
Auftraggeber: **Florian Weiss**
Auftragnehmer: **Team Laserwürfel**
Schuljahr: **2015/16** Klasse: **5CI**

VERSION	DATUM	AUTORIN/AUTOR	ÄNDERUNG
v0.1	21.09.2015	Clemens Stadlbauer	Dokument erstellt
v0.2	21.09.2015	Jakob Bal	Ist-Situation und Anforderungen erstellt
	21.09.2015	Robin Kaggl	Alternativen C++ erstellt
	21.09.2015	Erik Ritschl	Alternativen Python und Haskell erstellt
	21.09.2015	Clemens Stadlbauer	Alternativen Unity, Qt und OpenGL erstellt
v1.0	27.09.2015	Clemens Stadlbauer	Arbeiten zusammengeführt
v1.1	28.09.2015	Jakob Bal	Ist-Situation und Qt-Analyse upgedatet
v2.0	04.10.2015	Jakob Bal Erik Ritschl Clemens Stadlbauer	Feedback eingearbeitet und Dokument überarbeitet
v2.1	11.10.2015	Clemens Stadlbauer	Feedback eingearbeitet
v3.0	12.10.2015	Jakob Bal Erik Ritschl Clemens Stadlbauer	Feedback eingearbeitet und Levelgeneration eingetragen

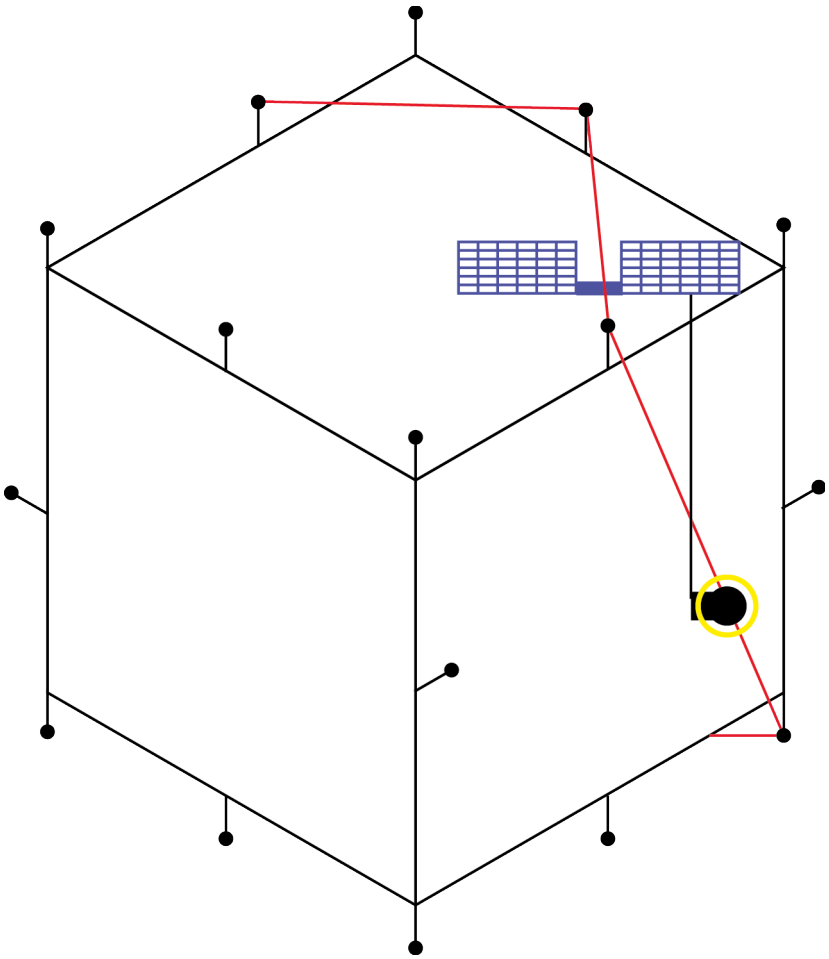
1 Ist Situation

Das Team besteht aus vier Teammitgliedern. Außerdem hat das Team zwei Betreuer an die es sich wenden kann, wobei einer dieser zwei der Hauptbetreuer ist.

Die Idee des Projekts ist es, ein 3D-Puzzlespiel zu erstellen. Die Spielmechanik des Spiels basiert darauf, dass es einen 3D-Würfel gibt, der an seinen Kanten und Ecken Verbindungspunkte für Laser hat. Ziel des Spieles ist es, den Laser mit allen Punkten zu verbinden ohne, dass sich die Laserstrahlen überkreuzen. Dabei muss der Laser an verschiedenen Hindernissen (Mauern usw.) umgeleitet werden.

