TriDat

Studienarbeit

Abteilung Informatik
Hochschule für Technik Rapperswil

Frühjahrssemester 2014

Autor(en): Nicola Jordan
Linda Urech

Betreuer: Dr. Prof. Markus Stolze

Projektpartner: HSR

triDat Dokumentation

Release 1.0

Linda Urech, Nicola Jordan

30.05.2014

Inhaltsverzeichnis

1 Abstract **1**

2 Management Summary **3**
2.1 Problem/Aufgabenstellung/Ausgangslage . . . 3
2.2 Lösung . . . 3
2.3 Schlussfolgerung . . . 4
2.4 Ausblick . . . 4
2.5 Lessions Learned . . . 4
2.6 Aufwandanalyse . . . 5
2.7 Fazit . . . 6

3 Vorwort **7**

4 Aufgabenstellung **9**

5 Projektplan **15**
5.1 Projektübersicht . . . 15
5.2 Projektorganisation . . . 15
5.3 Management . . . 15
5.4 Infrastruktur . . . 16
5.5 Qualitätsmassnahmen . . . 16
5.6 Zeitplan . . . 17

6 Vorstudie **19**
6.1 Initiale Stakeholderanalyse . . . 19
6.2 Benutzersituation . . . 19
6.3 Frameworkentscheid . . . 19
6.4 TouchScript . . . 24
6.5 Definition Projektumfang . . . 25
6.6 Prototyping . . . 27
6.7 Minimalanforderungen V1 . . . 29
6.8 Methodisches Vorgehen . . . 29

7 Anforderungsspezifikation **31**
7.1 Allgemeine Beschreibung . . . 31
7.2 Qualitätsanforderungen . . . 33
7.3 Technische Anforderungen . . . 34
7.4 Anforderungen an übrige Lieferbestandteile . . . 35
7.5 Anforderungen an durchzuführende Tätigkeiten . . . 35
7.6 Rechtlich-vertragliche Anforderungen . . . 35
7.7 Funktionale Anforderungen . . . 36
7.8 UserStories . . . 40
7.9 Personas . . . 41

8 Domainanalyse **45**
8.1 Daten . . . 45
8.2 GUI . . . 46

9 Risikoanalyse **51**
9.1 Risiko Matrix . . . 51

10 Risikomanagement **53**
10.1 Risko 1 - Finger Slip . . . 53
10.2 Risko 2 - Lag . . . 53
10.3 Risko 3 - Gestures . . . 54
10.4 Risko 4 - limitierte Anzahl Touchpunkte . . . 54
10.5 Risiko 5 - Continuous Integration . . . 54
10.6 Risiko 6 - Unity3D unterstützt keine automatisierbaren Tests . . . 55
10.7 Risiko 7 - Automatisierte Tests sind zu aufwendig . . . 55
10.8 Risiko 8 - Ausfall Entwicklungsmitglied . . . 55

11 Architektur **57**
11.1 Übersicht . . . 57
11.2 Weg eines Touchpoints . . . 57
11.3 Unity3D . . . 58
11.4 TouchScript . . . 58
11.5 Datei-Typen . . . 60
11.6 Wichtige Elemente der Architektur . . . 60

12 Testdokumentation **61**
12.1 Testing mit Unity3D . . . 61
12.2 Usability Tests . . . 62
12.3 Manuelle Tests . . . 65
12.4 Statische Code-Analyse . . . 67

13 Installationsanleitung **69**

14 Development Dokumentation **71**
14.1 Vorwort für den Programmierer . . . 71
14.2 Übersicht . . . 71
14.3 Spezielle Lösungen in triDat . . . 73
14.4 Wichtige Code Erläuterungen . . . 74
14.5 FEP (Frequently Encountered Problems) . . . 76
14.6 Bestehende Applikation Erweitern . . . 78
14.7 Nächste Schritte . . . 79
14.8 Best Practices Unity3D . . . 81

15 Erfahrungsberichte **85**
15.1 Erfahrungsbericht Linda Urech . . . 85
15.2 Erfahrungsbericht Nicola Jordan . . . 85

16 Anhang **87**
16.1 Glossar . . . 87

Abstract

Die vorliegende Arbeit resultierte in einem an Tetris angelehnten Spiel, das auf dem Multitouch-Tisch Pixelsense alleine oder zu mehreren mittels Touchinput in einer dreidimensionalen Umgebung gespielt werden kann. Umgesetzt wurde das Projekt mit Unity3D, wobei nur eine ausgewählte Funktion der eingebauten Physikengine verwendet wurde. Um dennoch ein erwartungskonformes Spielverhalten zu erreichen wurde ein entsprechend angepasstes Physikverhalten der Spielobjekte verwendet.

Im Rahmen unserer Arbeit wurde auch untersucht, wie sich gängige Softwareengineering-Techniken in Unity3D umsetzen lassen, im Speziellen wurde geprüft wie Unit-Testing und Integration Tests in Unity3D umgesetzt werden können. Es wurde festgestellt, dass Unit-Testing zwar möglich, aber mit erheblichem Aufwand verbunden ist und deshalb vor allem bei umfangreicheren Projekten eingesetzt werden sollte. Zudem waren Integrationtests, wie sie von Unity3D unterstützt werden, für dieses Projekt ungeeignet, da die Physikengine nicht genutzt wurde.

KAPITEL 2

Management Summary

> "First, solve the problem. Then, write the code."
>
> —John Johnson

2.1 Problem/Aufgabenstellung/Ausgangslage

Im Herbstsemester 2009 wurde von Mischa Trecco und Ricardo Alvarez im Rahmen einer Studienarbeit das 3D Tetris-Spiel T-Touch für den to-fuse Touchtisch der Firma to-fuse entwickelt. Dieser Tisch ist langsam etwas in die Jahre gekommen und deshalb wünschte sich die HSR ein neues Tetris-Spiel, diesmal aber für den Multitouch-Tisch Pixelsense. Da das vorhandene T-Touch ein Framework nutzt, das nur für den to-fuse Multitouch-Tisch ausgelegt ist, und deshalb nicht auf den Multitouch-Tisch Pixelsense portierbar ist, wurde entschieden eine neue Applikation zu entwickeln. Mulittouch-Spiele gibt es in der heutigen Zeit der Smartphones schon unzählige, eine neue Herausforderung war, das Spiel auf einem grossen Touchbildschirm gut bedienbar und für mehrere Personen gleichzeitig an einem Gerät spielbar zu halten. Dafür wurden diverse Paper-Prototypes angefertigt und mit Testpersonen getestet.

Das Ziel dieser Arbeit ist ein interaktives Tetris-ähnliches Spiel für den Multitouch-Tisch Pixelsense zu entwickeln. Das Spiel soll lediglich mit Touchinputs von einem, zwei und optional mehr Spielern einfach und ohne grosse Erklärungen bedienbar sein. Wichtig ist, dass neue Personen einfach ins Spiel einsteigen oder das Spiel wieder verlassen können, ohne dass jedesmal das gesamte Spiel abgebrochen und neu gestartet werden muss.

2.2 Lösung

2.2.1 Features

Zuerst wurden sogenannte Core-Features (Features, die unbedingt implementiert werden sollten) und Supplemental-Features (Features, die zusätzlich noch implementiert werden könnten) aufgelistet. Anhand dieser wurde der Umfang der Semesterarbeit festgelegt.

2.2.2 Technologie und Architektur

Ursprünglich sollte das Spiel mit XNA entwickelt werden, da Microsoft jedoch kurz vor Beginn der Arbeit durchblicken liess, dass die Weiterentwicklung von XNA definitiv eingestellt wurde, wurde nach Analyse diverser Kandidaten Unity3D als Entwicklungsumgebung gewählt, als Programmiersprache für die Behaviour-Scripts C#. Da Windows 7 keine Multitouch-Gesten unterstützt und für den Multitouch-Tisch Pixelsense zum Zeitpunkt des Entwicklungsbeginn kein Windows 8 existierte, wurde das open-source Framework TouchScript hinzugezogen, das die benötigten Multitouch-Gestures zur Verfügung stellt.

2.2.3 Design und Prototypen

Mit Hilfe von diversen Paperprototypes wurde durch Beobachtung von fünf Testbenutzern ein grobes Design des zukünftigen Spiels erstellt. Auch die Inputs des Betreuers an den jeweiligen Sitzungen wurden eingearbeitet. Anfangs wurden drei verschiedene Architektur-Prototypen erstellt, welche die grössten Risiken und Bedenken von Seiten des Entwicklungsteams und des Betreuers abdeckten.

2.3 Schlussfolgerung

Abschliessend kann gesagt werden, dass der grösste Teil der definierten Ziele erreicht wurde. Da es viele Risiken gab, die nicht einfach einzuschätzen waren, wurde von Anfang an viel Pufferzeit eingerechnet, die am Schluss für die Implementation zusätzlicher Features gebraucht werden konnte.

Was leider nicht nicht so wie geplant erreicht wurde, ist das gute Touchfeeling beim Benutzen der Applikation. Dies liegt jedoch an der Hardware (Multitouch-Tisch Pixelsense), der einen ziemlichen “Lag” aufweist, wenn eine schnellere Gesture ausgeführt wird. Dies ist auch in den diversen Beispielapplikationen von Microsoft, die mit dem Multitouch-Tisch Pixelsense ausgeliefert werden, sichtbar.

Ausserdem entschied sich das Entwicklerteam den dynamischen Modusswitch zugunsten einer einfacheren Menuführung und mehr Möglichkeiten in der ansprechenden Gestalltung der einzelnen Modi nicht wie geplant zu implementieren. Ebenfalls wurde das angestrebte automatische Testing nicht durchgeführt, da der zeitlich Aufwand zu hoch im Verhältnis zum Nutzen für dieses Projekt war.

2.4 Ausblick

Als Weiterführung dieser Arbeit könnten die Supplemental-Features, vor allem die Netzwerkfähigkeit implementiert werden. Ebenfalls interessant wäre eine Portierung auf Android oder Windows Phone, auch im Web könnte das Spiel freigegeben werden. Was in naher Zukunft (falls die HSR sich einen Multitouch-Tisch, der Windows 8 unterstützt anschaffen möchte) interessant wäre, ist eine Portierung auf Windows 8 und ein Upgrade auf TouchScript 5, das leider erst im letzten Teil der Entwicklungsarbeit veröffentlicht wurde.

2.5 Lessions Learned

Beide Teammitlieder waren zuvor noch nicht mit Unity3D vertraut, konnten sich durch die Studienarbeit einen guten Überblick über dieses Framework verschaffen. Schön war es zu erkennen, dass obwohl die Gameentwicklung (speziell mit Unity3D) für beide Teammitglieder ein neues Gebiet war, das Projekt am Ende erfolgreich abgeschlossen werden konnte. Auch die Zeitschätzung ging am Ende gut auf, obwohl es schwierig war etwas noch komplett unbekanntes nur anhand des eigenen Gefühls und Einträgen in entsprechenden Foren zu schätzen.

Auch wurde erkannt, dass bei der Entwicklung mit Touchinput die Hardware eine grosse Rolle spielt. Der Multitouch-Tisch Pixelsense hat einen gewissen “Lag” beim Erkennen von schnelleren Gesten, der bei einem Spiel im Vergleich zu einer Applikation, die hauptsächlich zum Arbeiten oder Veranschaulichen benutzt wird, negativ ins Gewicht fällt. Erschwerend kam hinzu, dass aufgrund der Update-Geschwindigkeit der Hardware ein Lag entstand. Des weiteren wurde erkannt, dass Paperprototypes sich sehr dazu eignen, herauszufinden ob ein Ablauf in einer Applikation für den Benutzer logisch und nachvollziehbar erscheint, wenn es jedoch darum geht zu testen, ob auch wirklich ein ansprechendes Touchfeeling vorhanden ist, bleibt nur die grobe Implementation des Features/Behaviours und das anschliessende Testen mit Testpersonen direkt auf dem Target-Device (in diesem Fall der Multitouch-Tisch Pixelsense). Ebenfalls ist es nur begrenzt möglich, ein mit Unity3D entwickeltes Game mit automatisierten Unit-Tests zu testen und Integrations-Tests eignen sich wegen der nicht Nutzung von “echter” Physik der UnityEngine nicht.

2.6 Aufwandanalyse

Die Studienarbeit ist 8 ECTS Credits pro Student wert, was einem Aufwand von 240 Stunden pro Student entspricht. Für die ganze Arbeit standen also theoretisch rund 480 Stunden zur Verfügung. Effektiv wurden aber 523 Stunden benötigt, was ein plus von ca 9% bedeutet. Dieser Sachverhalt wird in der Abbildung *Soll/Ist Stunden* aufgezeigt.

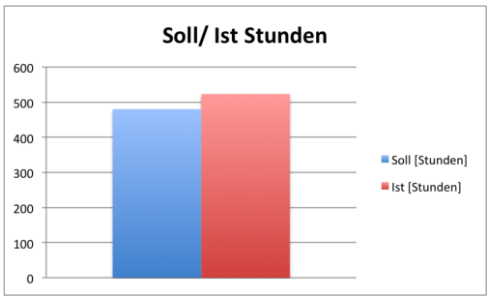


Abbildung 2.1: Soll/Ist Stunden [1]

In der Abbildung *Stundenaufteilung nach Kategorie* kann man sehen, dass rund 42% für die Dokumentation und rund 15% für administrative Aufgaben wie Meetings, deren Vor- und Nachbearbeitung, Aufsetzten von Servern, Einrichten von Git und Schätzen der Features verwendet wurde.

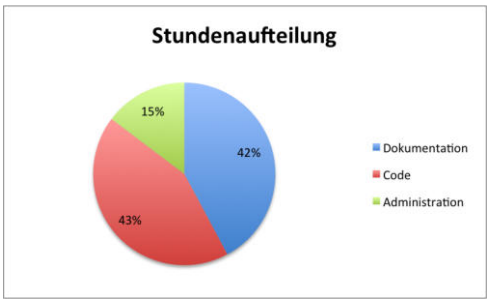


Abbildung 2.2: Stundenaufteilung nach Kategorie [2]

Aus den Abbildungen *Stundenaufteilung nach Kategorie von Nicola Jordan* und *Stundenaufteilung nach Kategorie von Linda Urech* wird ersichtlich, dass Nicola Jordan etwas mehr Zeit für Code und Administration (Aufsetzen der Server und Einrichten von Git) aufgewendet hat, während Linda Urech etwas mehr Zeit in die Dokumentation investiert hat.

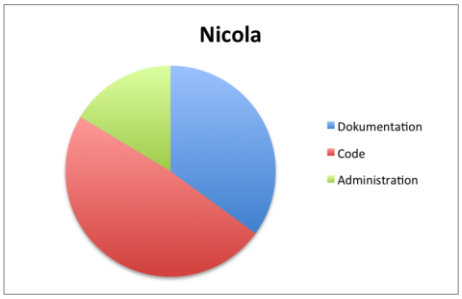


Abbildung 2.3: Stundenaufteilung nach Kategorie von Nicola Jordan [3]

Aus der Abbildung *Total aufgewendete Stunden* kann abgelesen werden, dass Nicola Jordan insgesamt rund 18 Stunden mehr Zeit aufgewendet hat.

In der Abildung *Stunden aufgeschlüsselt nach Kategorien* sind die Stunden aufgeschlüsselt nach den verschiedenen Kategorien pro Person.

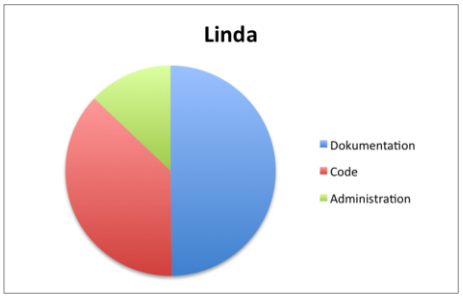


Abbildung 2.4: Stundenaufteilung nach Kategorie von Linda Urech [4]

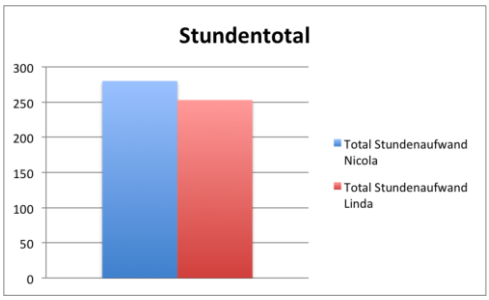


Abbildung 2.5: Total aufgewendete Stunden [5]

2.7 Fazit

Innerhalb von 14 Wochen wurde mit Unity3D ein tetrisähnliches 3D-Game für den Multitouch-Tisch Pixelsense entwickelt. Dabei wurden alle Stadien der Softwareentwicklung von der Planung über die Entwicklung bis zum Feinschliff am Ende einschliesslich diverser Test-Phasen mit Testbenutzern durchlaufen und das Team konnte wertvolle Erfahrungen für zukünftige Projekte sammeln. Obwohl nicht alle Ziele so erreicht werden konnten wie anfangs geplant, einige Anpassungen bezüglich der geplanten Methodik vorgenommen werden mussten und beim Testing der Schwerpunkt von automatisierten Unit- und Integration-Test zu manuellen Test und Usabilitytests verlegt werden musste, konnte das Projekt zu einem für das Entwicklerteam erfolgriechen Abschluss gebracht werden.

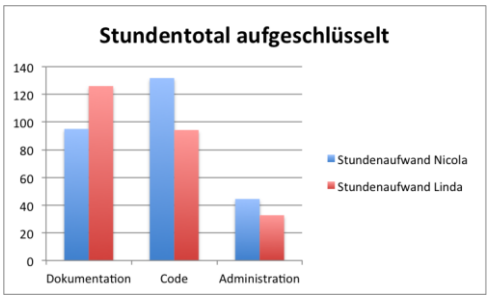


Abbildung 2.6: Stunden aufgeschlüsselt nach Kategorien [6]

KAPITEL 3

Vorwort

Da es uns ein Anliegen war, die Dokumentation für jeden Leser (auch ohne Kenntnisse von Programmierung und Unity3D) verständlilch zu formulieren, dies aber zu wenig tief gehen würde, entschieden wir uns, die Dokumentation grob in zwei Teile aufzuteilen: einen grösseren Hauptteil und einen kleineren Developmentteil.

Der Hauptteil beinhaltet alle Lieferbestandteile einer Dokumentation gemäss den Richtlinien der Hochschule für Technik Rapperswil, Abteilung Informatik. Dieser Teil richtet sich an alle interessierten Leser, die wissen möchten, was in dieser Studienarbeit gemacht wurde. Hier wird die Aufgabenstellung, das Vorgehen, die aufgetauchten Probleme und deren Lösungen, das daraus resultierende Ergebnis und vieles mehr beschrieben. Dieser Teil setzt zum Verständnis keine programmiertechnische oder Unity3D-spezifiesche Kennisse voraus.

Der zweite und kleinere Teil richtet sich zum einen an Leser, die selbst ein (ähnliches) Game entwickeln möchten und sich für die technischeren Details wie zum Beispiel programmiertechnische Konzepte, verwendete Algorithmen oder wichtige Code-Snippets interessiern. Zum anderen ist dieser Teil für Entwickler gedacht, die diese Arbeit warten, um weitere Features ergänzen oder anderweitig weiterführen möchte. Hier findet sich eine Auflistung der verwendeten Assets, wo der Code zu finden ist und architekturspezifische Details. Dieser Teil ist für Entwickler geschrieben und setzt Grundkenntisse der Programmierung und deren Konzepte im Allgemeinen, der Programmiersprache C#. Die sehr kurz gehaltene Einführung in Unity3D erlaubt es auch einem Leser, welcher nicht mit Unity3D vertraut ist, die stark mit Unity3D verknüpften Sachverhalte der Entwickler-Dokumentation zu verstehen.

Aufgabenstellung

Nachfolgend ist die eingescannte Aufgabenstellung zu finden.

Aufgabenstellung Studienarbeit Abteilung I, FS 2014
Nicola Jordan, Linda Urech

TriDat – MS Pixelsense Game

1. Betreuer & Auftraggeber

Betreuer dieser Arbeit ist
Prof. Dr. Markus Stolze, Institut für Software mstolze@hsr.ch

Auftraggeber dieser Arbeit ist die Abteilung Informatik der HSR
Prof. Dr. Markus Stolze, Institut für Software mstolze@hsr.ch

2. Ausgangslage

Die HSR Abteilung Informatik betreibt eine Ausstellung „Informatik zum Anfassen“ welche verschiedene interaktive Software-Systeme enthält. Zum grössten Teil sind dies Systeme welche von HSR Informatik-Studierenden in SE2-Projekten, Studien- oder Bachelor-Arbeiten entwickelt wurden. Die Ausstellung wird jeweils an den Info-Tagen der HSR aufgebaut und ausgestellt. Teile der Ausstellung werden auch an Besuchen durch Schulen an der HSR und bei Vorstellungen der HSR an Schulen demonstriert.

Eine wichtige Gruppe von Anwendungen sind diejenigen, welche Touch-Interaktion durch mehrere Personen gleichzeitig auf interaktiven Multitouch-Tischen ermöglichen. Diese Anwendungen eignen sich gut zur Demonstration, passen sehr gut zum Thema „Informatik zum Anfassen“ und stossen erfahrungsgemäss auf hohes Interesse bei den Teilnehmern.

Eine der ersten Anwendungen in dieser Gruppe welche an der HSR entwickelt wurde war ein Variation des Tetris Spiels bei der 2-Personen gegeneinander spielen konnten. Das besondere war hier, dass jeder Spieler seine Steine auf einer in 3D dargestellten Ebene platzieren musste. Dieses Spiel wurde für einen Touch Tisch entwickelt welcher heute nicht mehr weiter unterhalten wird. Auch das auf Java basierende (proprietäre) Touch Framework mit dem die Anwendung entwickelt wurde hat sich nicht durchgesetzt. Als Folge davon kann die attraktive 3D-Tetris Anwendung heute nicht mehr demonstriert werden.