Das Spiel wird auf dem PC spielbar sein. Die Steuerung erfolgt mittels Maus und/oder Tastatur. Es wird einen selbst komponierten Soundtrack geben, der das Spiel atmosphärisch unterstützt.

Skizze des Spielprinzips:



1.1 Analyse der IST-Situation

Es gibt viele Puzzlespiele auf dem Markt, die darauf aufbauen, dass man gewisse Dinge miteinander verbinden muss, die sich nicht überkreuzen dürfen (z.B.: Legen von Schienen, die ein Feld ausfüllen müssen). Allerdings ist keines von denen 3D.

Projektmanagement:

Know How bezüglich Scrum und Projektmanagement ist vorhanden, aber noch nicht ausgereift. Daher werden die Planungsphasen länger als üblich brauchen.

Technik:

Für statische Websites ist Wissen im Team vorhanden.

Theoretisches Wissen über Programmieren ist ebenfalls vorhanden, jedoch haben wir wenig Wissen bezüglich dem Erstellen von Puzzlespielen.

Budget:

Unser Budget ist sehr beschränkt, deswegen werden wir nur Geld für einen Webspace und eine Domain ausgeben.

2 Anforderungen

Es gibt eine einzige Benutzergruppe, nämlich die Spieler des finalen Produkts. Das Zielalter unserer User wird von uns, zum Zweck der besseren Planung, zwischen 16 und 25 Jahren festgelegt. Das gesamte Spiel wird in Englisch umgesetzt. Das fertige Produkt wird auf der Projektwebsite zum Download zur Verfügung stehen.

Das Spiel wird auf dem Computer gespielt und als native Applikation umgesetzt. Die Steuerung erfolgt mittels Tastatur und/oder Maus. Das Spiel soll auf einem modernen System mit mindestens 30 FPS laufen und eine intuitive Steuerung haben.

Die Level des Spiels werden auf zwei Arten bereitgestellt:

- Selbstgemacht
- Generiert

Die selbstgemachten Level werden durch eine fortlaufende Story unterstützt und bieten neuen Spielern einen einfachen Einstieg in das Spiel, wo sie die Mechaniken kennenlernen können. Für wiederkehrende Spieler gibt es die generierten Level, die jedes mal eine neue Herausforderung anbieten.

3 Lösungsalternativen

3.1 Grafik Framework

3.1.1 Unity 3D

Beschreibung

Unity ist ein professionelles Spielentwicklungstool.

Vorteile

Alle Tools und Funktionen die wir je brauchen könnten sind in Unity bereits vorhanden. Diese sind zum Beispiel 3D Modellierung, Animation, Interaktion und Audio.

Nachteile

Die Unity 3D Team Lizenz ist kostenpflichtig. Außerdem kann man das Projekt nur mit Unity bearbeiten.

Risiken

Da Unity ein proprietäres Dateiformat verwendet, können wir im Laufe des Projekts das Grafik Framework nicht mehr wechseln.

Lösungsbeschreibung

Da Unity für ein Team kostenpflichtig ist, fällt es als Alternative aus.

3.1.2 Qt

Beschreibung

Qt ist ein monolithisches Framework, welches für unsere Zwecke Funktionen wie 3D Rendering anbietet.

Vorteile

Deployment ist auf allen etablierten Plattformen möglich. Keine Kosten für Open Source Projekte.

Nachteile

Dokumentation ist aufgrund der Größe sehr unübersichtlich.

Risiken

Deployment mit Qt erfordert viele Schritte die den ganzen Vorgang unübersichtlich und fehlerhaft machen können.

Lösungsbeschreibung

Das Team muss sich mit der API und dem Workflow von Qt vertraut machen. Dann wird das Produkt mit dem Qt Framework entwickelt.

3.1.3 OpenGL

Beschreibung

Das am weitesten verbreitete Grafik Framework.

Vorteile

Beste Abdeckung, da es schon länger Industriestandard ist. Außerdem ist es sehr stabil und gut dokumentiert.

Nachteile

Keine Erfahrung.

Risiken

API ist zu komplex oder Funktionsumfang nicht ausreichend.

Lösungsbeschreibung

Das Team lernt OpenGL und verwendet es.

3.2 Programmiersprache

3.2.1 C++

Beschreibung

C++ ist eine sehr weit verbreitete, kompilierte, objektorientierte Programmiersprache.