3. Ziele der Arbeit

Das Ziel dieser Arbeit ist ein interaktives Tetris-ähnliches Spiel für den Microsoft Pixelsense Multitouch-Tisch zu entwickeln. Folgende Einschränkungen, Funktionsanforderungen und Qualitätsanforderungen sind hierbei zu erfüllen

Einschränkungen

- Optimiert für die Microsoft Pixelsense Multitouch-Hardware unter Windows 7
- Funktionstüchtig unter Windows 8

Funktionsanforderungen / Features

- Tertris-ähnliches Spiel
- Bedienung nur mittels Touch-input
- Bedienbar durch einen, zwei, oder mehr Spieler
- Einfacher Join/Leave: Neue Personen können sich sehr einfach in das Spiel einklinken und das Spiel wieder verlassen
- Einfaches und schnelles „Reset“ möglich
- Optional: Darstellung des Spielfeldes sichtbar in 3D
- Optional: Bindung der Spieler auch nach der Nutzung (zB. etwas zusenden lassen)

Qualitätsanforderungen

- Attraktivität: Flüssige Interaktion (glaubhafte „Physik“) durch 2 Personen auf dem Microsoft Pixelsense Multitouch-Tisch ist gegeben
  (Virtuelle Objekte lassen sich ohne merkliche Verzögerung manipulieren)
- Attraktivität/Einfachheit/Flow: Das Spiel lässt sich sehr einfach verstehen (spielbar auch ohne Demonstration durch eine Aufsichtsperson), wird aber nicht langweilig (Flow)
- Attraktivität: Sowohl „Gewinnen“ wie auch „Verlieren“ macht Spass.
- Stabilität: Das System sollte einen halben Tag im Spielbetrieb ohne Notwendigkeit für einen „Reset“ überstehen können (keine applikations-seitige Memoryleakes)
- Wartbarkeit: Die Software muss so geschrieben sein, dass sie einfach von Assistenten des IFS gewartet werden kann. Hierzu ist auf die Bereitstellung folgender Elemente zu achten: saubere Schichtenarchitektur, sinnvolle Dokumentation, sinnvolle Abdeckung mittels Unit-Tests und automatisierten Integrationstests (inkl. Mocking und UI Tests, ohne CI)

4. Zur Durchführung

Mit dem HSR-Betreuer finden in der Regel wöchentliche Besprechungen statt. Zusätzliche Besprechungen sind nach Bedarf durch die Studierenden zu veranlassen.

Alle Besprechungen sind von den Studenten mit einer Traktandenliste vorzubereiten und die Ergebnisse in einem Protokoll zu dokumentieren, welches stets zugreifbar ist.

Für die Durchführung der Arbeit ist ein Projektplan zu erstellen. Dabei ist auf einen kontinuierlichen und sichtbaren Arbeitsfortschritt zu achten. An Meilensteinen (gemäss Projektplan) sind einzelne Arbeitsresultate in vorläufigen Versionen abzugeben. Über die abgegebenen Arbeitsresultate erhalten die Studierenden ein Feedback. Eine definitive Beurteilung erfolgt aufgrund der am Abgabetermin abgelieferten Dokumentation. Die Evaluation erfolgt aufgrund des separat abgegebenen

Kriterienkatalogs in Übereinstimmung mit den Kriterien zur SA Beurteilung. Es sollten hierbei auch die Hinweise aus dem abgegebenen Dokument „Tipps für die Strukturierung und Planung von Studien-, Diplom- und Bachelorarbeiten“ beachtet werden.

5. Dokumentation

Über diese Arbeit ist eine Dokumentation gemäss den Richtlinien der Abteilung Informatik zu verfassen. Die zu erstellenden Dokumente sind im Projektplan festzuhalten. Alle Dokumente sind nachzuführen, d.h. sie sollen den Stand der Arbeit bei der Abgabe in konsistenter Form dokumentieren. Die Dokumentation ist vollständig auf CD/DVD in 2 Exemplaren abzugeben.

Zudem ist eine kurze Projektresultatdokumentation im öffentlichen Wiki von Prof. M. Stolze zu erstellen.

6. Weitere Regeln und Termine

Im Weiteren gelten die allgemeinen Regeln zu Bachelor und Studienarbeiten „Abläufe und Regelungen Studien- und Bachelorarbeiten im Studiengang Informatik“ (https://www.hsr.ch/Ablaeufe-und-Regelungen-Studie.7479.0.html)

Der Terminplan ist hier ersichtlich (HSR Intranet)
https://www.hsr.ch/Termine-Bachelor-und-Studiena.5142.0.html

7. Rechte

Sowohl die HSR wie auch die Studenten dürfen die entwickelte Software beliebig weiterentwickeln, öffentlich ausstellen oder auch verkaufen. Sowohl im Code als auch im User Interface soll auch bei abgeleiteten Werken aber die Autorenschaft der Studierenden nachverfolgbar bleiben. Es soll auch sichtbar bleiben, dass diese Arbeit im Rahmen einer Studienarbeit an der HSR entwickelt wurde.

8. Beurteilung

Eine erfolgreiche SA zählt 8 ECTS-Punkte pro Studierenden. Für 1 ECTS Punkt ist eine Arbeitsleistung von ca. 25 bis 30 Stunden budgetiert. Entsprechend sollten ca. 240h Arbeit für die Bachelorarbeit aufgewendet werden. Dies entspricht ungefähr 17h pro Woche (auf 14 Wochen) und damit ca. 2 Tage Arbeit pro Woche pro Student.

Für die Beurteilung ist der HSR-Betreuer verantwortlich.

Die Bewertung der Arbeit erfolgt entsprechend der verteilten Kriterienliste.

Die Aufgabenstellung wurde am 18.02.2014 vorbesprochen. Die definitive Aufgabenstellung wurde am 26.5.2014 beschlossen.

Rapperswil, 26.5.2014

Prof. Dr. Markus Stolze
Institut für Software
Hochschule für Technik Rapperswil

KAPITEL 5

Projektplan

5.1 Projektübersicht

Dieses Projekt beinhaltet die Implementation einer abgeänderten Version des allgemein bekannten Spiels Tetris.

5.1.1 Zweck und Ziel

Das Ziel ist einerseits ein Spiel zu programmieren, das auf dem Pixelsense Multitouchtisch auf Messen und Ausstellungen gezeigt werden kann. Anderseits soll die Applikation einfach zu warten und erweitern sein und ohne zusätzliche Unterstützung des Entwicklerteams auf einem anderen Multitouchtisch installiert werden können.

5.1.2 Lieferumfang

Ein tetrisähnliches Spiel, das auf einem Multitouchtisch sauber läuft, die entsprechende Dokumentation des Projektes, ein Plakat für die Ausstellung der Semesterarbeiten und ein Benutzerhandbuch mit einer Installationsanleitung.

5.1.3 Annahmen und Einschränkungen

5.2 Projektorganisation

Das Entwicklerteam besteht aus zwei einander gleichgestellten Mitglieder. Prof. Dr. Markus Stolze übernimmt dabei die Roller des Projektbetreuers.

5.3 Management

5.3.1 Projektkostenvoranschlag

Jedes Teammitglied hat insgesamt 240 Stunden pro Woche für das Projekt zur Verfügung, das Projekt wird also in rund 480 Stunden Arbeit realisiert. In diesen 480 Stunden sind Meetings, Einarbeitung, Programmierung, Dokumentation und andere anfallende Arbeiten bezüglich des Projektes inklusive.

5.3.2 Projektplan

Allfällige Änderungen im Projektplan können nach vorgängiger Besprechung mit allen Teammitgliedern vorgenommen werden.

5.3.3 Methodisches Vorgehen

Unser Ziel ist es, Agil zu entwickeln. Wir folgen grob dem Lean-IT Konzept [7], wobei wir uns am agilen Manifest orientieren [8].

Folgendes ist das Resultat der Entwicklungsstrategie für uns:

- 3 Wochen lange Sprints, wie sie aus Scrum bekannt sind, werden als Orientierungshilfe beibehalten
- Zu im Team abgesprochenen Zeitpunkten gibt es eine funktionierende Version des Spiels (mindestens einmal pro Woche)
- anstelle des Daily Meetings von Scrum wird in angemessenen Zeitabständen eine konkretere Planung für die nächsten zwei bis drei Tage erstellt und besprochen was von der letzten erreicht wurde und was nicht

Zu den Gründen, wieso wir kein Scrum verwenden, verweisen wir auf *Methodisches Vorgehen* der Vorstudie.

5.4 Infrastruktur

Das Entwicklerteam arbeitet hauptsächlich in dem von der HSR für Studienarbeiten zur Verfügung gestellten Raum 1.206. In diesem Raum steht für die Dauer des Projekts (mit wenigen Ausnahmen z.B. für eine Ausstellung) der Pixelsense Multitouchtisch für Studienzwecke und Testing zur Verfügung. Ausserdem bekommt jedes Teammitglied ein HSR-Arbeitsrechner zu Verfügung gestellt und arbeitet zusätzlich am eigenen Notebook.

5.5 Qualitätsmassnahmen

5.5.1 Dokumentation

Um für das Projekt eine hohe Qualität zu erreichen, werden alle Dokumente stets dann verfasst, wenn sie gebraucht werden und aktualisiert sobald als notwendig. Dadurch können alle Teammitglieder stehts den Projektstand einsehen und verfolgen.

5.5.2 Sitzungen

Sitzungen mit dem Entwicklerteam und dem Betreuer finden nach Notwendigkeit grundsätzlich einmal pro Woche, am Montag Nachmittag von 15:00-16:00 Uhr statt. Von jeder Sitzung wird ein Protokoll angefertigt, in welchem alle Beschlüsse schriftlich festgehalten werden. Das Protokoll wird spätestens einen Arbeitstag nach der Sitzung jedem Sitzungsmitglied elektronisch zugestellt (E-Mail) und auf dem Wiki in der Sparte Protokolle zur Archivierung abgelegt.

5.5.3 Gegenseitige Reviews

Sämtliche verfassten Dokumente und Code werden vom jeweils anderen Teammitglied gesichtet und wenn nötig, nach Rückmeldung/Besprechung korrigiert. Auf diese Weise wird die Qualität von Dokumenten und Code erhöht.

5.5.4 Risikomanagement

Das Risikomanagement wird im separaten Dokument *Risikomanagement* abgehandelt.

5.5.5 Tests

Unit-Tests

Für jeden dafür geeigneten Code sollten Unit-Test erstellt und durchgeführt werden.

Integration-Tests

Integration-Tests laufen nach den Unit-Tests und testen die Abhängigkeiten und das Zusammenspiel der einzelnen Module mit der Physik-Engine von Unity. Diese Tests sind in einer Spiele-Entwicklung deutlich schwerer zu gewichten, als bei einer "normalen" Software-Entwicklung, da viele Elemente gar nicht mit Unit-Tests abgedeckt werden können.

Usability-Tests

Vom Entwicklerteam ausgewählte Testpersonen testen die Prototypen und geben Rückmeldung bezüglich ihrer Erfahrungen. Dieses Feedback wird vom Entwicklerteam ins Projekt miteingearbeitet und soll Schwächen und Probleme im Design aufzeigen.

5.6 Zeitplan

Das agile Vorgehen bedeutet für dieses Projekt, dass nach dem groben Zeitplan ausgerichtete Arbeiten jeweils wenn passend dazu genommen werden, jedoch keine Pakete im Detail vordefiniert werden, sehr wohl aber der Rahmen und das Ziel definiert werden. Die Entwickler sprechen sich häufig (jeden Morgen) ab, was der Stand ist und was angepackt werden soll. Zudem wird die Zeiterfassung von jedem Entwicklungsmitglied selbständig erfasst und in einer Excel-Tabelle zusammengetragen.

5.6.1 Der grobe Zeitplan gliedert sich in 4 Etappen

Woche 1-3 (Nach SE: Inception + Elaboration)

Einarbeitung und Risikoabklärung, Tools und Umgebung einrichten, grösste Risiken abdecken.

Woche 4-10 (Nach SE: Construction)

Erstellung des Spiels und parallel die Dokumentation.

Woche 11-13 (Puffer)

Zeit für extra Features oder Verbesserungen des Spiels.

Woche 14-15 (Nach SE: Transition)

Erarbeitung des Posters, "finishing touches" an der Dokumentation, "Rollout".

KAPITEL 6

Vorstudie

> "Before software can be reusable it first has to be usable."
>
> —Ralph Johnson

6.1 Initiale Stakeholderanalyse

6.1.1 Auftraggeber

Der Auftraggeber ist die HSR (Hochschule für Technik Rapperswil), vertreten durch Prof. Dr. Markus Stolze, Leiter der Abteilung Informatik.

6.2 Benutzersituation

Ein Besucher einer Ausstellung kommt am Multitouch-Tisch Pixelsense vorbei und sieht triDat. Er erkennt es sofort als tetrisähnliches Spiel wieder. Interessiert begibt er sich zum Tisch, um welchen bereits einige Personen stehen. Diese scheinen zwei Gruppen gebildet zu haben und spielen gegeneinander triDat. Der Besucher gesellt sich dazu. Jemand fragt nach einem T-Block und der Besucher, der gerade einen erscheinen sieht, zieht diesen in Richtung Spielfeld des Fragenden. Nach relativ kurzer Zeit bewegt sich die Gruppe weiter und der Besucher bleibt allein am Tisch zurück. Er beendet das eine Spiel, und spielt auf dem anderen Spielfeld alleine weiter. Nach ein paar Minuten kommt ein weiterer Besucher und spielt spontan mit. Gemeinsam versuchen sie im CoOp-Modus das Spiel zu meistern.

6.3 Frameworkentscheid

6.3.1 Ausgangssituation - XNA is dead

Ursprünglich plante das Entwicklerteam triDat mit XNA zu entwickeln. Doch dann kursierten kurz vor dem eigentlichen Beginn der Arbeit im Internet in diversen Foren Gerüchte darüber, dass Microsoft XNA nicht mehr weiterentwickeln und weiterführen wird. Dies wurde von *gamasutra* am 1. Februar 2014 [1] bestätigt.

XNA, für welches die letzte Aktualisierung der Codebasis, ein sogenannter "Refresh" von der Version 4.0, 6. Oktober 2011 vorgenommen wurde, wurde auf April 2014 abgesetzt. Dies beinhaltete auch, dass ab diesem Zeitpunkt nicht mehr weiter entwickelt wurde. Für XBOX-One und Windows 8+ wird es ein Framework/Entwicklungwerkzeug geben, welches zu jenem Zeitpunkt (Februar 2014) noch unbekannt war. Zum Zeitpunkt des Abschlusses des Projekts (Ende Mai 2014) war von der offiziellen Microsoft Webseite sämtliche Dokumentationen aus dem navigierbaren Menu entfernt worden. Ein offizieller Ersatz für XNA wurde noch immer nicht bekannt gegeben, von offizieller Seite wird auf C++ mit DirectX und Unity3D als Partner verwiesen.

[1] http://www.gamasutra.com/view/news/185894/Its_official_XNA_is_dead.php

Da als eines der Hauptaugenmerke Wartbarkeit und Lebensdauer der Applikation genannt wurde, entschieden wir uns nach anderen Frameworks und Entwicklungsumgebungen Ausschau zu halten.

6.3.2 Grundanforderungen an Framework

Dass nicht alle bekannteren Tools, die es für die Gameentwicklung gibt, auch für unser Projekt geeignet waren, merkten wir schnell (so schied z.B. GameMaker schon ganz zu Beginn aus, da er zu einfach gehalten ist und wenig Möglichkeiten für eigene Ideen bietet). Daher suchten wir nach den "Must-Have"'s, die ein geeignetes Framework aufweisen sollte, dass es in die engere Auswahl kommt.

Nach einer ersten groben Evaluation blieben vier Frameworks übrig, die wir genauer untersuchen und vergleichen wollten: Monogame, OpenTK, SlimDX und Unity3D.

C# als Programmiersprache

Das Entwicklerteam entschied sich mit C# zu programmieren.

Plattform

Die fertige Applikation muss auf Windows-Plattformen im Allgemeinen und auf dem Multitouch-Tisch Pixelsense im Besonderen lauffähig sein. Ebenso sollte beachtet werden, dass die fertige Applikation mit grosser Wahrscheinlichkeit auch auf zukünftigen Multitouch-Tischen, z.B. mit Windows 8 als Betriebsystem, laufen wird.

6.3.3 Nutzwertanalyse

Ausschlusskriterien

Um die Fülle an möglichen Kandidaten zu verkleinern definierten wir die folgenden Kriterien, welche ein Kandidat unbedingt erfüllen muss, um in die nähere Auswahl zu kommen.

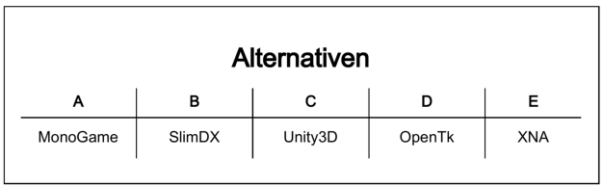
- Windows 8 muss unterstützt werden
- C# muss als Programmiersprache akzeptiert sein
- die Umgebung darf nicht zu sehr lowleveled sein

Kandidaten

XNA wurde hier in die Auswertung mit einbezogen, um zu zeigen, dass dies nach Anwendung unserer Auswahlkriterien, tatsächlich wegfällt.

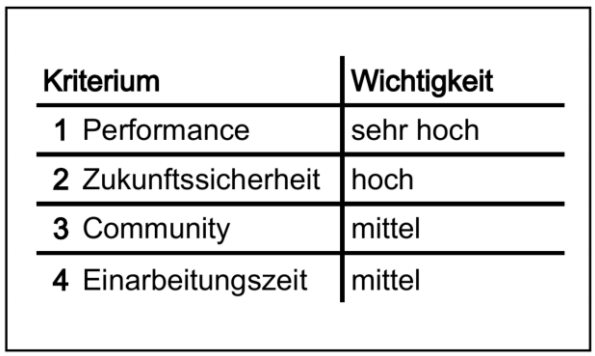
Nachdem die oben genannten Ausschschlusskriterien angewendet wurden, sind die folgenden vier Kandidaten übrig geblieben:

- **MonoGame**, eine OpenSource Implementation von Microsofts XNA (http://monogame.net/)
- **SlimDX**, eine OpenSource Library, welche Direct3D nutzt (http://slimdx.org/)
- **Unity3d**, ein kommerzielle Game-Engine mit Editor (https://unity3d.com/)
- **OpenTK**, eine Opensource Umgebung für OpenGL für .Net (http://www.opentk.com/)
- **XNA**, die Umgebung von Microsoft für Spiele-Entwicklung unter .Net (http://msdn.microsoft.com/dn629515)



Alternativen				
A	B	C	D	E
MonoGame	SlimDX	Unity3D	OpenTk	XNA

Abbildung 6.1: Nutzwertanalyse: Kandidaten [9]



Kriterium	Wichtigkeit
1 Performance	sehr hoch
2 Zukunftssicherheit	hoch
3 Community	mittel
4 Einarbeitungszeit	mittel

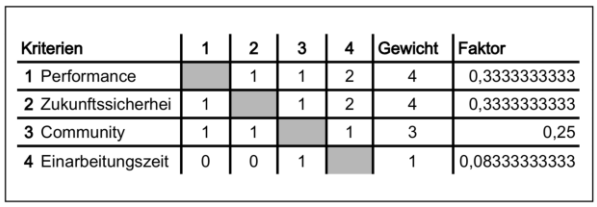
Abbildung 6.2: Nutzwertanalyse: Kriterien [10]

Kriterien

Das jeweilige Kriterium mit unserer Einschätzung, wie wichtig dies für unser Projekt ist.

Bemerkung: Die Einarbeitungszeit ist eine relativ grobe Schätzung, da noch kein Teammitglied eines dieser Tools kennt und diese Schätzung daher auf Meinungen anderer Nutzer basiert. Darum ist dieser Punkt relativ niedrig bewertet.

Berechnung der Faktorengewichtung



Kriterien	1	2	3	4	Gewicht	Faktor
1 Performance		1	1	2	4	0,3333333333
2 Zukunftssicherhei	1		1	2	4	0,3333333333
3 Community	1	1		1	3	0,25
4 Einarbeitungszeit	0	0	1		1	0,08333333333

Abbildung 6.3: Nutzwertanalyse: Gewichtung [11]

Die Gewichtungsfaktoren in den einzelnen Zellen bedeuten:

- **0**: Faktor aus der entsprechenden Zeile ist **weniger wichtig** wie der Faktor aus der entsprechenden Spalte
- **1**: Faktor aus der entsprechenden Zeile ist **gleich wichtig** wie der Faktor aus der entsprechenden Spalte
- **2**: Faktor aus der entsprechenden Zeile ist **wichtiger** wie der Faktor aus der entsprechenden Spalte

Bewertungserklärung

Die Bewertungserklärung enthält die Gründe für eine Bewertung und bildet die Grundlage zur Punktevergabe in der Auswertung.

Skala Kriterium	0-2 "schlecht"	3-5 "mittel"	6-8 "gut"
1 Performance	Touch kann Finger nicht folgen, FPS < 30, man muss alles selber Tweaken	Touch fühlt sich nicht natürlich an, kann gerade noch dem Finger folgen, FPS > 30, Leistung nur durch selber Manipulieren möglich	Natürliches Touch-Feeling, FPS 60+, gute GameEngine welche Tweaking unnötig macht
2 Zukunftssicherheit	Nicht mehr weiterentwickelte Umgebung/Framework, keine Unterstützung von moderneren Betriebssystemen (zB Windows 8), erschwerte Testbarkeit	Umgebung/Framework wird aktiv entwickelt, moderner Betriebsysteme werden möglicherweise später unterstützt, Testbarkeit gewährleistet	Aktive Entwicklung des Frameworks, moderne Betriebssysteme werden unterstützt, vereinfachte Testbarkeit (Hilfestellungen)
3 Community	Keine aktive Community, keine Dokumenation oder Tutorials	Aktive Community, Tutorials vorhanden und Dokumentation brauchbar	Gute Tutorials, schnelle Reaktion auf Fragen, viele Resourcen, mehrere Kanäle, gute Dokumentation/Wiki
4 Einarbeitungszeit	>20h	20h-10h	<10h

Abbildung 6.4: Nutzwertanalyse: Bewertungserklärungen [12]

Auswertung

Unity3D hat die höchste relative Bewertung (6.4), und ist somit der Kandidat der Wahl.

		MonoGame		SlimDX		Unity3D		OpenTk		XNA	
Kriterium	Gewichtung	Bewertung	Gesamt	Bewertung	Gesamt	Bewertung	Gesamt	Bewertung	Gesamt	Bewertung	Gesamt
Performance	33%	6	2	6	2	7	2,33333333	6	2	7	2,33333333
Zukunftssicherheit	33%	7	2,33333333	2	0,66666666	8	2,66666666	4	1,33333333	0	0
Community	25%	5	1,25	2	0,5	4	1	4	1	4	1
Einarbeitungszeit	8%	4	0,33333333	3	0,25	5	0,41666666	5	0,41666666	5	0,41666666
Total	**100%**	**22**	**5,91666666**	**13**	**3,41666666**	**24**	**6,41666666**	**19**	**4,75**	**16**	**3,75**
Cross-Plattform*		Ja		Nein		Ja		Ja		Nein	

*Cross-Plattform ist keine harte Anforderung, jedoch ein wünschenswertes Plus

Abbildung 6.5: Nutzwertanalyse: Auswertung [13]

Begründungen für die Punktevergabe

MonoGame

Kriterium	Punkte	Begründung
Performance	**6**	Sehr solide Performance, jedoch muss einiges “getunt” werden
Zukunftssicherheit	**7**	Weit verbreitet, akzeptierte Alternative zu XNA von Microsoft, aktiv entwickelt
Community	**5**	Gute Tutorials, gepflegte Community, Dokumentation etwas stark verteilt
Einarbeitungszeit	**4**	ca. 15h

SlimDX

Kriterium	Punkte	Begründung
Performance	**6**	Sehr solide Performance, viele low-level Funktionalität
Zukunftssicherheit	**2**	Es ist zwar OpenSource aber kaum genutzt, und auch nicht mehr aktiv weiter entwickelt (letzter Commit im Januar 2013, Stand: Januar 2014)
Community	**2**	Dokumentation noch von 2012, Benutzerfragen kaum beachtet, fehlende Tutorials
Einarbeitungszeit	**3**	sehr lowlevel, ca. 20h

Unity3D

Kriterium	Punkte	Begründung
Performance	**7**	Ist eine GameEngine: Für Performance entwickelt, stabil
Zukunftssicherheit	**8**	Unity3D wird von einer Firma entwickelt, welche nichts anderes macht (kaum wahrscheinlich, dass diese aufhört - höchstens Bankrott oder ähnliches)
Community	**4**	Sehr gute Tutorials, gepflegte Foren - aber: da kommerzielle Plugins ist das Interesse manchmal nicht weiterhelfen sondern "Plugin verkaufen"
Einarbeitungszeit	**5**	Graphischer Editor für Game-Objekte erleichtern den Einstieg: ca. 12h

OpenTK

Kriterium	Punkte	Begründung
Performance	**6**	Solide Performance, einige low-level Tuning Möglichkeiten
Zukunftssicherheit	**4**	Entwicklung stand über 2 Jahre still und hat erst gerade wieder ganz langsam begonnen - scheint etwas "verwahrlost"
Community	**4**	Viele alte Beiträge und offene Fragen in Foren, gutes Manual
Einarbeitungszeit	**5**	ca. 12h

XNA

Kriterium	Punkte	Begründung
Performance	**7**	Ist eine GameEngine: Für Performance entwickelt, stabil
Zukunftssicherheit	**8**	Unity3D wird von einer Firma entwickelt, welche nichts anderes macht (kaum wahrscheinlich, dass diese aufhört - höchstens Bankrott oder ähnliches)
Community	**4**	Sehr gute Tutorials, gepflegte Foren - aber: da kommerzielle Plugins ist das Interesse manchmal nicht weiterhelfen sondern "Plugin verkaufen"
Einarbeitungszeit	**5**	Graphischer Editor für Game-Objekte erleichtern den Einstieg: ca. 12h

6.3.4 Ausgeschiedene Kandidaten

- **vvvv (http://vvvv.org/)**: keine wirkliche GameNetwicklungsumgebung, entspricht also nicht unseren Ansprüchen
- **leadwerks (http://www.leadwerks.com/)**: "Nur" C++ und noch viel mehr HighLevel Entwicklung als Unity3D
- **irrlicht3d (http://irrlicht.sourceforge.net/)**: Kein Support für Windows 8 (bis jetzt, Stand Januar 2014)
- **SharpDX (http://sharpdx.org/)**: Zu sehr lowlevel (added complexity), die wir nicht brauchen

6.4 TouchScript

TouchScript ist ein Multitouch Framework welches in einer Version für Unity3D und einer für Flash ausgeliefert wird (in diesem Projekt wurde es nur für Unity3D verwendet) und wurde von Valentin Simonov bei *Interactive Lab* entwickelt. TouchScript wurde in erster Linie dazu konzipiert, auf einfache Art und Weise Multitouch Interfaces für grosse Touchflächen entwickeln zu können. Das Framework hilft die komplexen Interaktionen zwischen den verschiedenen Gesten zu managen und unterstützt verschiene Eingabemethoden. TouchScript orientiert sich bezüglich Design und Handling von Gesten an iOS und vereinfacht das Handling komplexer Gesten auf grossen Touchflächen.

TouchScript ist ein Open-Source Projekt und unterliegt den Bestimmungen der MIT Lizenz.

Unity3D weiss zwar dass es Touchinput gibt und kann diese auch registrieren, aber nicht auswerten, das heisst, Unity3D würde zwar erkennen, dass jemand die Touchoberfläche berührt, aber nicht erkennen, was der Benutzer bezwecken möchte. Windows 7, unter welchem der Multitouch-Tisch Pixelsense läuft, kennt grundsätzlich kein Touchinput, es gibt aber eine Library namens PixelSense, die als Layer über dem Betriebsystem liegt und so ermöglicht, dass das Windwos 7 auf dem Multitouch-Tisch Pixelsense mittels Touchinput bedient werden kann. Leider kennt Unity3D wiederum die PixelSense Library nicht, sprich, es wäre mit erheblichem Aufwand verbunden, diese einzubinden.

Zudem würde eigene Gestures zu definieren den Scope dieser Arbeit sprengen. Zum Glück lassen sich beide Probleme mit dem Einsatz von TouchScript lösen, welches sich um die Aufgreifung der Touch-Punkte und dessen Einteilung in Gestures kümmert.

6.4.1 Gestures

Gestures sind Bewegungen, die mit dem Finger oder einem geeigneten Gegenstand auf einer Touchoberfläche ausgeführt werden. Diese können entweder *descrete* oder *continuous* sein.

Descrete Gestures

Descrete Gestures werden erkannt und gleich abgeschlossen, ein Beispiel dafür ist die Tap-Gesture (einmaliges Berühren der Touchoberfläche mit einem Finger).

Descrete Gesture Typen

- **Tap Gesture** - erkennt einen single/double/tripple Tap
- **Press Gesture** - erkennt wenn der Benutzer ein Objekt berührt/fasst
- **Release Gesture** - erkennt wenn der Benutzer ein Objekt wieder los lässt
- **LongPress Gesture** - erkennt wenn der Benutzer ein Objekt für eine bestimmte Zeit berüht/(fest-)hält
- **Flick Gesture** - erkennt eine Flick Gesture in jede Richtung

Continuous Gestures

Continuous Gestures senden einen Event wenn sie erkannt wurden. Ein Beispiel dafür ist die Pan-Gesture (wird verwendet zum ziehen eines Objekts). Diese sendet einen Event, wenn sie als Pan-Gesture erkannt wird und danach sendet sie die Fingerpositionänderung (als sogenannte “delta changes”) solange bis sie beendet ist.

Continuous Gesture Typen

- **MetaGesture** - erkennt alle Touchevents als separate Events (nicht einer vordefinierten Gesture zugehörig)
- **(Simple) Pan Gesture** - erkennt wenn der Benutzer ein Objekt zieht

- **(Simple) Scale Gesture** - erkennt wenn der Benutzer ein Objekt skalieren möchte
- **(Simple) Rotate Gesture** - erkennt wenn der Benutzer ein Objekt dreht

Gesture States

Eine Gesture ist eine State Machine, sie bekommt Touchinputs und ändert ihren Status sobald sie ein Muster erkannt hat und weiss somit, was für ein Gesturetyp diese darstellt. Eine Gesture kennt folgende Stati:

- **Possible** - die Gesture hat das Muster noch nicht erkannt und weiss noch nicht welchen Typs sie ist
- **Began** (continuous) - die Gesture wurde erkannt und hat begonnen
- **Changed** (continous) - die Gesture hat sich geändert
- **Ended** (continuous) - die Gesture wurde beendet
- **Failed** - die Gesture konnte das Muster nicht erkennen oder wurde durch eine andere Gesture verhindert
- **Recognized** (descrete) - eine descrete Gesture hat ihr Muster erkannt

6.5 Definition Projektumfang

Aufgrund diverser Risiken und der Tatsache, dass agil entwicklelt werden soll, entschieden wir uns, das Projekt modular zu planen. Es wurden sogenannte Core-Features und Supplemental-Features definiert. Core-Features sind jene Funktionen, welche unverzichtbar sind, das heisst ohne welche das Projekt gemäss der Aufgabenstellung nicht realisiert werden könnte. Supplemental-Features sind die sogenannten “Nice-to-Haves”, die toll wären, aber nicht unbedingt notwendig. Es wurden absichtlich zu viele Supplemental-Features definiert, damit im Laufe des Projekts passende Supplemental-Features jederzeit den Core-Features modular hinzugefügt werden können. Die Supplemental-Features wurden auch laufend mit neuen Ideen wieder ergänzt.

6.5.1 Core-Features

0-n Spieler beschäftigen

Einerseits soll die Applikation ansprechend aussehen auch wenn niemand gerade am Spielen ist, andererseits sollen auch mehrere Spieler (gestestet mit 6 Personen) beschäftigt werden können. Dies wurde durch das Konzept des Fighting-Bereichs und der beiden Spielmodi erreicht, welche wahlweise ein oder zwei Spielfelder zur Auswahl stellen. Einige Spieler können im Fighting-Bereich um Blöcke kämpfen, während andere sich um das richtige Anordnen der Blöcke im Spielfeld kümmern können.

Fighting-Bereich

In der Mitte soll sich ein Bereich befinden, in welchem die neuen Blöcke im Spiel erscheinen. Die Blöcke können hier frei bewegt und gedreht werden. In diesem Bereich sollen die Spieler um die gewünschten Blöcke kämpfen können. Werden die Blöcke in die Nähe des jeweiligen Spielfelds gezogen, so “gehören” sie dem jeweiligen Spieler oder dem jeweiligen Team.

Spielfeld

Es sollen wahlweise ein oder zwei Spielfelder zur Verfügung stehen, auf denen gespielt werden kann. Die Blöcke sollen sich im Spielfeld nicht mehr zum oberen Rand hin bewegen lassen und sich selbstständig langsam zum jeweilig unteren Rand bewegen. Wird eine Rotation an einem Block ausgeführt so soll der Rotationswinkel immer 90 Grad betragen, dadurch scheint sich der Block immer am Grid auszurichten.

Modi

Es soll im Minimum zwei Modi geben, einen CoOp Modus und einen Vs Modus. Im CoOp Modus gibt es nur ein einziges Spielfeld und alle Spieler versuchen gemeinsam die Blöcke entsprechend gut anzuordnen, während es im Vs Modus zwei Spielfelder gibt und die Spieler sich in zwei Teams aufteilen können um gegeneinander zu spielen. Es wäre wünschenswert wenn es auch einen Demo Modus gäbe, in welchem ausgewählt werden kann ob der CoOp Modus oder der Vs Modus gespielt werden soll.

Dynamischer Modus Switch

Es soll möglich sein vom Vs Modus in den CoOp Modus und umgekehrt zu wechseln, ohne dass das laufende Spiel beendet werden muss.

6.5.2 Supplemental-Features

Spezialblöcke

Spezialblöcke werden aktiviert wenn sie vollständig von einem Spieler abgebaut worden sind. Sie können zum Beispiel verhindern, dass die nächsten drei Blöcke gedreht oder bewegt werden können. Ebenfalls möglich wäre, dass das gegnerische Spielfeld für einige Sekunden unsichtbar werden würde.

GameField Switch

Nach einer festgelegten Zeit, oder durch betätigen eines (virtuellen) Schalters werden die Spielfelder der Spielenden vertauscht. Jeder muss jetzt mit dem jeweils anderem Spiel weiterspielen. Die Schwierigkeit besteht darin, nur so gut zu spielen, dass bei einem Switch der Gegner nicht gewinnen kann oder umgekehrt, das eigene Spielfeld zu sabotieren in der Hoffnung, dass der Gegner mit diesem dann verlieren wird.

Netzwerkfähigkeit

Die Applikation müsste auch für Handys entwickelt werden, so dass interessierte Besucher diese auf ihr persönliches Smartphone herunterladen können (wahlweise könnte auch die HSR eine gewisse Anzahl Smartphones mit vorinstallierter Applikation zur Verfügung stellen). Auf dem Multitouch-Tisch Pixelsense würde sich dann nur noch der Fighting-Bereich befinden, das eigene Spielfeld hätte jeder Besucher auf seinem Smartphone. Eine Gruppe Besucher würde die Blöcke dann verteilen und diejenigen mit einem Smartphone könnten die verteilten Blöcke dann verwenden zum Spielen.

AI-Gegner

Ein einzelner Spieler oder ein Team von Spielern könnte gegen eine künstliche Intelligenz spielen, die unter Umständen auch verschiedene Schwierigkeitsstufen kennt.

Blöcke anpassen

Nach jeweils einer vordefinierten Zeitspanne bekommen beide Spieler/Teams einen Jocker in Form eines Hämmerchens mit welchem bei einem Block, der sich im Spielfeld befindet, ein einzelner Cube an einem Ende des Blocks “abgehackt” werden könnte. So wäre es möglich einen unpassenden Block der Situation entsprechen passend zu “hämmern”.

Sound

Die Applikation könnte mit passender Hintergrundmusik und Soundeffekten ausgestattet werden, hier müsste jedoch eine Anpassung des GUI vorgenommen werden, damit diese An-/Abgeschaltet und die Lautstärke reguliert werden könnte.

Advanced: Berührungslose Steuerung

Wie im Paper von Dr. Manju Kaushik und Rashmi Jain *Natural User Interfaces: Trend in Virtual Interaction* erläutert wird, gibt es eine Tendenz zu einer berührungslosen Interaktion zwischen Benutzer und Steuerung. Dies wäre eine Alterntive Steuerung für das Spiel, wenn man sich den Tisch wegdenkt und einen grossen Falchbildschirm an der Wand vorstellt.

Man könnte sich eine Variante vorstellen, welche zum Beispiel die Technik von Kinect der XBox verwendet, oder Stimm- und Gesten-Steuerung wie bei einigen Samsung Fernseher aufgekommen sind.

6.6 Prototyping

Nachdem die Entwicklungsumgebung gewählt und die Core-Features definiert wurden, mussten mit Hilfe der Risikomatrix die durch die Prototypen abzudeckenden Risiken bestimmt werden.

6.6.1 Prototyp 1

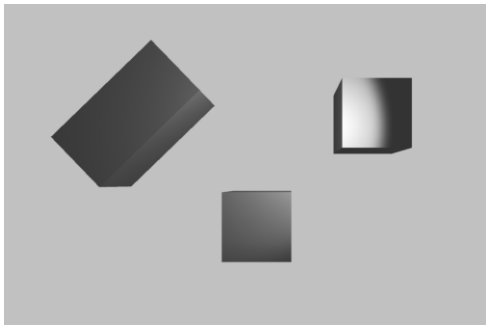


Abbildung 6.6: Prototype Gestures [14]

Der erste Prototyp deckt die Implementierung der Gestures ab. Er besteht aus - zuerst einem, später mehreren - Cubes (Würfel), welche mit einem Fingertouch bewegt und mit zwei Fingertouches rotiert werden können.

Das Bewegen der Cubes geschieht durch eine Pan-Gesture, das Rotieren mithilfe der Rotate-Gesture, die beide von TouchScript zur Verfügung gestellt werden. Ebenfalls soll getestet werden, ob beide Gestures gleichzeitig ausgeführt werden können, da, wenn eine Gesture erkannt wurde, die zugehörigen Touchpoints gesperrt werden (lock), bis die Gesture fertig ist. Falls der Benutzer aber einen Block drehen und gleichzeitig verschieben möchte, so ist es notwendig, dass die beiden Gestures sich die entsprechenden Touchpoints teilen können. Es zeigte sich, dass dies mit dem Feature der “Friendly Gestures” von TouchScript gelöst werden kann.

6.6.2 Prototyp 2 - 3 Kameras vs schiefe Ebene

Die Abbildung *Prototype 2 - 3 Kameras vs schiefe Ebene* zeigt den zweiten Prototypen.

Problemstellung

Um einen 3D-Effekt zu erzielen, gibt es 2 verschiedene Ansätze:

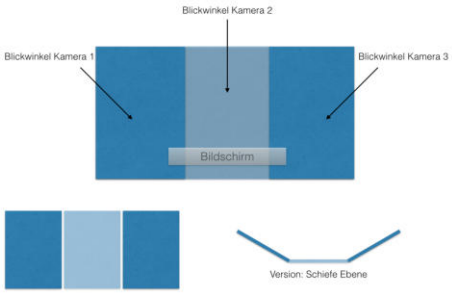


Abbildung 6.7: Prototype 2 - 3 Kameras vs schiefe Ebene [15]

- Mehrere Kameras, welche mit geteiltem Viewport eine Illusion von schiefer Ebene darstellen
- Eine Kamera, welche schiefe Ebenen zeigt (Realitätsnah)

Versuche

Variante	3 Kameras	schiefe Ebene
Vorteil	einfache Berechnung, da alle Objekte in der selben Ebene (flach)	Scaling zu verschiedenen Auflösungen ist einfacher, näher an der *Realität*
Nachteil	Für verschiedene Auflösungen müssen die Kameras jeweils wieder neu angeordnet werden	Sehr Aufwändige Berechnung, benötigt Raycasts mit Vector-Multiplikationen und lokale Gravity (Force)
Nachteil	Auffangen und Umrechnen der Touch-Points braucht komplizierte Kamera-Switch-Logik welche schwieriger wird, je mehr Objekte im Spiel sind (lösbar)	Touch-Points müssen in Objekt-Lokale Koordinaten umgerechnet werden

Diskussion, Schlussfolgerungen

Auf den ersten Blick scheint die Lösung mit den 3 Kameras die einfachere und bessere Lösung zu sein für unsere Aufgabenstellung. Das Grundproblem mit den verschiedenen Auflösungen wiegt jedoch sehr schwer: Man müsste jede Szene für jede geplante Auflösung neu Anordnen. Die erhöhte Komplexität der schiefen Ebene ist ein Risiko, welches jedoch in der Game-Entwicklung (Produkt-Unabhängig) praktisch immer vorkommt und somit in Kauf genommen werden kann.

6.6.3 Prototyp 3 - Finger Slip wenn auf schiefer Ebene

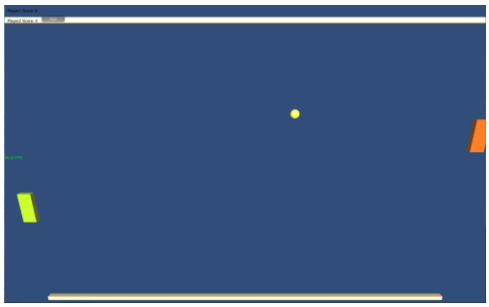


Abbildung 6.8: Prototype 3: Pong [16]

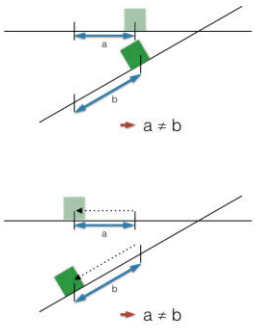


Abbildung 6.9: Fingerslip Erklärung [17]

Wie in der Abbildung *Fingerslip Erklärung* ersichtlich, ist ein bekanntes Problem, dass falls eine schiefe Ebene (in einem 3D Raum) dargestellt wird und ein Objekt auf dieser Ebene manipuliert werden soll, die Strecke, die in der 2D Ebene des Bildschirms oder der Touchoberfläche zurückgelegt wird nicht derjenigen entspricht, die auf der schiefen Ebene im 3D Raum zurückgelegt werden sollte. Unity3D löst dieses Problem mit den Methoden *Camera.ScreenToViewportPoint*, *Camera.ViewportToWorldPoint*, *Camera.ViewportToScreenPoint*, *Camera.WorldToViewportPoint* und *Camera.WorldToScreenPoint*, wobei mit "Screen" und "ViewPort" die 2D Ebene auf welcher der Benutzer arbeitet und mit "world" der 3D Raum in welchem das Spiel abläuft gemeint ist. Um dies zu testen wurde ein ganz simpler Pong-Klon, Abbildung *Prototype 3: Pong*, erstellt.

Der Prototyp konnte zeigen, dass aufgrund des Aufbaus der Unity3D Game-Engine, das Objekt nicht unter den Fingern wegrutschen kann, jedoch die Bewegungen auf dem Multitouch-Tisch Pixelsense teils etwas träge sind und zu einer leichten Verschiebung vom Objekt zum Touchpoint führen können.

6.7 Minimalanforderungen V1

In einer ersten Version von triDat muss es möglich sein zu zweit Blöcke, die zufällig im Spawn-Bereich erscheinen mittels Pan- und Rotate-Gesture zu bewegen und zu drehen und durch ziehen in Richtung Spielfeld, den Block auf das Spielfeld zu befördern. Im Spielfeld soll es möglich sein, die Blöcke am Grid ausgerichtet zu drehen und nach unten, nach links und nach rechts (nicht nach oben) zu bewegen.

6.8 Methodisches Vorgehen

Wenn von agiler Software Entwicklung berichtet wird, so fällt auch immer wieder der Begriff *Scrum*. Scrum ist eine gut etablierte Lösung für methodisches Vorgehen in der Softwareentwicklung, da es eine Struktur vorgibt, welche in einem gewissen Rahmen angepasst werden kann.

Darum war geplant, für dieses Projekt eine etwas abgeänderte Variante von Scrum zu verwenden, mit 3 Wochen Sprints:

- Backlogs werden jede 3. Woche für den nächsten Sprint (3 Wochen Dauer) festgelegt (Montags)
- Sprint dauert 3 Wochen
- "Daily"-Meeting wird jeweils Mittwochs (aka Zwischenbericht und Refinement) abgehalten
- Grooming erfolgt vor jedem neuen Sprint

Dies scheint auf den ersten Blick gut umsetzbar, jedoch musste feststellt werden, dass Scrum per Definition folgende von unserem Team (von 2 Personen) nicht erfüllbare Punkte enthält:

- Entwicklerteam von 5-7 Personen (wir sind nur 2)
- Scrum Master (ausserhalb des Entwicklerteams, dann wäre nur noch 1 Person im Entwicklerteam)
- Product Owner (Kommuniziert, was zu machen ist und gibt Feedback, fehlt komplett)

Zudem waren Daily Meetings unnötig, da wir im gleichen Raum und zur gleichen Zeit anwesend waren, und somit die anstehenden Punkte gleich zu diesem Zeitpunkt besprachen, wenn sie auftraten.

Wir haben darum entschieden, dem agilen Manifesto zu folgen, wie unter im Projektplan unter *Methodisches Vorgehen* beschrieben.

KAPITEL 7

Anforderungsspezifikation

> "Walking on water and developing software from a specification are easy if both are frozen."
>
> —Edward Berard

7.1 Allgemeine Beschreibung

7.1.1 Ziel und Zweck von triDat

Beim Projekt triDat handelt es sich um eine Version des allgemein bekannten Gameklassiker "Tetris", die für den Betrieb auf dem Pixelsense Multitouch-Tisch optimiert wurde. Die Highlights sind die Bedienung mittels Touch-Gestures und die Tatsache, dass das Spiel in einem 3D-Raum gespielt wird. triDat ist der Nachfolger für die 3D-Tetris Applikation T-Touch, die auf dem ToFuse Touchtisch der Firma to fuse (Zürich) an Ausstellungen und Messen im Einsatz war. Die neue Applikation soll nun auf dem neueren Multitouch-Tisch Pixelsense Besucher begeistern und für Spass und gute Laune sorgen.