Vorteile

Große Anzahl an Libraries ist vorhanden. Kann für jede gängige Prozessorarchitektur kompiliert werden.

Nachteile

Komplexe Syntax macht das Lernen schwieriger. Durch das direkte Memory Management können Fehler leichter entstehen.

Risiken

Wir schaffen es nicht die Sprache zu lernen.

Lösungsbeschreibung

Das Team lernt C++ und das Produkt wird darin umgesetzt.

3.2.2 Haskell

Beschreibung

Haskell ist eine beliebte funktionelle Programmiersprache, die außerhalb vom akademischen Kontext allerdings nicht sehr viel Verwendung findet.

Vorteile

- Der Datentyp von jeder Variable ist zu Compile-Time festgelegt, was eine ganze Klasse von Bugs von vornherein unmöglich macht.

- Schöner, robuster Code. Funktionen nehmen gewisse Parameter und liefern bestimmten Output, und haben keine weiteren, potentiell unerwarteten, Nebeneffekte. Es kann daher mathematisch bewiesen werden, dass ein Programm korrekt ist.
- Der Code wird kompiliert und nicht interpretiert, was bedeutet, dass der Endnutzer keine zusätzliche Software installieren muss. Außerdem läuft die Anwendung schneller.

Nachteile

- Beim funktionellen Programmieren verwendet man eine komplett andere Denkweise, um an Probleme heranzugehen. Der Lernaufwand ist dementsprechend groß.
- Der Code muss für jede Plattform (Windows, Unix, 32/64 Bit) individuell kompiliert werden.
- Haskell (und funktionelle Programmiersprachen im Allgemeinen) ist leider nicht gut geeignet für grafische Oberflächen, im speziellen Spiele. Es gibt Ansätze, mit denen man schön Spiele programmieren kann mit Haskell (siehe „Functional Reactive Programming“), aber diese Verwenden noch kompliziertere Konzepte, für die der Lernaufwand noch höher wäre.

Risiken

Der Lernaufwand ist zu groß, oder das Spiel ist in Haskell nur schwer umsetzbar.

Lösungsbeschreibung

Wir installieren die Haskell Plattform (verfügbar für jedes gängiges OS, auf https://www.haskell.org/downloads) auf unseren Arbeits-PCs. Dann lernen wir Haskell und funktionelles Programmieren. Wir setzen die Anwendung in Haskell um, und kompilieren eine Version für jede Plattform (Windows, Unix, 32/64 Bit).

3.2.3 Python

Beschreibung

Python ist eine sehr beliebte interpretierte und objektorientierte Programmiersprache.

Vorteile

- Es ist eine sehr leicht zu erlernende Sprache.
- Unsere Teammitglieder haben Erfahrung mit objektorientiertem Programmieren.
- Python hat sehr viele Bibliotheken, die zur freien Verfügung stehen.

Nachteile

- Der Endnutzer muss Python auf seinem Computer installiert haben, um das Spiel ausführen zu können.
- Whitespace als Element der Syntax ist für uns noch ein recht neues Konzept, und könnte zu Verwirrung führen.
- Abwesenheit von statischen Datentypen kann zu mehr Fehlern führen.

Risiken

- Der Endnutzer schafft es nicht, die richtige Version von Python zu installieren.

Lösungsbeschreibung

Das Team installieren Python auf unseren Computern und lernen die Grundlagen der Sprache. Das Team setzen die Anwendung in Python um, und lernen währenddessen die Details der Sprache, indem wir auf Probleme stoßen und diese durch eigenständige Recherche lösen.

3.3 Level für Endlosmodus

3.3.1 Handgemacht

Beschreibung

Eine große Anzahl von Leveln wird von dem Team manuell erstellt.

Vorteile

Die Level sind in der Schwierigkeit konsistenter.

Nachteile

Benötigt sehr viel Zeit.

Risiken

Die Anzahl der Level, die wir in der Projektzeit erstellen können, reicht nicht für den Endlosmodus.

Lösungsbeschreibung

Das Team erstellt eine möglichst große Anzahl an Level, die dann dem User in zufälliger eihenfolge präsentiert werden.

3.3.2 Generiert

Beschreibung

Ein Algorithmus erstellt dem User bei Bedarf einen Level.

Vorteile

Unbegrenzt viele Level sind möglich.

Nachteile

Konsistenz der Levekqualität kann nicht garantiert werden.

Risiken

Der Algorithmus wird unvollständig oder fehlerhaft, und generiert unlösbare oder langweilige Level.