7.1.2 Produkt Perspektive

triDat wird in der ausgelieferten Version auf dem Mulittouch-Tisch Pixelsense unter Windows 7 eingesetzt. Jedoch kann es mit wenig zusätzlichen Aufwand auch unter Windwos 8 betrieben werden. Da zum jetzigen Zeitpunkt allerdings keine offizielle Treiber für eine Windows 8 Unterstützung für den Multitouch-Tisch Pixelsense existieren, konnte die Applikation lediglich auf einem Samsung Slate Series 7 Tablet unter Windows 8 gestestet werden.

triDat wird mit dem Multitouch-Tisch Pixelsense hauptsächlich an Ausstellungen und Messen eingesetzt werden. Eine Portierung auf Android könnte in Zukunft noch durchgeführt werden. Ebenfalls wäre ein Ausbau möglich, so dass triDat auch über ein Netzwerk gespielt werden könnte.

7.1.3 Produkt Funktion

triDat ist für einen oder mehrere Spieler konzipiert, die sich in einem oder zwei Teams zusammenschliessen. Dabei kann im VS-Mode gegen einen anderen Spieler/ein anderes Team gespielt werden, oder im CoOp-Mode miteinander an einem Spielfeld gespielt werden.

7.1.4 Produkt Umfeld

Im App-Markt für Smartphone und Tablets gibt es unzählige Tetris-Klone, die heruntergeladen werden können. Obwohl es auch diverse Multiplayer-Tetris Spiele gibt, können diese nur über das Netzwerk mit anderen gespielt werden, da es aufgrund der begrenzten Bildschirmgrösse schwierig ist ein Multiplayer-Tetris, bei welchem mehrere Spieler auf einem Gerät spielen können, zu realisieren.

Microsoft hat mit dem Vorgängermodell des Multitouch-Tischs Pixelsense, dem Surface Tisch, ein Tetris-ähnliches Spiel ausgeliefert, bei welchem es darum geht gleichfarbige Quadrate in Reihen zu bringen. Bei diesem

Abbildung 7.1: Beispiel eines Tetris-Multiplayer (Twinflix) [18]

Abbildung 7.2: Multiplayer via Netzwerk (Twinflix) [19]

Spiel können die Quadrate aber nur nach links oder rechts bewegt werden, eine Rotation-Geste wird nicht unterstützt (ist in diesem Kontext auch nicht nötig, aber es wäre schön um zu zeigen was für Potenzial ein Multitouch-Tisch hat).

Abbildung 7.3: Blocks auf dem alten Surface I Tisch [20]

Im Herbstsemester 2009 haben die zwei Studenten Mischa Trecco und Ricardo Alvarez im Rahmen ihrer Studienarbeit das tetris-ähnliche Spiel T-Touch für den Multitouch-Tisch toFuse der Firma tofuse (Zürich) entwickelt. Dieses galt es mit dem Resultat dieser Semesterarbeit zu ersetzen, da der Multitouch-Tisch toFuse nicht länger mehr in Betrieb ist.

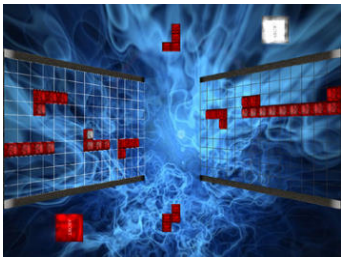


Abbildung 7.4: T-Touch [21]

7.1.5 Einschränkungen

Reaktionszeit des Multitouch-Tisches Pixelsense

Ein gutes Touchfeeling und somit ein gutes Gamefeeling steht in direktem Zusammenhang mit der Reaktionszeit der Hardware. Der Multitouch-Tisch Pixelsense weist einen gewissen Lag bei der Erkennung eines sich bewegenden Touchpoints auf, sobald sich dieser etwas schneller bewegt.

7.1.6 Prototyping

Da kein Mitglied des Entwicklungsteams Erfahrungen mit Unity3D hatte, wurde beschlossen mehrere Prototypen zu erstellen, die jeweils ein grösseres zuvor identifiziertes Problem abdecken. Insgesamt wurden 3 Prototypen erstellt. Eine genauere Beschreibung der Prototypen findet sich in der "Vorstudie" unter *Prototyping*.

7.2 Qualitätsanforderungen

7.2.1 Erwartungskonformität

Es dürfen für den Benutzer keine Überraschungen bezüglich des Verhaltens einzelner Spiel-Objekte auftreten. Alle Reaktionen des Spiels auf Benutzerinteraktionen sollen logisch nachvollziehbar sein.

7.2.2 Benutzbarkeit

Der Benutzer soll keine Einführung oder Anleitung in die Spielmechanik benötigen um spielen zu können. Ausserdem soll die Software einfach zu starten und zu beenden sein.

7.2.3 Effizienz

Der Benutzer soll kein Stocken in der Bewegung der zu bedienenden Elemente wahrnehmen. Die Framerate soll daher während dem Spielbetrieb immer mindestens 30 FPS sein.

7.2.4 Portierbarkeit

triDat soll hauptsächlich für den Multitouch-Tisch Pixelsense entwickelt werden. Es soll jedoch möglich sein, die Applikation mit geringem Aufwand auf Windows 8 zu portieren. Ausserdem soll eine Portierung für die Systeme, die von Unity3D unterstüzt werden, wie zum Beispiel Android, möglich sein.

7.2.5 Produktnutzung

Das Spiel soll als ausführbare .exe Datei geliefert werden, welche das Spiel installiert und als entzippbare Datei. Die Langlebigkeit der Applikation soll durch die saubere Architektur, die zahlreichen Code-Reviews sowie regelmässigen Refactorings des Codes gewärleistet werden.

7.2.6 Übertragbarkeit

Die Übertragbarkeit ist vollständig gewährleistet unter Einschränkung der Portierbarkeit, obwohl triDat speziell für die HSR entwickelt worden ist. Es ist anzunehmen, dass triDat auch auf anderen Pixlsense Multitouch-Tischen einwandfrei funktioniert.

7.2.7 Zuverlässigkeit

Die Applikation ist so fehlertolerant, dass falls ein Spieler eine Interaktion (z.B eine TouchGesture) durchführt, die keine Auswirkung auf den Spielverlauf haben soll, kein Fehlverhalten der Applikation entsteht. Die Applikation sollte während mindestens eines halben Tages stabil und ohne durch die Applikation selbst verursachte Abstürze laufen. Sollte dennoch in der Applikation ein Fehler auftreten, so kann das Spiel innerhalb weniger Minuten durch einen Neustart oder Reset wieder in Betrieb genommen werden.

7.2.8 Funktionalität

Die in einem Tetrisspiel sinnvoll einbaubaren Möglichkeiten des Mulittouch Tisches Pixelsense sollen ausgeschöpft werden. So soll mindestens eine Pan-Gesture, eine Rotate-Gesture und die Möglichkeit der parallelen Erkennung mehrerer Touchpunkte verwendet werden.

7.3 Technische Anforderungen

7.3.1 Hardware

Abbildung 7.5: Multitouch-Tisch Pixelsense [22]

Da triDat vorrangig für den Multitouch-Tisch Pixelsense *Multitouch-Tisch Pixelsense* entwickelt wurde, ist dieser auch die primär benötigte Hardware. Sollte jedoch eine Portierung vorgenommen werden, so ist die Hardware des Zielsystems primär.

7.3.2 Software

Als Entwicklungsumgebung soll Unity3D in der kostenlosen Version für die graphischen Elemente und MonoDevelop für die Programmierung der Behavior-Scripts verwendet werden.

7.3.3 Programmiersprache

Unity3D stellt verschiede (Script-)Sprachen zur Programmierung der sogenannten Behaviourscripts zur Verfügung unter anderem UnityScript, BooScript, (eine leicht abgeänderte Version von) JavaScript und C#. Wir haben uns für C# entschieden um die im Modul ".NET Technologien" angeeigneten Kenntnisse anzuwenden und da jeweils mindestens ein Teammitglied in den anderen Sprachen noch keine Erfahrung aufweisen konnte.

7.4 Anforderungen an übrige Lieferbestandteile

7.4.1 Softwaredokumentation

Über diese Arbeit ist eine Dokumentation gemäss den Richtlinien der Abteilung Informatik zu verfassen. Die zu erstellenden Dokumente sind im Projektplan festzuhalten. Alle Dokumente sind nachzuführen, d.h. sie sollen den Stand der Arbeit bei der Abgabe in konsistenter Form dokumentieren. Die Dokumentation ist vollständig auf CD/DVD in 2 Exemplaren abzugeben. Zudem ist eine kurze Projektresultatdokumentation im öffentlichen Wiki von Prof. M. Stolze zu erstellen.

7.4.2 Projektplan

Für die Durchführung der Arbeit ist ein Projektplan zu erstellen. Dabei ist auf einen kontinuierlichen und sichtbaren Arbeitsfortschritt zu achten.

7.5 Anforderungen an durchzuführende Tätigkeiten

7.5.1 Management

Mit dem HSR-Betreuer finden in der Regel wöchentliche Besprechungen statt. Zusätzliche Besprechungen sind nach Bedarf durch die Studierenden zu veranlassen. Alle Besprechungen sind von den Studenten mit einer Traktandenliste vorzubereiten und die Ergebnisse in einem Protokoll zu dokumentieren, welches stets zugreifbar ist.

7.6 Rechtlich-vertragliche Anforderungen

7.6.1 Vertragliche Anforderungen

Es muss gemäss Vorlage auf der HSR-Website eine Vereinbarung über Urheber- und Nutzungsrechte von Auftraggeber, HSR und den Studierenden unterzeichnet werden. Ausserdem ist eine Eigenständigkeitserklärung sowie eine Einverständniserklärung zur Publikation der Arbeit auf eprints von allen Betroffenen unterschrieben beizulegen.

7.7 Funktionale Anforderungen

7.7.1 Spielablauf

Das Spiel ist für einen, zwei oder mehrere Spieler, die sich in ein bis zwei Teams aufteilen, vorgesehen. Anfangs wird der Demo-Screen eingeblendet, auf welchem die Spieler die Möglichkeit haben zwischen zwei verschiedenen Spielmodi zu wählen. Der CoOp Modus stellt ein Spielfeld zur Verfügung und die Spieler versuchen gemeinsam mit den Blöcken möglichst viele komplette Linien zu bauen. Der Vs Modus stellt zwei sich gegenüberliegende Spielfelder zur Verfügung, die durch einen sogenannten Fightingbereich getrennt sind. In diesem Modus können die Spieler in zwei Teams gegeneinander antreten.

Das Spiel endet, wie in der klassischen Version von Tetris, wenn auf einem der Spielfelder kein Platz mehr ist zum Platzieren von weiteren Blöcken, so dass ein Teil des Blockes über das Spielfeld hinausragen würde. Alternativ kann das Spiel beendet werden, indem ein Spieler im Menu die Option “Exit to Main Menu” wählt. In diesem Fall kehrt die Applikation zum Demo-Screen zurück. Alternativ gibt es im Menu auch die Möglichkeit einen “Reset” des Spiels durchzuführen, zudem kann im Menu das Hintergrundbild (Theme) geändert werden.

7.7.2 Spielbereich

Das Design des Spielbereichs ändert sich je nach dem, in welchem Modus sich der Spieler gerade befindet. Nachfolgend werden die einzelnen Hauptkomponenten und die verschiedenen Modi beschrieben.

Fighting-Bereich

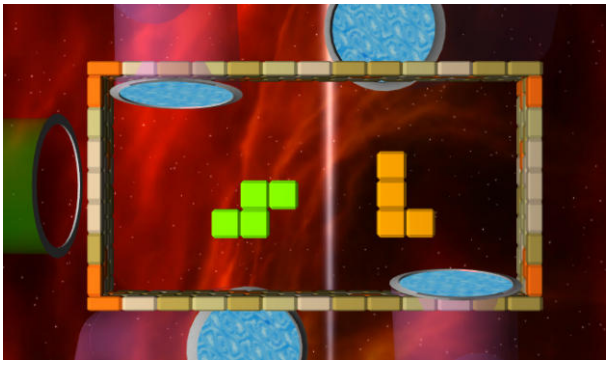


Abbildung 7.6: Fighting-Bereich [23]

Der Fightingbereich wird durch eine kleine Mauer begrenzt, so dass der Benutzer auf den ersten Blick sieht wo er die Blöcke frei bewegen kann. Eine Röhre, die ein “Wurmloch” darstellt, spuckt neue Blöcke in den Fighting-Bereich. Befindet sich eine festgelegte Anzahl Blöcke im Fighting-Bereich, so deaktiviert sich das Wurmloch und aktiviert sich wieder sobald sich wieder weniger Blöcke im Fighting-Bereich befinden. Um Blöcke, die sich im Fighting-Bereich befinden, kann gekämpft werden, indem sie mit einem oder mehreren Fingern zu dem entsprechenden “Wurmloch”, das zum eigenen Spielfeld führt, gezogen werden. Befindet sich ein Block genug nah an einem aktivierten “Wurmlock”, so wird er eingesogen. Auf dem Spielfeld dürfen sich nur eine vorher festgelegte Anzahl aktiver Blöcke befinden. Aktive Blöcke, sind jene, die noch bewegt werden können, inaktive Blöcke haben sich bereits in die vorher gesetzen Blöcke integriert.

Transportsystem mittels Wurmlöcher

Wurmlöcher bestehen aus Röhren und dienen zum Transport von Blöcken, die sich weder im Spielfeld noch im Fighting-Bereich befinden und daher vom Benutzer nicht manipuliert werden können. Wurmlocheingangsportale haben zwei Stati: aktiv und inaktiv. Ein aktives Wurmlocheingangsportal kann benutzt werden, ein inaktives ist im “Ruhezustand” und kann keine neuen Blöcke aufnehmen.

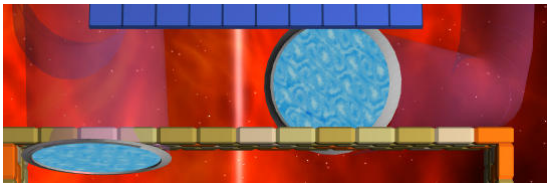


Abbildung 7.7: Wurmloch Portal [24]

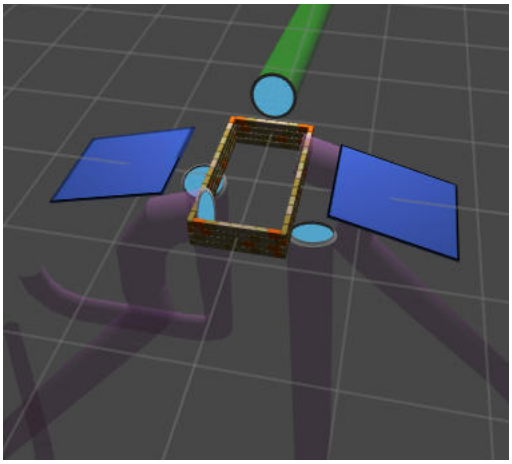


Abbildung 7.8: Röhren-System [25]

Das eine Wurmloch befördert die neuen Blöcke in den Fighting-Bereich, die anderen beiden sind zuständig für den Transport der “eroberten” Blöcke vom Fighting-Bereich zum Spielfeld. Der Benutzer kann so seine Blöcke während des gesamten Transportvorgangs verfolgen.

Auf dem Spielfeld dürfen sich nur eine festgelegte Anzahl aktiver Blöcke befinden. Aktive Blöcke sind jene, die noch bewegt werden können, inaktive Blöcke haben sich bereits in die vorher gesetzen Blöcke integriert. In den Wurmlöchern, die vom Fighting-Bereich zu den jeweiligen Spielfeldern führt, befindet sich integriert eine Queue, in welche die Blöcke kommen, die noch nicht ins Spielfeld dürfen, da sich dort bereits zu viele aktive Blöcke befinden. Diese Queue dient gleichzeitig als Vorschau für die nächsten Blöcke. Diese Queu dient auch gleich als Limit für Anzahl der Blöcke, die sich gleichzeitig im Wurmloch befinden können.

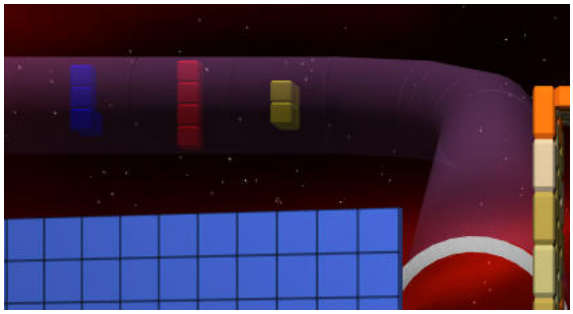


Abbildung 7.9: Queue [26]

7.7.3 Animation

Um dem Benutzer zu signalisieren, dass etwas mit dem Objekt passiert, das er gerade kontrolliert (es wird zum Beispiel zerstört, irgendwo eingesogen, verkleinert, etc) eigenen sich kurze Animationen. In unserem Fall ist dies notwendig, sobald ein Block in ein Wurmloch gezogen wird. Dies geschieht ab einem bestimmten Punkt und der Benutzer bemerkt, dass er den Block nicht mehr kontrollieren kann.

Der andere Anwendungsfall ist, wenn ein Objekt ausserhalb der Reichweite der durch den Benutzer kontrollierbare Fläche bewegt werden muss. Für uns tritt dieser Anwendungsfall auf wenn die Blöcke vom Fighting-Bereich via die Queue zum Spielbereich transportiert werden.

Menu

Abbildung 7.10: Menu [27]

Das Menu ist sehr schlicht gehalten und bietet die Möglichkeit, das laufenden Spiel zu beenden und in den Demomodus zurückzukehren, einen Reset des laufenden Spiels durchzuführen und den Hintergrund (Theme) zu ändern. Das Menu kann von jedem beliebigen Spieler benutzt werden.

Abbildung 7.11: Menu im Demomodus [28]
Im Demomodus ist das Menu in einer etwas abgespeckten Version nochmals vorhanden.

Spielfeld

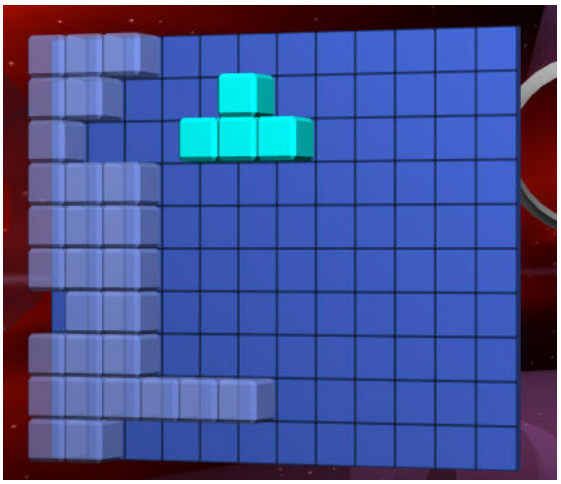


Abbildung 7.12: Spielfeld [29]

Im *Spielfeld* kann der Benutzer die vorher erkämpften Blöcke verarbeiten. Wie in der originalen Version des Spiels geht es darum, durch Drehen und Verschieben der einzelnen Blöcke möglichst viele komplette Linie zu bilden, die dann verschwinden und wieder mehr Platz für neue Blöcke im Spielfeld schaffen. Für jede Linie gibt es eine bestimmte Anzahl Punkte, so können sich die Spieler jederzeit selbst bewerten wie gut sie im Moment spielen.

Punkteanzeige

Für jedes Spielfeld gibt es eine eigene Punkteanzeige, welche den jeweiligen Punktestand anzeigt.

Demomodus

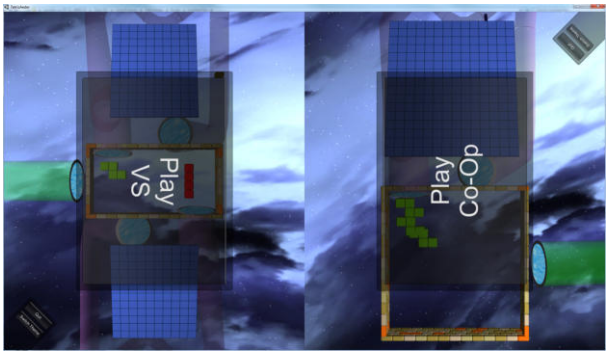


Abbildung 7.13: DemoScreen [30]

Der *Demomodus* stellt zwei Buttons zur Verfügung, mit welchen der Benutzer entweder den CoOp oder den Vs Modus starten kann.

CoOp Modus

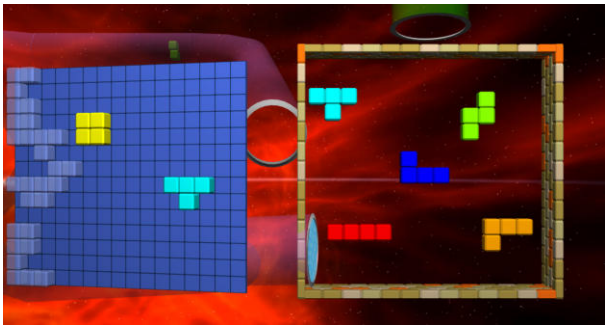


Abbildung 7.14: Co-Op Mode [31]

Der *CoOp Modus* stellt ein Fighting-Bereich und darunter liegend ein Spielfeld zur Verfügung, auf welchem einer oder mehrere Benutzer miteinander spielen können. Durch die Punkteanzeige können die Spieler ihren eigenen Fortschritt beobachten.

Vs Modus

Der *Vs Modus* stellt einen Fighting-Bereich in der Mitte und auf jeder Seite einander zugewandt zwei Spielfelder auf welchen zwei Spieler oder Teams gegeneinander antreten können. Während des gesamten Spiels gibt die Punkteanzeige Auskunft darüber wer gerade vorne liegt.

7.7.4 Blöcke

Blöcke bestehen aus jeweils vier Cubes, welche Würfel mit der selben Kantenlänge sind wie sie ein Feld im Grid des Spielfeldes hat. Die Würfel haben rundum geschnittene Kanten, ein Konstrukt, das in der Fachwelt auch als “Beveled Cube” bekannt ist. Die vier Cubes werden so kombiniert, dass die klassischen Tetris-Stein-Kompositionen entstehen. Jeder Block-Typ hat eine eigene Farbe, so dass sie schneller auf den ersten Blick unterschieden werden können.

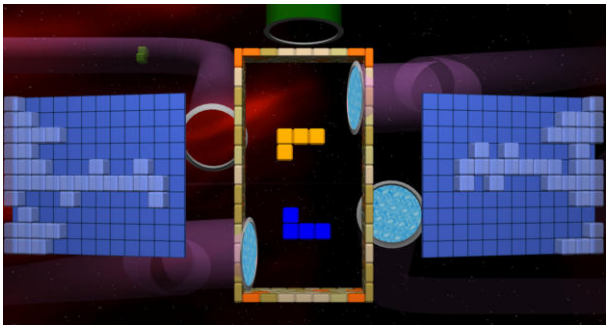


Abbildung 7.15: Vs Mode [32]

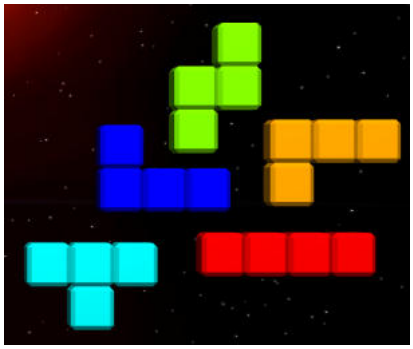


Abbildung 7.16: Verschiedene Blöcke [33]

7.7.5 Interaktionen durch Benutzer

Der Benutzer interagiert mit triDat mit Hilfe der Touchoberfläche des Multitouch-Tisches Pixelsense. Im Fighting-Bereich können die Blöcke beliebig bewegt und gedreht werden, ohne dass sich die Blöcke nach einem Grid ausrichten. Sobald sich die Blöcke auf dem Spielfeld befinden richten sie sich automatisch bei jeder Bewegung am Grid des Spielfelds aus. Ausserdem kann der Benutzer während des gesamten Spiels das Menu bedienen.

7.8 UserStories

Abbildung 7.17: Planning Board für UserStories, Features und Tasks [34]

Die Userstories wurden alle gleichzeitig bei einem Teammeeting identifiziert auf einem *Planning Board* festgehalten. Anschliessend wurde zuerst mit der ersten UserStory begonnen und kurz darauf auch mit der zweiten, da diese sehr eng zusammenhängen. Kurz nach Beginn der Arbeit an den UserStories wurde vorerst intensiv an den Prototypen gearbeitet, deren Resultate und gewonnenen Erkenntnisse in die Entwicklung der UserStories einflossen. UserStory 3 wurde erst etwas später begonnen, da diese keine elementare Spiellogik enthält.

7.8.1 UserStory 1: Im CoOp Modus spielen

[identified: 21.02.14, started: 24.02.14, tested/completed: 26.05.14]

Der Benutzer berührt im Demomodus den Button CoOp Modus und gelangt dadurch in den in den CoOp Modus. Im Fighting-Bereich werden die ersten Blöcke durch das Wurmloch in den abgegrenzten Bereich "gespuckt". Der Benutzer wählt einen dieser Blöcke aus und zieht ihn ins aktive Wurmlocheingangsportal. Anschliessend versucht er den Block durch drehen und ziehen auf dem Spielfeld geschickt zu platzieren. Hat der Benutzer eine komplette Linie gebaut, so verschwindet diese und er kriegt entsprechend Punkte gutgeschrieben. Wenn auf dem Spielfeld kein Platz für neue Blöcke mehr ist, so hat der Benutzer das Spiel verloren und das Spiel kehrt in den Demomodus zurück.

7.8.2 UserStory 2: Im Vs Modus spielen

[identified: 21.02.14, started: 24.02.14, tested/completed: 26.05.14]

Der Benutzer berührt im Demomodus den Button Vs Modus und gelangt dadurch in den in den Vs Modus. Im Fighting-Bereich werden die ersten Blöcke durch das Wurmloch in den abgegrenzten Bereich "gespuckt". Der Benutzer erkämpft sich (eventuell vom Gegner behindert) einen dieser Blöcke und zieht diesen in sein aktives Wurmlocheingangsportal. Anschliessend versucht der Benutzer den Block durch Drehen und Ziehen auf dem Spielfeld geschickt zu platzieren. Hat er eine komplette Linie gebaut, so verschwindet diese und der Benutzer kriegt entsprechend Punkte gutgeschrieben. Wenn auf dem Spielfeld kein Platz für neue Blöcke mehr ist, so ist das Spiel zu Ende und kehrt in den Demomodus zurück. Gewonnen hat wer mehr Punkte erzielt hat. Alle Benutzer können jederzeit das Spiel zurücksetzen (*Reset*) oder Beenden (*Exit to Main Menu*).

7.8.3 UserStory 3: triDat starten und beenden

[identified: 21.02.14, started: 12.03.14, tested/completed: 27.05.14]

Der Benutzer (Betreuer des Multitouch-Tisches Pixelsense) startet das Game wie eine normale Applikation. Am Ende einer Ausstellung drückt er den Button *Quit* im Demomodus und die Applikation beendet sich.

7.9 Personas

Folgende Bilder sind eigene Skizzen [35].

7.9.1 Besucher

Martin Berufsabgänger

Martin Berufsabgänger

19 Jahre alt

Lehre als Informatiker

Wird bald die Lehre mit BMS abschliessen

Interessiert sich für ein Studium

Fortgeschrittene PC-Kenntnisse

Besitzt Smartphone

Martin Berufsabgänger wird in Kürze seine Lehre als Informatiker mit BMS abschliessen und interessiert sich nun für ein Studium an der HSR.

Martin programmiert auch in seiner Freizeit sehr gerne und arbeitet momentan an einem privaten Programmier-Projekt. Mit Kollegen zockt er von Zeit zu Zeit, er überlegt sich dabei jedoch viel lieber wie die einzelnen Komponenten wohl programmiertechnisch umgesetzt wurden.

Martin ist 19 Jahre alt und weiss genau was er sich von der Zukunft erhofft, nämlich ein Studium an der HSR über das er sich jetzt informieren möchte.

Leider sind im Moment alle Ansprechpartner besetzt und Martin muss kurz warten. Er möchte sich die Zeit vertreiben, bis ein Ansprechpartner frei wird. Ihm fällt ein tetris-ähnliches Spiel ins Auge, das er sich interessiert anschauen möchte.

Christoph Gelangweilt

Christoph Gelangweilt

20 Jahre alt

Lehre als Kaufmann

Wird bald die Lehre abschliessen

Wurde von seiner Mutter an den Infotag der HSR mitgeschleppt

Gamed gerne

PC-Anwenderkenntnisse

Christoph Gelangweilt wurde von seiner Mutter genötigt, den Infotag der HSR zu besuchen, da er „was rechtes lernen“ soll. Christoph interessiert sich aber nicht wirklich für Schule und Lernen. Mit Maschinenbau, Informatik oder Elektrotechnik kann er auch nicht viel anfangen. Viel lieber spielt er mit seinen Kollegen (oder auch mal alleine wenn jene lernen müssen) Computerspiele.

Christoph hat seine Lehre als Kaufmann mit Ach und Krach abgeschlossen, die BMS hat er schon nach einem Jahr aufgegeben.

Während er mässig interessiert durch die verschiedenen Ausstellung schlendert fällt ihm ein Tetrisspiel ins Auge, das er sich genauer anschauen möchte. Nach wenigen Minuten ist er schon ganz fasziniert, eine der Ansprechpersonen fragt ihn, ob er sich für Informatik interessiere. Christoph weiss nachher noch nicht ob Informatik das richtige für ihn ist, aber er würde sich gerne mehr anschauen.

Paul Nichtgamer

Paul Nichtgamer

74 Jahre alt

Pensionierter
Abteilungsleiter Produktion

Spielt gerne Rätselspiele

Vermisst sein
Schrebergärtchen

Besitzt kein Smartphone

Skeptisch gegenüber
Technik

Paul Nichtgamer besucht mit seiner Frau den Infotag der HSR aus Neugierde und weil es mal eine Abwechslung auf ihrem täglichen Spaziergang ist. Paul war früher Leiter der Produktionsabteilung einer Firma die Kleinteile herstellt.

Paul und seine Frau hatten vor 50 Jahren ein Schrebergarten an der Stelle an der jetzt die HSR steht. Das „Tech“ kennt er seit es entstanden ist. Auf seiner Spazierroute zum See muss er immer zwischen den Gebäuden der HSR durch.

Paul löst in seiner Freizeit gerne Rätsel und allerlei Knobbeleien. Alles auf Papier. Ein Smartphone hat er nicht, geschweige denn einen Laptop. Tetris hat er schon mal irgendwo gesehen.

Am Touch-Tisch an der Ausstellung gesellt er sich zu den Jugendlichen, die sich darum drängen, da muss es ja was interessantes zu sehen geben.
Ohne gross zu überlegen berührt er einen der Blöcke mit dem Finger, als sich die Gelegenheit ergibt. Erstaunt stellt er fest, dass sich der Block mit dem Finger bewegen lässt. Das ist ja ganz einfach! Nur wohin er den Block bewegen soll, weiss er noch nicht so richtig. Einer der anderen Zuschauer erledigt Pauls Problem indem er den Block durchs Portal aufs Spielfeld zieht. Paul freut sich über seinen Erfolg eine Touch-Oberfläche bedient zu haben.

Anna Klein

Anna Klein

12 Jahre alt

Schülerin

Besucht den Infotag mit
ihrem grossen Bruder

Seit kurzen stolze
Smartphone-Besitzerin

Anna Klein ist 12 Jahre und noch Schülerin. Sie hat vor einem Monat stolz ihr erstes Smartphone bekommen. Umgehen kann sie damit aber schon lange, schliesslich hat Mami auch eines.

Anna langweilt sich am Infotag etwas, da sie nicht viel von dem versteht was die Erwachsenen da reden.

Den Multitouch-Tisch Pixelsense mit dem TriDat findet sie aber sehr interessant und kurzweilig. Seit 20 Minuten spielt sie mit und gegen die verschiedensten Leute und hat nebenbei noch erfahren was „Touch“ bedeutet, warum sie den Bildschirm vom PC im Büro von Mami nicht anfassen soll und dass TriDat theoretisch auch auf ihrem neuen Smartphone laufen würde.

Am nächsten Tag hat Anna in der Schule viel zu erzählen und ihr Bruder konnte sich ungestört über den Studiengang Elektrotechnik informieren.

7.9.2 Aussteller

Andreas Auskunft

Andreas Auskunft

24 Jahre alt

Student

Möchte mit einem Studentenjob am Infotag etwas Geld verdienen

Technisch versiert

Besitzt Smartphone

Andreas Auskunft ist Student an der HSR und kümmert sich am Infotag um die Betreuung des Multitouch-Tisches Pixelsense mit dem tetris-änlichen Spiel TriDat. Er möchte sich mit dem Studentenjob ein wenig zusätzliches Geld verdienen, kann aber aufgrund des Studiums nicht viele Stunden ins Kennenlernen des Tisches und TriDat investieren.

Andreas hat Erfahrung mit technischen Geräten und deren Umgang.

Er möchte den Tisch am Morgen mit wenigen Handgriffen in Betrieb nehmen können, den Tag durch Fragen zum Spiel/Tisch/Studium beantworten und abends den Tisch mit ähnlich wenig Aufwand wieder verräumen.

Bis jetzt musste die aktuelle Anwendung immer wieder neu gestartet werden, da sie irgendwann ins Stocken geriet.

KAPITEL 8

Domainanalyse

"Life is like Tetris; if it doesn't fit, just flip it over"

– Sabine Hein

8.1 Daten

8.1.1 Domain

BlockSpawner

Das Script BlockSpawner überprüft in jedem Durchlauf des GameLoops [1] ob sich im Fighting-Bereich bereits die maximale Anzahl Blöcke befinden. Falls nicht, wird ein ein entsprechender Block in der Methode *InstantiatePrefab()* erstellt und dem Spiel hinzugefügt. Gleichzeitig wird der Counter aktualisiert, der die aktuelle Anzahl Blöcke im Spawnbereich überwacht.

GameClock

Die GameClock ist ein Singleton, der die game-interne Zeit verwaltet. Durch unterbrechen der Zählung der Ticks mittels der Methode *Pause()* kann das Spiel angehalten werden.

MoveOnGrid

MoveOnGrid steuert das Verhalten der Blöcke, sobald sie sich auf dem Spielfeld befinden und ein Grid-Alignment gefragt ist wenn sie sich bewegen. In einem bestimmten Intervall wird der Block jeweils unaufhaltsam um eine Grid-Einheit nach unten zum Spielfeldrand bewegt. Ausserdem fängt das Skript die Pan-Gesture und die Rotate-Gesture ab. Falls eine Pan-Gesture erkannt wurde, so wird ermittelt, in welche Richtung sie ausgeführt wurde und die entsprechende Move-Methode (zum Beispiel *MoveLeft()*) im PlayerFieldBlockCoordinator. Entsprechen wird die Rotate-Gesture behandelt. Für diese wird im PlayerFieldBlockCoordinator die geforderte 90º Rotation separat berechnet (eine 75º Rotation ausgeführt durch den Benutzer wird zu einer 90º Rotation korrigiert, da sonst kein Grid-Alignment möglich ist).

Transporter

Das Transporter-Script "transportiert" die Blöcke vom Fightingbereich zum Spielfeld des jeweiligen Spielers. Ein Block der sich im Fighting-Bereich befindet hat ein anderes Bewegungsverhalten als wenn er sich im Spielfeld befindet. Das ist auch genau das, was das Transporter-Script macht: wenn ein Block mit dem "Portal zum Wurmloch" kollidiert, so wird der Block mit angepasstem Verhalten ins Spielfeld verschoben.

[1] http://docs.unity3d.com/Manual/ExecutionOrder.html

PlayerFieldBlockCoordinator

Fast jedes Game braucht ein so genannter "Manager", eine Instanz oder hier ein Script, das den Überblick über das gesamte Geschehen hat. Bei triDat ist dies das PlayerFieldBlockCoordinator-Script. Dieses kennt die Positionen aller Blöcke auf dem Spielfeld und kann daher sagen, ob ein Block zum Beispiel noch eine Grid-Einheit weiter nach unten rutschen darf, oder ob sich dort bereits ein Block befindet. Ebenso ist es verantwortlich für das effektive ausführen von Positionswechseln eines Blockes im Spielfeld

MoveFreely

MoveFreely ist das Äquivalent zu MoveOnGrid im Fighting-Bereich. MoveFreely fängt Pan- und Rotate-Gestures ab und ändert entsprechend die Position des Blockes.

Block/Cube

Das Block-Script erstellt die Spielblöcke für triDat. Diese bestehen aus einer List von Cubes, welche im Cube-Script initialisiert werden. Durch diesen Aufbau ist es relativ einfach, Farbe, Aussehen oder Form eines Blockes zu ändern.

8.2 GUI

8.2.1 Paper-Prototyping

1. Prototyp

Demo Modus

Es wurde verschiedenen Testpersonen ein Paperprototyp des Demo Modus vorgelegt. Keine der Personen hatte Probleme ein Spiel im Vs oder CoOp Modus zu starten. Allerdings viel auf, dass wenn mehrere Personen rund um den Prototyp standen, nur ca. ein viertel aller Personen den Text auf den Buttons richig herum lesen konnten. Für ca. ein viertel stand die Schrift auf dem Kopf, was zwar nicht als Problem angesehen wurde, jedoch als verbesserungswürdig.

Redesign

Aufgrund der Ergebnisse der vorangegangenen Tests wurde die Schrift auf beiden Buttons gegen den Rand des Mulittouch-Tisches Pixelsense und damit dem Benutzer zugewandt ausgerichtet. Ausserdem wurde beschlossen, die triDat-Blöcke, die sich zu Dekozwecken im Hintergrund befanden, wegzulassen, da sie zu Verwirrungen bezüglich ihres Zwecks führten.

Buttons

Im Spiel soll es verschiedene Möglichkeiten geben mit Hilfe von Buttons mit dem Spiel zu interagieren. Als Beispiel sei hier der "Quit"/"Beenden"-Button (*Abbildung vor Redesign: Quit Button*) aufgeführt. Durch drücken dieses Buttons soll vom Vs Modus zum CoOp Modus gewechselt werden können (via Demoscreen). Ursprünglich sollte dieser Button ähnlich wie ein roter Buzzer aussehen, welchen der Spieler, der gehen möchte drücken kann, so dass der andere Spieler trotzdem weiterspielen kann. In unseren Tests stellte sich jedoch heraus, dass ein Grossteil aller Testpersonen ohne vorher zu überlegen den Button drückten.

Darauf wurde der Button mit einem weissen Schriftzug "QUIT" versehen. Am Testergebnis änderte sich allerdings nicht viel, auch nicht als die Farbe von Rot zu Grün gewechselt wurde. Offensichtlich animierte die eines Buzzers nachempfundenen Form zu sehr, diesen einfach so schnell als möglich mit viel Freude zu drücken.

Aufgrund der Ergebnisse der vorangegangenen Tests wurde entschieden, die Buttons komplett aus dem spielbaren Bereich zu entfernen und stattdessen ein Menu zu entwickeln, das durch Swipen über den randnahen Bereich

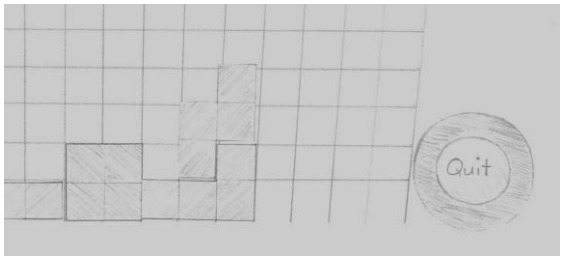


Abbildung 8.1: vor Redesign: Quit Button [36]

“heraufgeholt” werden kann. In diesem Menu sollen alle Buttons wie “Quit”/”Beenden”, “Reset”, “Theme”, etc untergebracht werden.

2. Prototyp

Menu

Nachdem klar war, dass ein einzelner Button nicht geeignet war, konzipierten wir ein Menu, in welchem die verschiedenen Aktionen gebündelt an einem Ort ausgeführt werden können. Wir orientierten uns am Design von iOS und erstellten ein Menu, das durch Swipen über den unteren Touchflächenrand nach oben gezogen und nach Gebrauch wieder ausserhalb des Bildschirms versorgt werden kann. Den Papierprototypen davon konnten auch alle Testpersonen gut bedienen. Als wir jedoch eine erste Version davon implementierten und neue Testpersonen baten, das Spiel zu beenden, fanden nur wenige das versteckte Menu obwohl es ein bisschen ins Bild ragte. Daher entschieden wir uns gegen dieses Konzept.

Pixelsense kennt eine Funktion namens *“Start Over”*, welche in den Ecken des Multitouch-Tisches aufgerufen werden kann. Dieses Konzept eignet sich jedoch nur, falls nur eine Funktion abgedeckt werden soll, in unserem Fall wären es jedoch mindestens zwei (“Quit” und “Restart”). Ausserdem findet dieses Bedienungskonzept des Ausführens einer Aktion nur bei Pixelsense Anwendung und ist daher den meisten Benutzern nicht bekannt und wird nur schwer gefunden. Zudem ist eine Beschriftung, so dass der Benutzer erkennt, was passieren wird, ungeeignet, da unklar wäre zu was sie gehört.

Abbildung 8.2: “Start Over” Funktion von Pixelsense [37]

Das nächste Problem war, dass die Toucherkennung des Multitouch-Tisches Pixelsense am Rand der Touchfläche sehr mangelhaft ist. Ein weiterer Grund dafür, dass ein eigentlich bereits erfolgreich eingesetzes Konzept in unserer Situation nicht funktionierte, war einerseits die Tatsache, dass beim Multitouch-Tisch Pixelsense im Vergleich zu herkömmlichen Smartphones die Touchfläche leicht abgesenkt ist, was verhindert, dass der Benutzer so nah am Rand die Touchfläche mit dem Tisch interagiert und anderseits die Tatsache, dass die Touchfläche des Multitouch-Tisch Pixelsense einfach viel grösser ist als die eines Smartphones oder eines Tablets und daher von Benutzern ganz anders bedient wird.

Aufgrund der Ergebnisse und Erfahrungen des vorangegangenen Test verwarfen wir die Idee eines *swipeable Menus* und entwarfen ein besser sichtbares Menu, das zwar kleiner ist und damit auch kleinere Buttons hat, dafür aber dauerhaft während des gesamten Spiels sichtbar ist.

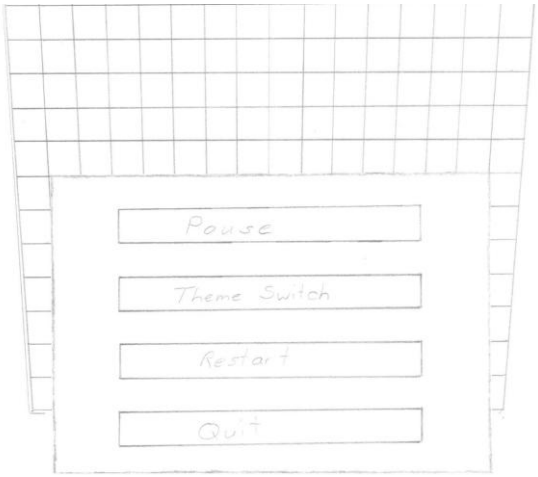


Abbildung 8.3: nach Redesign: SwipeableMenu [38]

Fightingbereich

Ursprünglich war der Fighting-Bereich in Form einer *Wolke*, in welcher die Blöcke dann erscheinen geplant. Beim Paperprototyptest kamen auch diverse positive Rückmeldungen betreffend des Wolken-Design. Bei der Implementation fiel allerdings auf, dass die zugehörigen Effekte wie zum Beispiel aufwirbeln der Wolke beim Bewegen von Blöcken entweder ungenügend aussahen und damit ein falsches Feeling vermittelten oder ansonsten schlicht zu viel Rechenleistung brauchten. Ausserdem war nach einer Testimplementation für einige Testpersonen unklar warum ein Block nicht aus der Wolke herausbewegt werden konnte. Obwohl das Wolken-Design super aussah, war es in unserem Fall zur Umsetzung nicht geeignet.

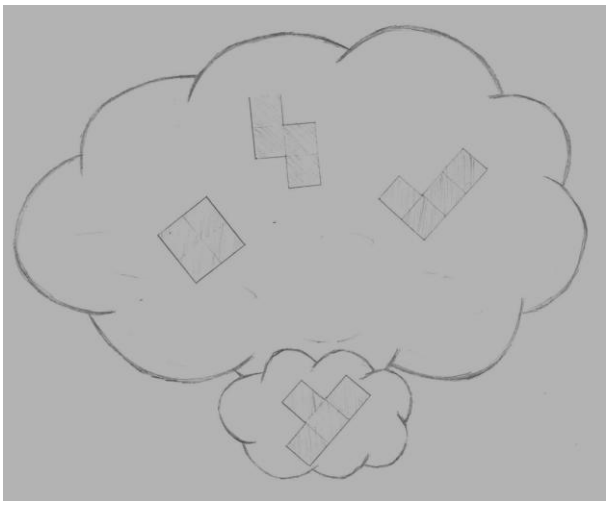


Abbildung 8.4: vor Redesign: Wolke [39]

Wir entschieden uns das ganze Thema “Himmel” etwas anders umzusetzen und so entstand das Thema “Space/Weltraum”. Die Wolke wurde durch eine kleine Mauer ersetzt, so dass der Spieler genau nachvollziehen kann, warum er den Block nicht aus dem Fightingbereich herausbewegen. Über diese Mauer führt ein röhrenartiges “Wurmloch”, welches neue Blöcke in den Fighting-Bereich transportiert. Pro Spielfeld führt ein weiteres Wurmloch wieder aus dem Fighting-Bereich heraus. Der Spieler kann seinen erkämpften Block nun also in ein Wurmloch bewegen, statt wie vorher durch ein Portal.