Lösungsbeschreibung

Das Team wird einen Algorithmus erstellen, der zuerst einen zufälligen Pfad auf einen leeren Würfel legt, und dann Hindernisse hinzufügt.

4 Analyse der Lösungsalternativen

4.1 Grafik Framework

4.1.1 Unity 3D

Kosten

Zeitkosten für Lernaufwand und Lizenskosten.

Nutzen

Vollständiges 3D-Game Framework.

Zeitrahmen

Unser Team hat noch nie mit Unity 3D gearbeitet und müsste deshalb erst einmal die Funktionen, danach die Anwendung und abschließend den Workflow lernen. Dies ist nicht in unserer verfügbaren Projektzeit duchführbar.

Ressourcen

Online-Tutorials.

Risiken

Wir schaffen es nicht, uns dem fremden Workflow von Unity anzupassen.

Technische Machbarkeit

Unity 3D bietet uns alle benötigten Funktionen (siehe Beschreibung).

4.1.2 Qt

Kosten

Da unser Produkt Open Source ist, haben wir nur die Zeitkosten für Lernaufwand.

Nutzen

Deployment ist auf allen etablierten Plattformen möglich, wodurch das fertige Produkt auf allen Plattformen verfügbar ist. Durch die Stabilität und gute Dokumentation wird die Entwicklung und das Lernen erheblich erleichtert.

Zeitrahmen

Es wird ungefähr 4 Wochen brauchen bis das Team mit QT vertraut ist und die benötigten Funktionen für das Projekt umsetzten kann. Außerdem kommt noch der Lernaufwand für C++ dazu, damit QT verstanden werden und das Projekt mit seinen Funktionen umgesetzt werden kann.

Ressourcen

Die Website des Frameworks stellt alle Dinge (Tutorials, Dokumentation,...) zur Verfügung, damit wir es lernen können. Außerdem ist der Download auch auf der Website bereitgestellt.

Risiken

Die API ist zu komplex und wir schaffen es nicht das Produkt rechtzeitig fertigzustellen.

Technische Machbarkeit

Für alle etablierten Betriebssysteme sind Compiler für C++ vorhanden. Mithilfe von OpenGL können mit Qt 3D-Grafiken umgesetzt werden. Für das Interface bietet es einen GUI Editor Out of the box an.

4.1.3 OpenGL

Kosten

Zeitkosten für Lernaufwand

Nutzen

Es funktioniert auf jeder etablierten Plattform. Der Lernvorgang wird durch die gute Dokumentation beschleunigt.

Zeitrahmen

Zwei Wochen Lernzeit um die Grundlagen kennenzulernen, danach geschätzt bis zu zwei Stunden pro Feature (z.B. Objekt erstellen und zeichnen). Die Anzahl der Features wird wahrscheinlich im Bereich von zehn liegen.

Ressourcen

Online-Tutorials wie z.B. http://learnopengl.com/.

Risiken

Lernaufwand zu groß.

Technische Machbarkeit

OpenGL ist auf allen ausgewählten Zielplattformen verfügbar.

4.2 Programmiersprache

4.2.1 C++

Kosten

Zeitkosten für Lernaufwand

Nutzen

Es wird auf jeder etablierten Plattform laufen, und weil es kompiliert ist, gute Performanz bieten.

Zeitrahmen

Alleine die Syntax würde mindestens zwei Wochen zum lernen benötigen. Mit C++ effizient Arbeiten zu können, würde die gesamte verfügbare Projektzeit in Anspruch nehmen.

Ressourcen

Online-Tutorials.

Risiken

Lernaufwand zu groß.

Technische Machbarkeit

C++ Compiler sind für alle ausgewählten Zielplattformen verfügbar.

4.2.2 Haskell

Kosten

Die Sprache muss erlernt werden.

Nutzen

Es sind weniger Bugs möglich. Wir bekommen Erfahrung nicht nur in einer neuen Programmiersprache, sondern einer ganz neuen Art zu Programmieren und an Probleme heranzugehen.

Zeitrahmen

Um die Sprache in ausreichendem Ausmaß zu lernen, würden wir die gesamte verfügbare Projektzeit benötigen.

Ressourcen

Es gibt eine Vielzahl von Tutorials und Kursen im Internet zu Haskell und funktionellem Programmieren.

Risiken

Der Lernaufwand ist einfach zu groß, und im Rahmen dieses Projekts nicht umsetzbar. Daher scheidet diese Sprache aus. Die folgende Analyse ist deshalb nicht all zu detaillreich.