8.2.2 Touch Interaction Guidelines

Nachfolgend findet sich eine Liste von Touch Interaction Guidelines, die für grosse Touchflächen und für unsere Applikation mehr oder weniger wichtig sind.

Buttons

Buttons müssen eine gewisse Grösse haben, dass sie einfach getroffen werden können, konkret müssen sie mindestens so gross wie einen Touchpunkt am Finger sein.

Blickwinkel des Benutzers beachten

Je nach dem um was für ein Gerät es sich handelt betrachtet der Benutzer die Touchoberfläche aus verschiedenen Winkeln. Da der Multitouch-Tisch Pixelsense durch das Bedienen im Stehen hauptsächlich von oben betrachtet wird, ist diese Guidline für uns nicht sehr relevant.

Direkte Manipulation von Objekten

Benutzer sind viel engagierter bei der Sache wenn sie Objekte direkt mittels Gesten manipulieren können und nicht mit separaten Controls. Direkte Manipulation wird wahrgenommen wenn

- mittels Gesten Objekte manipuliert werden können
- wenn eine Aktion sofort sichtbare Resultate mit sich bringt

Siehe Usability Guidelines von Apple [2].

8.2.3 Accessability

Es ist keine Unterstützung für Blinde und Sehbehinderte vorgesehen. Farbenblinde können triDat jedoch trotzdem spielen, da die Farben jeweils nur zur Unterstützung des Erkennens um welchen Blocktyp es sich handelt oder als Hintergrund verwendet werden.

[2] https://developer.apple.com/library/iOs/documentation/UserExperience/Conceptual/MobileHIG/Principles.html

KAPITEL 9

Risikoanalyse

> "If builders built buildings the way programmers wrote programs, then the first woodpecker that came along would destroy civilization."
>
> —Gerald Weinberg

9.1 Risiko Matrix

Tabelle 9.1: Risikoanalyse

Risiko	Bezeichnung	EW	Stufe	Massnahme
1	Finger slip, Objekte können nicht gehalten werden	100%	20	Prototyp
2	Lag (nachziehen) des Objektes	100%	20	Prototyp, Anforderungen ändern
3	Eigene Gestures entwickeln müssen	70%-80%	15	Mehr Entwicklungszeit einplanen, Evaluation vorhandener Plugins
4	Nicht genügend Touchpunkte werden erkannt	20%	12	Prototyp für Touch
5	Continuous Integration kann nicht erstellt werden	100%	5	CI wird weggelassen ohne grosse Konsequenzen
6	Unity3D unterstützt keine automatisierbare Tests	40%	15	Automatisierte Tests werden weggelassen
7	Automatisierte Tests sind zu aufwendig	80%	15	Automatisierte Tests werden weggelassen
8	Entwicklerteammitglied wird krank/fällt aus etc	80%	5	Puffer einberechnen, Dokumentation ist immer nachgeführt, jeder weiss was der andere tut

EW Eintritts-Wahrscheinlichkeit

Stufe 1-25, siehe *Stufenerklärung*

9.1.1 Stufenerklärung

Die Risiken werden in einer Skala von 1-25 eingeteilt. Die Aufschlüsselung kann der Abbildung *Risiko Matrix* entnommen werden.

Abbildung 9.1: Risiko Matrix [40]

9.1.2 Risiko Strategie

Die Behandlungsstrategie für die anfallenden Risiken wird im Diagramm *Behandlungsstrategie* erläutert.

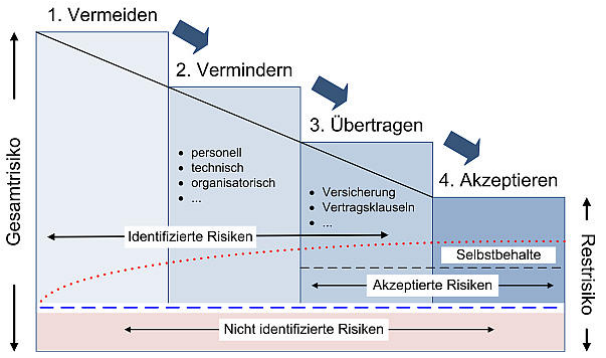


Abbildung 9.2: Risiko Behandlungsstrategien [41]

9.1.3 Risiko Prozess

Die Methode zur Umgang von Risiken wird im Diagramm *Risiko Prozess* behandelt.

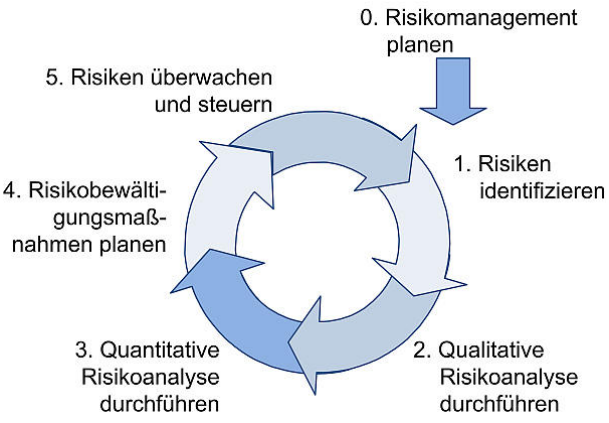


Abbildung 9.3: Risiko Prozess [42]

KAPITEL 10

Risikomanagement

10.1 Risko 1 - Finger Slip

10.1.1 Risiko

Die Objekte könnten aufgrund der Tatsache, dass sie virtuell auf einer schiefen Ebene bewegt werden, dem Benutzer beim Interagieren das Gefühl hervorrufen, sie würden unter seinem Finger wegrutschen. Diese Illusion kann zustande kommen, da der effektive Weg, den ein Objekt auf der Ebene zurücklegen müsste nicht derselbe ist, wie ihn der Finger des Benutzers auf der Oberfläche macht.

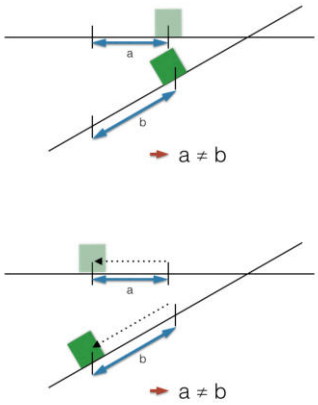


Abbildung 10.1: Risiko Fingerslip [43]

10.1.2 Management

Dieses Risiko wird vor Beginn der eigentlichen Entwicklungsarbeiten durch einen Prototyp abgedeckt. Falls das Risiko eintrifft muss eine Umrechnung der Koordinaten von der Oberfläche zur schiefen Ebene entwickelt werden.

10.2 Risko 2 - Lag

10.2.1 Risiko

Falls die Update-Frequenz für visuelle Komponenten des Multitouch-Tisches Pixelsense zu tief ist, wird dem Benutzer das Gefühl vermittelt, das Objekt würde bei schnelleren Gesten hinter seinem Finger “herhinken”.

10.2.2 Management

Dieses Risiko wird vor Beginn der eigentlichen Entwicklungsarbeiten durch einen Prototyp abgedeckt. Trifft das Risiko ein, müssen die Anforderungen entsprechend angepasst werden, da die Hardware nicht ausgewechselt werden kann aufgrund der Aufgabenstellung, die eine Gameentwicklung für den Multitouch-Tisch Pixelsense vorschreibt.

10.3 Risko 3 - Gestures

10.3.1 Risiko

Da Unity3D selbst keine Gesten erkennen kann, muss ein Framework gefunden werden, das diese Aufgabe übernimmt. Wird keine entsprechende Lösung gefunden muss die Gestenerkennung im schlimmsten Fall selbst entwickelt werden.

10.3.2 Management

Müsste die Gestenerkennung selbst entwickelt werden so würde dies einen erheblich grösseren Aufwand bedeuten. Diesem Risiko wird entgegengewirkt indem ein Prototyp entwickelt wird, der die notwendigen Gesten abdeckt und genügend Zeit eingeplant wird für den Fall, dass doch eine eigene Lösung entwickelt werden muss.

10.4 Risko 4 - limitierte Anzahl Touchpunkte

10.4.1 Risiko

Es könnte sein, dass der Multitouch-Tisch Pixelsense nicht fähig ist, genügend “simultaneous Touchpoints” zu erkennen um die angestrebte Anzahl gleichzeitiger Benutzer zu unterstützen. Pro Benutzer wird mit durchschnittlich fünf simultaneous Touchpoints gerechnet, da ein Benutzer einen Block nur schon aufgrund dessen relativ zum Finger kleinen Grösse nicht mit mehr als zwei Fingern bedienen wird. Ein Benutzer hat normalerweise zwei Hände und kann daher zwei Blöcke gleichzeitig unabhängig voneinander bewegen, zusammen ergibt das durchschnittlich vier Touchpoints. Der Multitouch-Tisch Pixelsense kann aber auch Berührungen mit dem Handballen als Touchpoint interpretieren, daher gilt die Annahme, dass pro Benutzer mit durchschnittlich fünf simultaneous Touchpoints gerechnet werden kann. Zudem muss ein Block um weiterzuspielen wieder losgelassen werden, was wiederum die Touchpoints auch wieder frei gibt.

10.4.2 Management

Dieses Risiko kann mit einem Prototyp abgedeckt werden, welcher aus diversen Objekten besteht, die von der gewünschten Anzahl Benutzer bewegt werden sollen. Ausserdem findet sich in der Spezifikation des Multitouch-Tischs Pixelsense der Hinweis, dass mindestens 50 simultaneous Touchpoints erkennt werden können.

10.5 Risiko 5 - Continuous Integration

10.5.1 Risiko

Unter Umständen kann die Umsetzung von Continuous Integration aufgrund von zu hohem (Initial-) Aufwand und entsprechend zu wenig Nutzen für dieses Projekt nicht durchgeführt werden.

10.5.2 Management

Im Falle des Eintretens dieses Risikos wird Continuous Integration ohne einschneidende Konsequenzen weggelassen.

10.6 Risiko 6 - Unity3D unterstützt keine automatisierbaren Tests

10.6.1 Risiko

Mit Unity3D können keine automatisierten Test erstellt werden.

10.6.2 Management

Automatisierte Tests werden bei Eintreffen dieses Risikos weggelassen und durch vermehrte manuelle Test kompensiert.

10.7 Risiko 7 - Automatisierte Tests sind zu aufwendig

10.7.1 Risiko

Unter Umständen kann die Umsetzung von automatisierten Tests aufgrund von zu hohem (Initial-) Aufwand und entsprechend zu wenig Nutzen für dieses Projekt nicht durchgeführt werden.

10.7.2 Management

Automatisierte Tests werden bei Eintreffen dieses Risikos weggelassen und durch vermehrte manuelle Test kompensiert.

10.8 Risiko 8 - Ausfall Entwicklungsmitglied

10.8.1 Risiko

Wird ein Entwicklungsmitglied krank oder ist sonst verhindert, so kann unter Umständen der Zeitplan nicht eingehalten werden. Ebenfalls können dann die gemeinsamen Gespräche zur Orientierung des Stands nicht abgehalten werden.

10.8.2 Management

Damit jedes Mitglied auch von Zuhause aus arbeiten könnte wird darauf geachtet, dass die Dokumentation up-to-date gehalten und wichtige Neuerungen/Änderungen über elektronische Kommunikationswege mitgeteilt werden. Auf diese Weise bleiben die Entwicklungsteammitglieder immer in Kontakt und können auf diese Weise trotz Abwesenheit von der kleinen Teamgrösse und den daraus resultierenden Vorteile wie die einfachen Gespräche profitieren. Zusätzlich werden alle Arbeiten grosszügig geschätzt, damit ein temporärer Ausfall eines Entwicklungsteammitglieds keine gravierenden Auswirkungen auf den Projektplan haben.

KAPITEL 11

Architektur

> "There are two ways of constructing a software design: One way is to make it so simple that there are obviously no deficiencies, and the other way is to make it so complicated that there are no obvious deficiencies. The first method is far more difficult."
>
> —C.A.R. Hoare

11.1 Übersicht

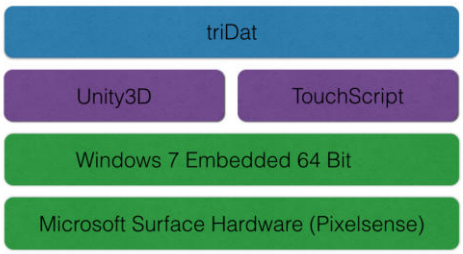


Abbildung 11.1: Architektur Übersicht [44]

Auf dem Multitouch-Tisch Pixelsense mit Windows 7 läuft die Unity3D Game-Engine mittels der wir das Spiel triDat via die TouchScript Library bedienbar machen können. Die Unity3D Game-Engine treibt das Spiel triDat an, während die TouchScript Library dafür sorgt, dass die Touchinputs korrekt interpretiert werden, die zuvor von Windows 7 generiert und dann von der Game-Engine durchgereicht wurden.

11.2 Weg eines Touchpoints

Wenn der Benutzer mit der Touchoberfläche interagiert, so generiert Windows 7 ein Touchpoint-Event, der nach oben gereicht wird. Die Game-Engine von Unity3D bekommt diesen Event, wertet diesen aber selber nicht aus, sondern gibt ihn an TouchScript weiter, welches die Gesture, die ausgeführt wurde, interpretieren kann. Nachdem der Gesture-Typ eindeutig bestimmt wurde, wird das Ergebnis an die Applikation triDat weitergeleitet, die zum Beispiel die entsprechende Transformation des Spiel-Objekts berechnet und darstellt.

11.3 Unity3D

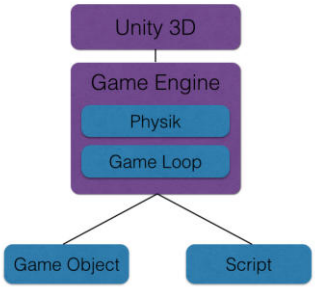


Abbildung 11.2: GameEngine, Script und GameObjects [45]

Die Schnittstelle zwischen triDat und Unity3D besteht aus einem Zusammenspiel von zwei Komponenten: die eine Komponente ist zuständig für die Darstellung und die andere für die Tranformationen eines Objektes, wie zum Beispiel Grösse, Form, Position oder Rotation. Unity3D erhält von Windows 7 einen Touchpoint-Event und reicht diesen an TouchScript weiter. Die Game-Engine ist dafür zuständig, dass wenn die Applikation triDat einen Touchpointevent erhält und entschieden hat, was mit dem entsprechenden Game-Objekt geschehen soll, diese Anweisung (zum Beispiel eine Transformation) bei Aufruf der nächsten Interation des Game-Loops ausgeführt und in der darauffolgenden Iteration dargestellt wird.

11.3.1 Game Engine

Die Game Engine Komponente von Unity3D übernimmt die eigentliche Physik, die in einem Spiel verwendet werden kann. Jedes Game Object lebt sozusagen in der Game Engine. Die Game Engine ist auch zuständig für den Game Loop. Der Game Loop wird durch die (Fixed)Update Methode realisiert. Diese wird regelmässig (bei jedem Durchlauf des Game Loops) ausgeführt.

11.3.2 Script

Mit Hilfe der sogenannten Behaviour Skripts kann das Verhalten eines einzelnen Game Objects gesteuert werden. Das Script wird an ein Game Object angehängt und kann so auf alle Komponenten dieses Game Objects (wie zum Beispiel den RigidBody) zugreifen und diese - wo zugelassen - verändern. Eine Vorwärtsbewegung wird zum Beispiel durch Manipulation der absoluten Position des GameObjects im Raum erreicht.

11.3.3 GameObjects

Game Objects sind die grundlegenden Bausteine aller Gegenstände in einem Spiel. Es gibt verschiede vorgefertige Objekte wie zum Beispiel Plane, Cube, Sphere, etc, die alle von Game Object ableiten. Ein GameObject kann beliebig durch Anfügen von zusätzlichen Komponenten wie zum Beispiel Texturen, Materialien, Behaviour Scripts, Effekte, RigidBody, etc. verändert, ergänzt und dem persönlichen Bedarf angepasst werden.

11.4 TouchScript

Jeder Touchpoint muss irgendwo generiert worden sein und dann via die in *Übersicht TouchPoint* eingezeichnete InputSource in den Zuständigkeitsbereich von TouchScript gelangen. Der eingezeigchnete Remapper wird notwendig, falls das InputDevice nicht korrekt zum Screen ausgerichtet ist. In diesem Fall korrigiert der Remapper

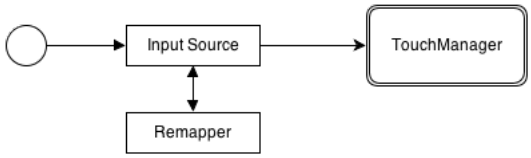


Abbildung 11.3: Übersicht TouchPoint [46]

das Mapping bevor der Touchpoint zum TouchManager weitergereicht wird. Da TouchPoints theoretisch auch zwischen zwei Frames entstehen können, sammelt der TouchManager alle TouchPoints, die er innerhalb eines Frames bekommt und verarbeitet sie jeweils alle am Ende dieses.

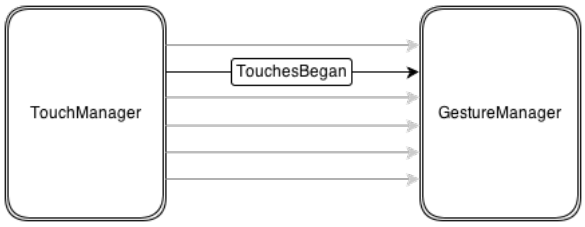


Abbildung 11.4: Übergang TouchManager - GestureManager [47]

Jeder TouchPoint ist einem Target (Spiel-Objekt) zugeordnet. Wenn dieses bestummen wurde für einen eingehenden TouchPoint, so wird dieser dem GestureManager weitergereicht. Dieser überprüft ob eine Gesture auf dem Target an diesem TouchPoint interessiert ist. Dies ist ein etwas komplizierterer Vorgang.

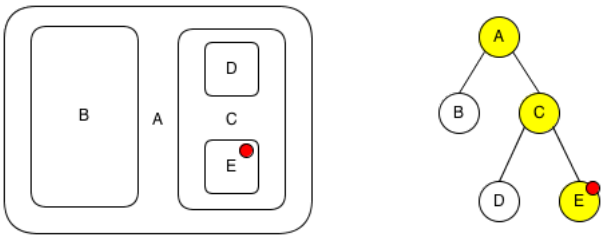


Abbildung 11.5: Zuordnung eines TouchPoints [48]

In der Abbildung *Zuordnung eines TouchPoints* ist als Beispiel die Situation mit verschachtelten Boxen dargestellt. Wenn der Benutzer Box E berührt, so überprüft das System alle Gestures, die an Objekten hängen, die an diesem TouchPoint interessiert sein könnten (in diesem Fall die Boxen E, C und A). Falls sich im daraus entstehenden Graphen keine aktive Gesture befindet, die am Target zugeordnet ist, bekommen alle Gestures den TouchPoint, bis eine ihren Status zu *Recognized* oder *Began* ändert (für weitere Informationen bezüglich Stati der Gestures siehe Vorstudie - *Gestures*).

Nehmen wir an, dass Box E den TouchPoint vom oben erläuterten Beispiel bekommen hat und den Gesture Modus in *Started* gewechselt hat. Der TouchPoint gehört jetzt zu dieser Gesture und alle Gesture, die nicht friendly sind werden gezwungen zu *Failed* zu wechseln oder sich zu resetten.

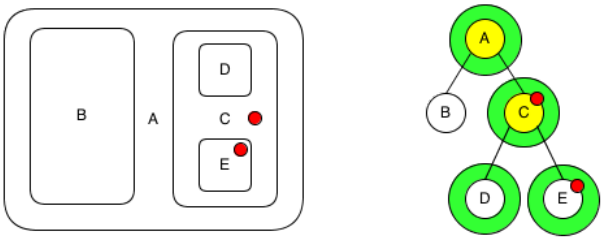


Abbildung 11.6: Zuordnung eines TouchPoints in einer Hierarchie [49]

Als Weiterführung des oberen Beispiels: der Benutzer berührt nun Box C, die ein Parent-Container ist. Das System nimmt nun alle Gestures der gelb markierten Objekte und vergleicht sie mit denen der grün eingekreisten Objekten. In diesem Fall ist eine Gesture von Box E aktiv und verhindert das die Gestures von C und weiter oben starten können.

Damit Gestures in Hierarchien miteinander zusammenarbeiten können, müssen sie jeweils zueinander *friendly* sein, dann können sie sich TouchPoints, die die eine Gesture bereits benutzt untereinander teilen und eine andere Gesture kann den TouchPoint auch benutzen.

11.5 Datei-Typen

Ein Unity3D-Projekt besteht aus verschiedenen Datei-Typen und kann mit verschiedensten Datei-Formaten umgehen. Im folgenden sind diejenigen Aufgelistet und kurz beschrieben, welche im triDat verwendung gefunden haben.

.unityproj Unity3D Projekt Datei

.cs Eine C# Source-Code Datei

.blend Eine Blender Datei, welche mit dem Programm Blender erstellt wurde, und in Unity3D importiert werden kann

.svg Vektorgrafiken

.png, .jpg Bilder

.pdf Portable Document Format von Acrobat

.doc, .ppt, .docx, .pptx .keynote Text und Präsentationsdokumente von Microsoft und Apple

.3ds Autodesk Dateiformat für 3D-Modelle

.mat Material Datei von Unity3D

.xcf Gimp Bildbearbeitungsprogrammdatei

11.6 Wichtige Elemente der Architektur

11.6.1 Gebrauch von Singletons

Der häufige (und oft ungerechtfertigte) Gebrauch von Singletons, hat diesen einen etwas zwielichtigen Ruf eingebracht. Oftmals ist lieber auf sie zu verzichten, in unserem Fall sind Singletons jedoch als “Manager” durchaus sinnvoll, da es nicht mehrere Instanzen eines Managers geben soll. Sinnvoll sind Singletons konkret zum Beispiel als GameClock oder BlockCounter, der die aktuelle Anzahl Blöcke, die sich in einem bestimmten Bereich befinden, zählt.

11.6.2 Physik

Physikbehaftete Bewegungen werden durch Kraftvektoren beschrieben, welche eine Richtung und eine Kraft besitzen. Um eine genaue Bewegung zu beschreiben, muss also der Reibungskoeffizent mit einbezogen werden, damit von Punkt A nach Punkt B eine Verschiebung stattfindet. Dies ist jedoch sehr aufwändig und sorgt für unnötigen Overhead, wenn das einzige Ziel eine möglichst punktgnaue Verschiebung ist. Aufgrund dieser Erkenntnisse wurden nahezu sämtliche physikalische “Eigenschaften” deaktiviert. Einzig die Collision-Detection, welche im Fighting-Bereich zwischen Blöcken und Wänden benötigt wird, bleibt während dieser Zeit aktiviert. Ausserdem wurde die Bewegung des einzelnen Blocks auf die X- und Z-Achse limitiert, welche als Vektoren betrachtet die Inputfläche aufspannen. Die Blöcke können als nicht in den Tisch “hineinfallen”.

KAPITEL 12

Testdokumentation

> "Any fool can write code that a computer can understand. Good programmers write code that humans can understand."
>
> —Martin Fowler

12.1 Testing mit Unity3D

Für Unity3D wurde im Dezember 2013 ein Tool namens "Unity Test Tools" herausgegeben, welches Unit und Integration Test mit Unity3D vereinfacht ermöglicht [1].

Am 2. Juni 2013 kündigte Kasper Anderson, Teamleiter des Toolsmith Teams von Unity, das Runtime Test Framework an [2].

Nebst NUnit haben wir versucht, diese beiden Tools einzusetzen. Wie nachfolgend genauer erläutert, konnte das Testing auf Grund von zu hohem Aufwand gegenüber dem Nutzen für dieses Projekt so nicht durchgeführt werden.

12.1.1 Testframework NUnit

NUnit ist ein unit-testing Framwork für .Net Spreachen. Ursprünglich wurde es aus JUnit portiert, der aktuelle Release (v 2.6) ist der siebt-grösste Release von diesem auch xUnit basierendem unit testing Tool.

NUnit kann zum Testen von Code aus Behaviour-Scripts von Unity3D verwendet werden.

12.1.2 Unit Tests

Games sind grundsätzlich mittels Unit Tests schwieriger zu testen als andere Software, da es keine so klare Abgrenzungen innerhalb des Codes gibt.

In unserem Fall wird ein etwas anderer Entwicklungsansatz als bisher gewohnt verfolgt, da Skripte geschrieben werden, die lediglich das Verhalten eines Objektes beinhalten und dann an das entsprechende Game Object angehängt werden, was das Testen von einzelnen Scripts mittels Unit Testing nicht einfacher macht. Ein weiterer Punkt ist, dass viele Scripts/Klassen von MonoBehaviour erben (müssen) und MonoBehaviour selbst nicht vom Programmierer instanziert werden kann; dies macht nur die Unity3D Game Engine während der Laufzeit. Ausserdem müsste ein Mocking Objekt für MonoBehaviour erstellt werden, was zwar möglich ist, aber in unserem Fall im Verhältnis zum Nutzen, das es bringt, viel zu viel Aufwand darstellt.

[1] http://blogs.unity3d.com/2013/12/18/unity-test-tools-released/
[2] http://blogs.unity3d.com/2013/06/02/runtime-tests-unitys-runtime-api-test-framework/

12.1.3 Integration Tests

Integration Tests testen layerübergreifend das Verhalten und Zusammenspiel von verschiedenen Komponenten. Mit den Integration Tests, die das Tool "Unity Test Tools" zu Verfügung stellt, können Dinge wie zum Beispiel die Physik von Objekten überprüft werden. Beispielsweise ist es möglich zu testen, ob eine Kollision stattgefunden hat.

Solche Integration Tests sind wichtig für physikbasierte Systeme, wie es die meisten Games sind. In unserem Fall sind wir die bereitgestellte Physik der Game Engine aber umgangen, da sie für diesen Fall mehr hinderlich als nützlich gewesen sind. Unity3D trägt diesem Umstand auch Rechnung und erlaubt es jedem Programmierer selbst zu entscheiden ob er die Physik Engine ganz, nur teilweise oder gar nicht nutzen möchte. Die weiteren Ausführungen über die Gründe dieses Entscheids, finden sich in der Development Dokumentation unter dem Punkt *Eigene Physik*. Dieser hohe Mehraufwand im Verhältnis zum Ertrag war auch der Grund dass wir keine automatisierten Tests integriert haben.

In grösseren Systemen machen diese Tests absolut Sinn, in unserem Fall war die Zeit jedoch sehr begrenzt und wir gewichteten die Lauffähigkeit der Applikation nach Abschluss der Entwicklungsarbeiten höher als die automatische Testbarkeit. Ausserdem wäre für ene sinnvolle Umsetzung ein Build-Server notwendig gewesen, was zusätzlichen Aufwand bedeutet hätte.

12.2 Usability Tests

Usability Test wurden einerseits mittels Paperprototyping und anderseits mittels direkten User-Test am Multitouch-Tisch Pixelsense durchgeführt. Als Test-Benutzer wurden verschiedene Personen mit unterschiedlichem Hintergrund und Wissen bezüglich Informatik ausgewählt. Die Test-Benutzer wurden aufgefordert verschiedene Aufgaben wie zum Beispiel ein Spiel zu beginnen oder abzubrechen zu lösen. Anschliessend sollten die Test-Benutzer einfach mit dem System interagieren, also triDat spielen und uns Feedback zu Touchfeeling, Aussehen, Verständlichkeit und möglichen Problemen geben. Dieses Feedback wurde gleich im Team mit der Testperson besprochen und anschliessend ein Redesign entwickelt. Nach jedem Redesign wurden wieder Test-Personen gebeten das neue Design oder Feature zu testen.

12.2.1 Paperprototyping

Ausserdem wurden diverse Paperprototyping-Test durchgeführt, um abzuklären, ob die zukünftigen Benutzer das System überhaupt intuitiv bedienen können. Die verschiedenen Ergebnisse aus diesen Paperprototyp-Tests und die daraus resultierenden Redesigns, sind unter Domainanalyse unter GUI in *Paper-Prototyping* genauer beschrieben.

Es wurde festgestellt, dass die Paperprototype-Tests für Bereiche wie zum Beispiel Menu-Führung oder Abläufe in der Applikation sehr gut sind, da bereits bevor etwas implementiert wurde, festgestellt werden kann, wenn der Grossteil der Benutzer zum Beispiel eine Aktion oder ein Feature gar nicht versteht. Anderseits eignen sich Paperprototype-Test nicht um Fragen rund um die Themen Touch-Feeling, Flow, Spass oder auch Design zu klären. So war uns bis nach der Implementation zum Beispiel nicht bewusst, dass sich ein Swipeable-Menu für den Multitouch-Tisch Pixelsense nicht eignet, da dessen Touchfläche im Vergleich zum Rahmen leicht tiefer liegt.

12.2.2 Testing mittels implementierter Features

Testbenutzer wurden gebeten gestellte Aufgaben direkt in der Applikation am Multitouch-Tisch Pixelsense zu lösen. So konnten die Komponenten, die mittels Paperprototyping schwierig zu testen sind, wie zum Beispiel ein gutes Touchfeeling trotzdem getestet werden. Einige Probleme wurden auch nur am Multitouch-Tisch Pixelsense erkannt (Swipeable Menu), während andere bereits bei den Papierprototypen auftraten.

12.2.3 Usability Testlog

Tabelle 12.1: Testlog der Usability Tests Teil 1

Test	Was	Getestet auf	Resultat
1	erster Papierprototyp	wird Fighting-Bereich / Spielbereich Unterscheidung erkannt	ja, nur etwas schwer mit Papierprototyp zu testen
2	Prototyp 1: Block bewegen	wird erkannt, dass Rotate-Gesture und Pan-Gesture gleichzeitg ausgeführt werden können	ja, dieses Verhalten wurde intuitiv vorausgesetzt
3	Implementation: Wolke (Fighting-Bereich)	wird erkannt, wohin die Blöcke gezogen werden sollen	nein, Testbenutzer wunderten sich, dass die Blöcke nur an einer Stelle über die Wolke hinweg gezogen werden können
4	Implementation Quit-Button	wird erkannt, wie das Spiel beendet werden kann	ja, allerdings zufällig, Testbenutzer drückten den Button ohne das Spiel beenden zu wollen
5	Implementation Quit-Button mit Schriftzug	wird jetzt erkannt, wie das Spiel beendet werden kann	ja, allerdings drückten Testbenutzer den Button noch immer ohne das Spiel beenden zu wollen bevor sie den Schriftzug aktiv gelesen haben
6	Papierprototyp Swipeable Menu	wird erkannt, wo sich das Menu befindet	ja, allerdings konnten die Testbenutzer vermuten, dass ausserhalb der Touchoberfläche sich noch etwas befand, da das Menu abgedeckt wurde
7	Implementation Swipeable Menu	wird erkannt, wie das Menu zu bedienen ist	ja, allerdings fiel den Testbenutzern das “heraufholen” des Menus sehr schwer, da die Toucherkennung am Rand nicht sehr zuverlässig ist und das Rändchen störte
8	Implementation Eck-Menu	ist die Bedienung und Bedeutung der einzelnen Menupunke klar	ja, dieses Menu erhielt das beste Feedback
9	Papierprototyp Wurmloch	ist klar, was mit den Blöcken passiert wenn sie ins Wurmloch befördert werden	ja, die Testpersonen konnten voraussagen, dass dies ein Transportsystem zum Spielfeld ist
10	Implementation: Wurmloch	ist klar, was mit den Blöcken passiert wenn sie ins Wurmloch befördert werden	ja, die Testpersonen konnten voraussagen, dass dies ein Transportsystem zum Spielfeld ist
11	Papierprototyp Queue	wird erkannt, für was die Queue dient	ja, die Testpersonen erkannten die Queue als Vorschau für die nächsten Blöcke
12	Implementation: Queue	wird erkannt, für was die Queue dient	ja, die Testpersonen erkannten die Queue als Vorschau für die nächsten Blöcke
13	Implementation: Wurmlocheingangsportal	werden die zwei Stati “aktiv” und “inaktiv” anhand der Animation erkannt	ja, die meisten Testpersonen erkannten auf anhieb, dass die Blöcke nur bei aktivierter Animation ins Wurmloch befördert werden können, die anderen verstanden das Prinzipt sobald der Block trotz kurzen Bemühungen erst durchging als das Wurmlocheingangsportal wieder aktiviert wurde
14	Start Over (in Pixelsense Shell)	wird erkannt, wie ein Start Over ausgeführt werden kann in der Pixelsense Shell	nein, einige bemerkten gar nicht, dass das Logo ein interagierbares Element ist, anschliessend war nicht klar, was damit gemacht werden kann

Test 1

Hier wurde mittels des erste Papierprototyps (eine einfache Skizze von Spielbereich und Fighting-Bereich (Wolke)) getestet ob die Testbenutzer die Aufteilung und den Unterschied zwischen Spielfeld und Fighting-Bereich

erkennen. Dafür wurden einige ausgeschnittene Blöcke in die Wolke gelegt und die Testbenutzer aufgefordert sie an den unteren Spielfeldrand zu bringen. Alle Testbenutzer bewegten die Blöcke an den Rand der Wolke, wo die Testbetreuer die Blöcke auf dem Spielfeld platzierten und die Testbenutzer die Blöcke drehten und anschliessend nach unten zogen. Die Kriterien wurden zwar erfüllt, jedoch zeigte sich, dass ein Papierprototyp zum Testen solches Verhaltens nicht geeignet ist.

Test 2

Nachdem der erste Prototyp (bestehend aus einigen Blöcken, die frei im Raum bewegt werden konnten) implementiert war, wurde er auch für einen Usability-Test verwendet um zu überprüfen ob die Benutzer realisieren, dass sie den Block gleichzeitig ziehen und drehen können. Die meisten Testbenutzer bemerkten gar nicht, dass sie den Test bereits abgeschlossen hatten, da sie dieses Verhalten (bewegen und drehen gleichzeitig) intuitiv voraussetzen.

Test 3

Nach einer ersten Implementation des Fighting-Bereichs in Form einer Wolke sollte mittels Testbenutzern herausgefunden werden ob diese erkennen, wie sie den Block vom Fighting-Bereich auf das Spielfeld bekommen. Die meisten Testbenutzer versuchten den Block irgendwo über den Rand der Wolke zu ziehen, was aber aufgrund der Begrenzungen nicht möglich war. Dies wirkte irritierend. Der “Durchgang” zum Spielfeld wurde oft nur per Zufall gefunden.

Test 4

Um das Spiel zu beenden wurde ein Quit-Button in Form eines Buzzers implementiert. Es sollte mittels Testbenutzern herausgefunden werden, ob diese erkennen, wie das Spiel beendet werden konnte. Die meisten beendeten das Spiel auch korrekt, jedoch ohne es zu wollen, da sie sobald sie den Button sahen, diesen betätigten. Auf die Frage warum sie den Button betätigten, antworteten einige mit “ich weiss nicht...” und andere mit “Er sah aus als sollte ich ihn drücken”.

Test 5

Auch mit der Aufschrift “Quit” änderte sich nicht viel am Resultat von Test 4. Die Testpersonen antworteten auf die Frage ob sie die Aufschrift nicht lesen konnten, dass sie erst im Nachhinein bemerkten, was dort stand. Offensichtlich war das Design, das an einen Buzzer angelehnt war, unglücklich gewählt.

Test 6

Mit diesem Papierprototyp sollte evaluiert werden, ob die Testbenutzer erkannten, dass ein Menu “heraufgezogen” werden kann. Leider brachte dieser Papierprototyp keine relevanten Resultate, da die Testbenutzer aufgrund des abgedeckten Menus vermuteten, dass sich dort etwas befand, das bewegt werden kann.

Test 7

Nachdem eine erste Version des Swipeable Menu implementiert war, wurden Testbenutzer gebeten, das Spiel neu zu starten. Die meisten erkannten, dass am unteren Rand sich wohl was befindet, konnten das Menu aber nur schlecht oder gar nicht “heraufziehen”, da die Toucherkennung an den Rändern der Touchoberfläche nicht so gut ist wie mehr in der Mitte. Ausserdem empfanden einige Testbenutzer das Rändchen mit dem kleinen Absatz vom Übergang Rand - Touchfläche als störend.

Test 8

Als Alternative wurde ein neues Menu in der Ecke, das immer sichtbar bleibt implementiert. Die Testbenutzer konnten nun die gestellten Aufgaben wie Beenden und Neustarten des Spiels, Hintergrund-Theme-Wechsel, etc ohne Probleme glöst werden. Diese Version des Menus erhielt allgemein das beste Feedback.

Test 9

Den Testpersonen wurde ein Papierprototyp des Röhrensystems (Wurmlöcher) vorgelegt und gebeten, die Blöcke vom Fighting-Bereich in den Spielbereich zu bringen. Aufgrund des fehlen von Animationen war anfangs nicht allen klar, dass die Blöcke in den Röhreneingang geschoben werden müssen. Anschliessend konnten aber alle Benutzer voraussagen, dass die Blöcke zum Spielfeld transportiert werden und es sich hier offensichtlich um ein Transportsystem handelt. Da sich die Blöcke in einer Röhre befanden versuchte auch keiner der Testbenutzer, die Blöcke innerhalb des Wurmlochs zu manipulieren.

Test 10

Das selbe Ergebnis wie in Test 9 kam auch raus, nachdem das Transportsystem implementiert war und Testbenutzern zum erneuten Testen vorgelegt wurde.

Test 11

In einem weiteren Test mit dem Papierprototyp des Röhrensystems sollte überprüft werden, ob die Testpersonen erkannten, was die Queue darstellt und warum die Blöcke nicht weiter durch konnten. Die meisten Testbenutzer erkannten auf Anhieb, dass es sich um eine Vorschau der nächsten Blöcke handelt. Einige wunderten sich, warum der Block "stecken" bleibt, sobald jedoch sich ein zweiter hinter dem ersten Block einordnet, erkannten auch diese was der Zweck dieser Queue war.

Test 12

Das selbe Ergebnis wie in Test 11 kam auch raus, nachdem die Queue implementiert war und Testbenutzern zum erneuten Testen vorgelegt wurde.

Test 13

In diesem Test wurden die Testbenutzer erneut gebeten die Blöcke vom Fighting-Bereich via Wurmloch zum Spielfeld transportieren zu lassen. Fokus wurde dabei auf die Erkennung der verschiedenen Stati des Wurmlocheingangsportal gelegt. Die Testbenutzer wurden gebeten wie immer ihre Gedanken laut auszusprechen und einige fragten sich, warum denn der Block manchmal nicht ins Wurmloch bewegt werden kann. Allerdings fanden alle nach ein paar Sekunden mit einem "aha" heraus, dass sich die Animation am Wurmlocheingangsportal ändert. Es brauchte also nur eine sehr geringe Lernphase pro Testbenutzer.

Test 14

Für diesen Test wurde die Surface Shell auf dem Multitouch-Tisch Pixelsense gestartet und die Testbenutzer gebeten ein Start Over durchzuführen. Den meisten Benutzern war nicht klar, dass das Logo ein bedienbares Element ist. Auch wie das Start Over letzendlich funktioniert fanden nur wenige heraus.

12.3 Manuelle Tests

Grundsätzlich sind die meisten Skipts von einandern unabhängig. Darum gilt hier die Faustregel: Wenn an einem Skript etwas geändert wurde, verändert sich das Verhalten bei einem anderen Skript nicht!

Da die automatischen Tests weggefallen sind, wurde vermehrt auf manuelle Tests gesetzt. Im Allgemeinen ist folgendes Protokoll nach jeder Änderung in etwa dieser Abfolge durchgeführt worden, natürlich jeweils noch Abhängig vom Implementationsstand:

1. Wurde die gewünschte Änderung erreicht?
2. Funktioniert die Bewegung auf dem Grid/Fighting-Bereich noch?

3. Werden die Blöcke noch in der richtigen Anzahl und durch die richtigen Portale/Wurmlöcher geschleust?
4. Enstehen unerwünschte Artefakte/Nebenwirkungen?

Zudem wurden etwas umfangreichere manuelle Tests im folgenden Log niedergeschrieben.

Tabelle 12.2: Testlog der manuellen Test Teil 1

Wo	Was	Wann	Resultat	Getestet auf
Fighting Bereich	kann Block frei (rotate und pan gleichzeitig) bewegt werden	24.03.2014	freie Bewegung funktioniert	Entwicklermaschine
Fighting Bereich	Wird der Block mittels Kollision vom Durchdringen der Wand abgehalten	24.03.2014	Kollision wird erkannt, jedoch bleiben Blöcke zT in der Wand stecken	Entwicklermaschine
Fighting Bereich	werden die richtigen Blöcke mit Hilfe des BlockSpawner-Scripts korrekt erzeugt	30.03.2014	die richtigen Blöcke werden korrekt erzeugt	Entwicklermaschine
Fighting Bereich	wird die richtige Anzahl Blöcke erzeugt	30.03.2014	die Anzahl erzeugter Blöcke war korrekt	Entwicklermaschine
Fighting Bereich	sehen die Partikel wie eine reale Wolkenformation aus	01.04.2014	ja, die Partikel sahen sehr ähnlich aus	Entwicklermaschine
Transporter	wird der Block beim Übergang vom Fighting-Bereich zum Spielerfeld korrekt zerstört und wieder erstellt	04.04.2014	der Übergang verlief reibungslos	Entwicklermaschine
Spielerfeld	kann der Block auf dem Grid mittels Touchinput korrekt bewegt werden	07.04.2014	der Block konnte korrekt bewegt werden, die Rotation schlug allerdings fehl	Pixelsense
Spielerfeld	werden die Prefabs am korrekten Ort generiert	11.04.2014	nein, die Blöcke erschienen nicht innerhalb des Grids	Entwicklermaschine
Wurmloch	bewegt sich der Block korrekt innerhalb des Wurmlochs entlang des vordefinierten Pfads	12.04.2014	ja, der Block bewegt sich korrekt entlang des Pfads	Entwicklermaschine
Spielerfeld	werden die Prefabs am korrekten Ort generiert	13.04.2014	ja, die Blöcke wurden am korrekten Ort erstellt	Entwicklermaschine
Spielerfeld	werden die Blöcke korrekt als fixiert erkannt wenn sie auf den Spielerfeldboden oder einen bereits fixierten Block treffen	14.04.2014	ja, die Blöcke wurden als fixiert erkannt, jedoch im Array falsch platziert	Entwicklermaschine
Spielerfeld	wird eine komplette Linie korrekt erkannt und gelöscht	14.04.2014	nein, da die Fixierung nicht korrekt war, konnte die Linie nicht korrekt gelöscht werden	Entwicklermaschine
Fighting Bereich	eignet sich die Wolke als Fighting-Bereich	14.04.2014	nein, da die Blöcke in der Wolke verschwinden können und dann nicht mehr kontrollierbar sind	Entwicklermaschine

Tabelle 12.3: Testlog der manuellen Test Teil 2

Wo	Was	Wann	Resultat	Getestet auf
Spielerfeld	werden die Blöcke korrekt als fixiert erkannt wenn sie auf den Spielerfeldboden oder einen bereits fixierten Block treffen	17.04.2014	ja, die Blöcke werden als fixiert erkannt, jetzt auch mit korrekter Platzierung	Entwicklermaschine
Spielerfeld	wird eine komplette Linie korrekt erkannt und gelöscht	17.04.2014	ja, da nun die Fixierung klappt, konnte die Linie korrekt gelöscht werden	Entwicklermaschine
Spielerfeld	kann der Block auf dem Grid mittels Touchinput korrekt rotiert werden	17.04.2014	nein, da die Rotationsberechnung über Pivotelement falsche Koordinaten lieferte	Entwicklermaschine
Spielerfeld	kann der Block auf dem Grid mittels Touchinput korrekt rotiert werden	21.04.2014	der Block konnte korrekt rotiert werden	Pixelsense
Fighting Bereich	bleibt der Block innerhalb der Abgrenzung des Fighting-Bereichs	28.04.2014	Block bleibt in der Wand stecken	Pixelsense
Fighting Bereich	bleibt der Block innerhalb der Abgrenzung des Fighting-Bereichs	29.04.2014	Block bleibt nicht mehr in der Wand stecken	Entwicklermaschine
Fighting Bereich	kann der Block bei Wandberührung wieder von dieser weggezogen werden	29.04.2014	nein, der Block bleibt kleben	Entwicklermaschine
Fighting Bereich	kann der Block bei Wandberührung wieder von dieser weggezogen werden	29.04.2014	ja	Entwicklermaschine
Spielerfeld	kann mehr als ein Block gleichzeitig auf dem Spielerfeld kontrolliert werden	30.04.2014	ja	Entwicklermaschine
Blockpreview	werden die Blöcke in der korrekten Reihenfolge in der Preview (Queue) angezeigt	05.05.2014	ja	Entwicklermaschine
Wurmloch	bewegt sich der Block korrekt innerhalb des Wurmlochs entlang des vordefinierten Pfads	06.05.2014	nein, der Block blieb im Wurmloch stecken	Entwicklermaschine

12.4 Statische Code-Analyse

Da die Skripte in Unity3D nicht nach gängigen SE Richtlinien gegliedert sind, ist eine statische Code-Analyse nicht sinnvoll. Zudem sind übliche C#-Conventions wie beispielsweise Naming-Conventions mit Underscore für private Variablen selbst in Beispielen auf dem Unity3D Forum und in der Skript-Referenz nicht immer eingehalten.

KAPITEL 13

Installationsanleitung

> "In theory, theory and practice are the same. In practice, they're not."
>
> —Yoggi Berra

Die Installationsanleitung wurde ausschliesslich unter Windows 7 getestet, sollte jedoch auch auf Windows 8 Geräten funktionieren.