Technische Machbarkeit

Die notwendigen Tools zum Entwickeln sind auf allen gängigen Betriebssystemen verfügbar. Die kompilierte Anwendung kann ebenfalls auf allen PCs unserer Zielgruppe ausgeführt werden. Die Verwendung von z.B. OpenGL ist durchaus möglich, aber noch recht unüblich in der Haskell Welt, und wir würden daher auf mehrere Probleme in der Umsetzung stoßen, die wir selber lösen müssten.

4.2.3 Python

Kosten

Die Sprache muss erlernt werden.

Nutzen

Durch die Leichtigkeit der Syntax können wir die Sprache schnell erlernen und sparen dadurch Zeit.

Zeitrahmen

Die Grundlagen können in etwa einer Woche angeeignet werden. Wir werden die Sprache laufend lernen während wir das Projekt umsetzen. Dafür werden wir etwa 20% der Arbeitszeit aufwenden müssen.

Ressourcen

Es stehen zahlreiche Python Tutorials und Kurse im Internet zur freien Verfügung (z.B. Learn Python the Hard Way, Python Wiki, Python Programming (Video)). Python selbst kann für jedes Betriebssystem einfach installiert werden.

Unsere Entwickler haben alle bereits eine Lieblings-Entwicklungsumgebung, die für Python verwendet werden kann, denn für Python ist keine spezielle IDE notwendig.

Risiken

Lernaufwand zu groß.

Technische Machbarkeit

Python läuft auf allen gängigen Betriebssystemen (und mehr) und kann auf https://www.python.org/downloads/ heruntergeladen werden. Falls wir uns dazu entscheiden, dass wir es unseren Usern nicht zutrauen wollen, Python selber zu installieren, gibt es Möglichkeiten, die Anwendung mit Python mitzuliefern.

4.3 Level für Endlosmodus

4.3.1 Handgemacht

Kosten

Zeitkosten für die Level.

Nutzen

Der User wird mit einheitlicheren Leveln präsentiert.

Zeitrahmen

Pro Level ist geschätzt eine Stunde nötig. Damit der User nicht mit sich wiederholenden Leveln gelangweilt wird, sind mindestens 100 notwendig. Besser wäre im Rahmen von 1000. Daraus ergeben sich mindestens 100 Stunden.

Ressourcen

Know-how von den Storyleveln.

Risiken

Es werden nicht genug Level erstellt.

Technische Machbarkeit

Das Erstellen von Leveln ist möglich, allerdings sprengt die Anzahl der Level unsere Projektzeit.

4.3.2 Generiert

Kosten

Zeitkosten zum Programmieren des Algorithmus.

Nutzen

Dem User stehen theoretisch unendlich viele Level zur Verfügung.

Zeitrahmen

Zum Programmieren des Algorithmus werden geschätzt drei Wochen benötigt. Weitere Verfeinerungen erfordern je nach gewünschter Genauigkeit zusätzliche Zeit.

Ressourcen

Online stehen einige Anleitungen zur allgemeinen Levelgeneration zur Verfügung.

Risiken

Der Algorithmus wird nicht rechtzeitig zum Projektende fertig, oder die generierten Level sind mangelhaft.

Technische Machbarkeit

Das Programmieren des Algorithmus ist auf jeden Fall innerhalb der Projektzeit möglich, die resultierende Qualität kann allerdings, bei Problemen während der Entwicklung, mangelhaft sein.

5 Bewertung der Alternativen

5.1 Vergleich der Alternativen

5.1.1 Grafik Framework

Unity 3D hat einen eigenen Workflow, ein proprietäres Datenformat und Lizenzkosten, die es für uns als Alternative unmöglich machen. Qt verwendet für 3D im Hintergrund OpenGL und bietet uns nur komplexe Abstraktionsschichten und einen an. Dadurch ist OpenGL die erste Wahl.

5.1.2 Programmiersprache

Sowohl C++ als auch Haskell haben einen zu hohen Lernaufwand im Vergleich zu Python, bieten aber keinen Vorteil, der das wiedergutmacht.

5.1.3 Level für Endlosmodus

Um den Endlosmodus wahrhaftig „endlos“ zu machen, und um dem Team auch Zeit für andere Aufgaben zu lassen, sind generierte Level zu Empfehlen.

6 Empfehlung

Aufgrund unseres absichtlich fehlenden Budgets und unserem Zeitdruck empfehlen wir OpenGL als Grafik Framework. Für die Programmiersprache empfehlen wir Python, da wir bereits Erfahrung damit haben und der Lernaufwand am geringsten ist.
Im Endlosmodus werden die Level durch einen Algorithmus generiert.