Grundsätzlich lässt sich das Spiel mit einer Ausführbaren Datei namens *triDat.exe* und dem entsprechenden Daten Ordner *triDat_data* mit Doppelklick von jedem am PC anschliessbaren Medium starten. Da keine Daten gespeichert werden, sollte es zumindest in Theorie auch von CD startbar sein (ungetestet).

Darum gibt es zwei Varianten der Installation von triDat:

- via entzippen vom *triDat.zip* Zip-File und anschliessendem Aufrufen von triDat.exe in dem entpackten Ordner.
- via dem triDatsetup.exe, welches die Dateien an den (durch den Benutzer änderbaren) Ort installiert, und einen Eintrag ins Windows Startmenu machen kann. Somit kann triDat gewohnt aus dem Startmenu aufgerufen werden.

Die ausführbare Datei "triDatSetup.exe" per Doppelklick ausführen und den Instruktionen am Bildschirm befolgen. Danach kann das Spiel über den gewohnten Mechanismus über das Startmenu gestartet werden.

KAPITEL 14

Development Dokumentation

> "Talk is cheap. Show me the code."
>
> —Linus Torvalds

14.1 Vorwort für den Programmierer

Diese Dokumentation ist gedacht für diejenigen, die dieses Programm erweitern und unterhalten möchten oder selber ein entsprechendes Spiel erstellen, und von dieser Vorarbeit profitieren möchten. Es erklärt den Aufbau, Zusammenhang und Zusammenspiel aus der Sicht des Programmierers für den Prgrammierer. Daher sind grundlegende Programmierkenntnisse (in C#) vorausgesetzt.

Unity3D hat eine extensive Dokumentation, welche hier nicht wiedergegeben werden soll. Für weitere Informationen diesbezüglich sei auf folgende nützliche Webseiten verwiesen:

- Offizielle Unity3D Dokumenation (http://unity3d.com/learn/documentation)
- Ofizielle Unity3D Tutorials (http://unity3d.com/learn/tutorials/modules)
- Unify Wiki (http://wiki.unity3d.com/index.php?title=Main_Page)
- Übersicht über den Ablauf der Skripte und GameLoop (http://docs.unity3d.com/Manual/ExecutionOrder.html)

Komplett soll auf eine kurze Einführung doch nicht verzichtet werden, da gewisse Grundkenntnisse bezüglich Unity3D nötig sind, um die folgenden Abschnitte zu verstehen. Wer Unity3D jedoch schon etwas kennt, kann den nächsten Abschnitt überspringen.

14.2 Übersicht

Die Game-Engine von Unity3D, die sogenannte UnityEngine, bietet mehrere Schnittstellen zur Interaktion zwischen Programmierer und Objekten, welche sich in der virtuellen Welt bewegen.

14.2.1 Objekte in Unity3D

Ein Spiel-Objekt, in Unity3D *GameObject* genannt, besteht aus einer Beschreibung der Form und allerlei Zusatzinformationen wie physikalische Eigenschaften, Verhalten bei Kollisionen, Farbe und vieles mehr. Diese Informationen können im Editor direkt an ein Objekt angehängt werden, oder aber via *Scripting*.

Die UnityEngine beinhaltet unter anderem eine Physik Simulation, welche den erstellten Objekte Verhaltensregeln vorgibt. Weil wir unter anderem dem Finger bei der Touch-Bewegung exakt folgen wollen, machen wir jedoch kaum gebrauch davon (für genauere Information hierzu siehe *Eigene Physik*).

14.2.2 Koordinatensystem in Unity3D

Unity3D verwendet ein kartesisches Koordniatensystem, welches seinen Ursprung bei *0,0,0* besitzt. Verschiebung und Scaling (zu deutsch *zentrische Streckung*) werden mit Hilfe von Vektoren gemacht, welche je eine x, y, und z Koordinate besitzen (im dreidimensionalen Raum). Bei der Platzierung beschreibt der Vektor jeweils eine Position im kartesischen Koordinatensystem, beim Scaling bezieht sich die Angabe in welche Richtung das Objekt um wieviel gestreckt wird.

Die Rotation wird mit einem sogenannten Quaternion beschrieben, einem Konzept aus der Mathematik, welches von Unity3D auch intern verwendet wird, um Rotationen darzustellen. Da dieses Konzept kompliziert ist, wird oft auf Hilfsmethoden ausgewichen, wie zum Beispiel ein Objekt um eine bestimmte Achse um einen bestimmten Winkel (in Grad) zu drehen, auch Euler-Winkel-Drehung genannt.

Lokales und globales Koordinatensystem in Unity3D

Neben dem globalen Koordinatensystem, welches sich auf den absoluten Nullpunkt im virtuellen Raum bezieht, sieht Unity3D vor, dass Objekte *relativ* zu einander bewegt, rotiert und gescalet werden können. Dies bedeutet auch, dass Objekte geschachtelt werden können, und dann sowohl Rotation, Bewegung und Scaling relativ zum Eltern-Objekt geschieht.

Viewport und globales Koordinatesystem in Unity3D

Der Input, ob von einem Maus-Cursor oder direkt als Touchpunkt, wird in einem 2D-Overlay-Koordinatensystem, dem Viewport, erfasst. Der Viewport ist durch ein separates Koordinatensystem definiert, welches den Ursprung in der unteren linken Ecke des Bildes hat, und bis 1,1 auf der oberen rechten Ecke geht. Zur Umrechnung gibt Unity3D eine Hilfsfunktion mit, welche einem das Umrechnen in virtuelle Welt Koordinaten erspart. Zudem wird eine dritte Grösse mitgeliefert, welche angibt, auf welcher Tiefe der (gedachte) Strahl des Punktes auf ein Objekt trifft.

Diese Technik für die Inputerfassung beinhaltet sowohl Maus und Touch wie auch andere Eingabemöglichkeiten, welche einen (bestimmten) Punkt auf der Bildfläche darstellen.

Bewegungen im Koordinatensystem in Unity3D

Unity3D sieht grundsätzlich zwei unterschiedliche Bewegungsarten vor:

1. von Punkt A nach Punkt B, definiert durch zwei Vektoren, durch eine absolute Verschiebung/Rotation, welche das Objekt direkt ohne Zwischenstopps von A nach B verschiebt/rotiert. Damit der Vorgang nicht ruckelt, kann ein Lerp angewendet werden (eine Interpolation zwischen den zwei angebenen Vektoren, das das Objekt phasenweise verschiebt/rotiert).
2. durch ein Kraft in eine Richtung mit einer bestimmten Stärke (Auslenkung), welche sowohl Verschiebung wie auch Rotation beschreiben kann. Hierbei wirkt die (definierbare) Trägheit und Reibungskraft auf das Objekt ein, so dass es sich auf verschiedenen Oberflächen anders verhält

Letzteres wird bei den meisten Spielen empfohlen, weil es eine bessere Physik-Simulation erlaubt, da Kräfte zusammengezählt werden können und der resultierende Kraftvektor verwendet werden kann. Zudem lässt sich dasselbe Skript mit zum Beispiel veränderter Gravitationsstärke wieder verwenden. In unserem Fall überwog jedoch der Nutzen, die Objekte genau platzieren zu können, da wir in diesem Fall ohne Physik gut auskommen.

Kollisionsbehandlung in Unity3D

Sobald zwei Objekte mit je einem angehängten Collider aufeinander treffen, und mindestens eines davon einen Rigidbody besitzt, wird eine Kollision erkannt, sofern diese beiden Objekte auf einem Layer liegen, welcher Kollisionserkennung aktiviert hat.

14.2.3 Scripting

In Unity3D kann man mittels Scripts, die man an ein Objekt anhängt, alle Eigenschaften beinflussen, wie zB Farbe, Geschwindigkeit, Position oder Texturen und diese auch instantiieren (neues Objekt erstellen aus einer Vorlage oder Kopie eines schon bestehenden Objekts).

14.3 Spezielle Lösungen in triDat

In triDat gibt es ein paar Skripte und "Eigenheiten", welche einer Erklärung bedürfen. Dies beinhaltet einerseits Element wie zB der eigenen Physik (*Eigene Physik*), welche fortlaufend aus den Bedürfnissen des Spiels heraus gewachsen sind, und anderseits den Eigenheiten, wie zB das "GameClock" Skript (*Game Clock*), welche aus Empfehlungen von "Best Practices" (*Best Practices Unity3D*) entstanden sind.

14.3.1 *Eigene* Physik

UnityEngine beinhaltet eine Game-Engine, welche effizient mit Objekten umgehen kann, Gravitation simuliert, verschiedene Kollisionen erkennt und die Objekte in Richtung, Drehung und Skalierung in der virtuellen Welt auf einem Koordinatensystem platziert (genaueres unter *Koordinatensystem in Unity3D*).

Bei einer Touch-Oberfläche muss das Objekt dem Finger ziemlich exakt folgen, da sonst das Gefühl entsteht, das Objekt rutsche unter dem Finger weg. Dies hat zur Folge, dass wir zum grössten Teil auf die Physik verzichteten, da diese mit Kräften arbeitet (Auslenkungsstärke). Wie sich bei den Voruntersuchungen gezeigt hatte, ist die Berechnung der exakten Fingerposition mit Kräften nicht gut umsetzbar. Die Lösung hierzu ist simpel: Wir reduzierten alle Kräfte auf null und setzen das Objekt immer auf die Position des erhaltenen Touch-Punktes - mit einer entsprechenden zeitlichen Lerp Bewegung etwas geglättet, damit es keine ruckartigen Bewegungen gibt und sich realitätsnaher anfühlt.

Zudem sind in triDat die Layers für die Kollsionserkennung sehr eingeschränkt, so dass nur diejenigen Kollisionen erkannt werden, welche mit einem aktiven Block im Fighting-Bereich und der Wand (Fighting-Bereich "Umzäunung") stattfinden. Sollte ein Block mit einer Wand kollidieren, wird dieser wieder auf das Spielfeld *zurückgeworfen*.

Im Bereich, in welchem ein Grid-Movement (*Grid-Movement*) benötigt wird (Spielfeld), kann die Kollisionsbehandlung jedoch nicht verwenden werden, da Wände ungeeignet sind, um die Blöcke auf einem Grid zu bewegen. Die Lösung ist in diesem Fall etwas komplizierter:

- Höhe und Breite in Anzahl Reihen und Spalten wird gespeichert
- Ein Array speichert die belegten (aktiven und bereits fixierten) Cube-Stellen
- Der Spielfeldrand wird als belegt betrachtet, jedoch nicht im Array gespeichert
- Bei einer gewünschten Bewegung wird erst überprüft ob diese möglich ist, und nur ausgeführt falls diese Prüfung positiv endet
- Der Block wird automatisch nach einem Tick der GameClock um eine Einheit auf dem Grid nach unten bewegt. Trifft der Block auf einen fixierten Block oder den Spielfeldboden, so wird der Block ebenfalls fixiert.

14.3.2 Game Clock

Zeitgebung, also die virtuelle Uhr im Spiel, ist in Unity3D mittels einer *Time* Klasse gelöst, welche unter anderem die verstrichene Zeit seit dem Laden eines Levels angibt und die Zeit seit dem Start der Applikation. Um die Zeit besser steuern zu können und eigene Zeitfenster und Hilfsfunktionen zu definieren, macht es Sinn, diese Funktionen in einer eigenen Klasse zu kapseln, und dann konsequent überall diese Klasse anstelle der Time Klasse von Unity3D selbst zu verwenden.

Im Namespace *TriDat* gibt es ein *Singleton* namens GameClock, welches für jede Aktion, die zeitabhängig ist verwendet werden soll. *Time*, wie es von Unity3D zur Verfügung gestellt wird, sollte nie direkt verwendet werden,

da zB spätere Erweiterungen, wie Beschleunigung der gesamten Zeit (schnellerer Spielablauf), Pausieren und Game-Over von dieser Klasse abhängig sind.

14.3.3 Game Object Life Cycle für triDat

Überblick über den Spielablauf in triDat:

1. Ein neuer Block wird erstellt
2. Dieser Block bewegt sich entlang der Spline (*Spline-Movement (Hermite)*) innerhalb eines Wurmlochs zur Fighting-Area und wird dort in die Fighting-Area geworfen
3. Ist ein Wurmlocheingangsportal zum Spielerfeld offen und hat der Benutzer den Block zum entsprechenden Wurmlocheingangsportal gezogen:
   - wird der Block in das Wurmloch (Röhrensystem) gezogen und verliert dabei die Fähigkeit, auf Touchinput zu reagieren (*Free Movement*)
   - landet er am Ende des Wurmlochs auf dem Spielfeld, sofern es an seinem vorgesehenen "Landeplatz" Platz hat (*Grid-Movement*)
   - falls der "Landeplatz" besetzt ist, hält der Block auf einem Vorschaufeld, bis der Platz frei ist
4. Landet auf dem Spielerfeld, wo er wieder Touch-Eigenschaften erhält und nur noch innerhalb der Einschränkungen des Grids bewegt werden kann (*Grid-Movement*).
5. Der Block wird als "fixiert" bezeichnet, sobald er sich nicht weiter nach unten bewegen kann (*Grid-Movement*) (da er auf einen fixieren Block oder den Spielfeldboden traf).
   - sobald eine oder mehrere komplette Linien (bestehend aus fixierten Blöcken/Cubes) existieren, werden diese gelöscht und der Punktestand aktualisiert (*Grid-Movement*)
6. Sobald keine Blöcke mehr auf dem Spielfeld landen können, führt dies zu einem Gameover (*Grid-Movement*)

Vergleiche auch mit *Komponenten Digramm*

14.4 Wichtige Code Erläuterungen

14.4.1 Spline-Movement (Hermite)

> "Bei der Spline-Interpolation versucht man, gegebene Stützstellen, auch Knoten genannt, mit Hilfe stückweise stetiger Polynome, genauer Splines, zu interpolieren."
>
> —Wikipedia [50]

Und genau das ist hier das Ziel: Auf einer "Kurvenbahn", welche durch fixe Punkte gegeben ist, einen Block fliegen/folgen zu lassen. Dass er sich dabei noch dreht, ist nur ein erwünschter Nebeneffekt, sofern man die Punkte nicht alle gleich ausrichtet.

Dank diesem Konstrukt ist es möglich, zu definieren, zu welchem Zeitpunkt der Block an welche Stück der Spline sein muss, wenn die Animationsdauer gegeben ist.

Der meiste Code dafür wurde aus dem Unity3D-Wiki entlehnt: http://wiki.unity3d.com/index.php/Hermite_Spline_Controller

Der mathematische Teil befindet sich im Script *MathUtils.cs*, um dieser Linie zu folgen muss das Script *Spline-Controller.cs* angehängt werden.

14.4.2 Free Movement

Dieses Verhalten wird an den Block angeheftet, sobald die Endposition der Spline-Interpolation erreicht wurde und führt dazu, dass die Blöcke innerhalb der Fighting-Area frei, selbst durch einander hindurch, bewegt werden können.

Ausserdem werden dem Block in der Fighting-Area noch zwei weitere Eigenschaften gegeben.

Die eine ist die Ausheblung der Kollisionsbehandlung durch Unity3D, wobei die (durch die eingbaute Physik-Engine automatisch gesetzten) Kräfte auf den Block auf null gesetzt werden (*StopBlock()*).

```
void OnCollisionExit(Collision collisionInfo) {
    Repell(collisionInfo);
}
void OnCollisionStay(Collision collisionInfo) {
    Repell(collisionInfo);
}
void Repell(Collision collision) {
    isRunning = true;
    StopBlock ();
}
void StopBlock()
{
    if (!myRigidbody.isKinematic && isRunning) {
        myRigidbody.angularVelocity = Vector3.zero;
        myRigidbody.velocity = Vector3.zero;
        isRunning = false;
    }
}
```

Die andere ist die eigene Kollisionsbehandlung, wenn der Block mit der Wand zusammenstösst. Entgegen der Erwartung, kommt hier jedoch explizit keine Kollisionserkennung im Sinne der Unity3D GameEngine [1] zum Zug, da sich der Block sonst immer wieder in der Wand verheddert und falls er genügend beschleunigt wurde sogar komplett durch die Wand durch fahren würde, darum wird auf einen "Trick" zurückgegriffen:

```
private void ReturnToPlaceBeforeCollison()
{
    Vector3 movementThisStep = myTransform.position - previousPosition;
    float movementSqrMagnitude = movementThisStep.sqrMagnitude;
    float movementMagnitude = Mathf.Sqrt(movementSqrMagnitude);
    RaycastHit hitInfo;

    if (Physics.Raycast(previousPosition, movementThisStep, out hitInfo,
        movementMagnitude, layerMask.value))
    {
        myTransform.position = hitInfo.point;
    }
    previousPosition = myTransform.position;
}
```

Diese Funktion wird während jedem Update aufgerufen, und vergleicht die momentane mit der vorhergehenden Position des Objekts. Falls sich innnerhalb dieser Distanz ein Objekt befunden hat, wird der Block zurück auf seine alte Position gesetzt. Der Test geschieht mittels einem auf einen bestimmten Layer beschränkten Raycast.

14.4.3 Grid-Movement

Sobald der Block aus dem Wurmloch auf das Spielfeld gelangt, wird das Skript *MoveOnGrid* angehängt, welches die Benutzer Eingaben (Touchinputs) an den *PlayerFieldBlockCoordinator* weiterleitet, welcher wiederum prüft, ob die gewünschte Aktion erlaubt ist.

[1] Der einzige Ort, wo die Kollisionserkennung tatsächlich gebraucht wird, ist das Portal, von wo aus der Block ins Wurmloch eingesaugt wird. Da dieses Verhalten nicht geschwindigkeitsabhängig ist, kann hier mit Kollisionsdetektierung gearbeitet werden. Hierauf gehen wir hier jedoch nicht genauer ein, da dies ein Standard-Konzept von Unity3D ist.

Im Skript *PlayerFieldBlockCoordinator* wird keine Kollisionsbehandlung verwendet, sondern der Ansatz mittels in einem Array gespeicherten Objekt-Referenzen und einem Zustands-Array, der die Position mittels Array-Index und Art des Blocks speichert, verfolgt. Bei einer entsprechenden Bewegung, unanbhängig ob vom Benutzer oder von der Zeitgebung (der nächste Tick) initiiert, wird zuerst das Array aktualisiert und anschliessend der Block optisch um eine Einheit "verschoben".

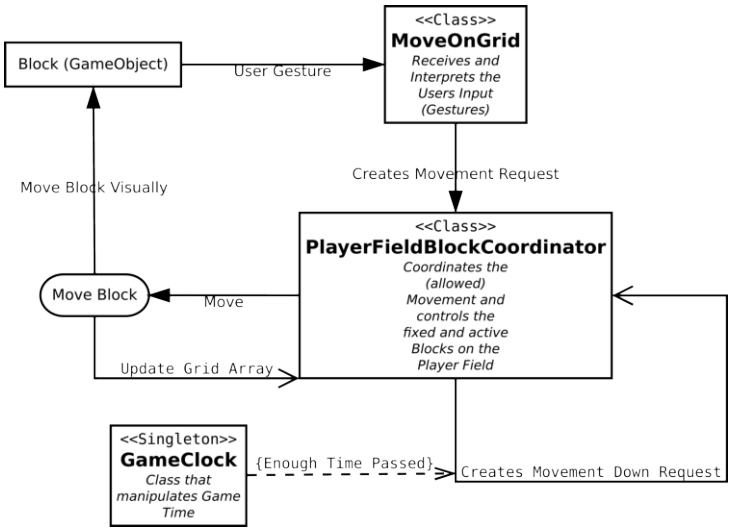


Abbildung 14.1: Grid-Movement: Übersicht [51]

MoveOnGrid gibt dem Skript *PlayerFieldBlockCoordinator* die Benutzerinteraktion (Intent) weiter, welches anschliessend prüft, wie sich der Block Verhalten soll und ob diese Bewegung durchführbar ist.

14.5 FEP (Frequently Encountered Problems)

14.5.1 Unity Errors/Messages

Message: *No camera layers. Adding one for the main camera.*

Diese Fehlermeldung stammt von TouchScript. Dieses braucht mindestend einen Layer um korrekt zu funktionieren. Die Meldung ist keine Fehlermeldung, sondern nur ein Log Eintrag um den Programmierer auf diese Tatsache aufmerksam zu machen,und somit unbedenklich. Selber einen Layer für die Kamera hinzuzufügen führt aber zu keiner Änderung.

Message: *MissingReferenceException: ...*

> MissingReferenceException: The object of type *put-the-name-here* has been destroyed but you are still trying to access it. Your script should either check if it is null or you should not destroy the object.

Wenn dies nicht offensichtlich ist, woher es stammt: wurden alle Delegates/Callbacks auch wieder *deregistriert*? Am einfachsten in der *OnDestroy()* Methode.

Message: *Some objects were not cleaned up when closing the scene*

Ähnlich wie die "MissingReferenceException", wenn also keine Objekte direkt in der *OnDestroy* Methode erstellt, so kann eine Ursache sein, dass nicht alle Delegates/Callbacks auch wieder *deregistriert* wurden.

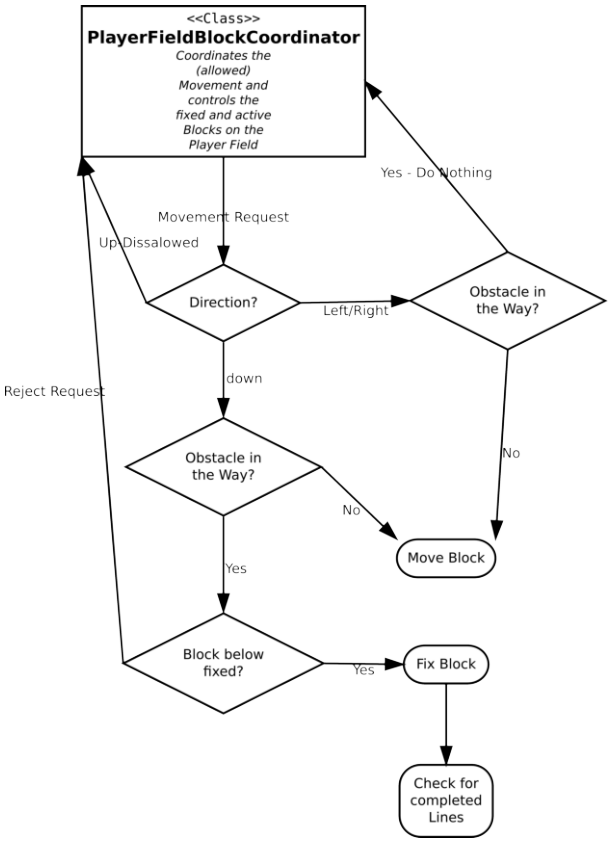


Abbildung 14.2: Grid-Movement: Abklärungskette [52]
Die Abklärungskette im *PlayerFieldBlockCoordinator* prüft, welche Aktionen ausgeführt werden dürfen.

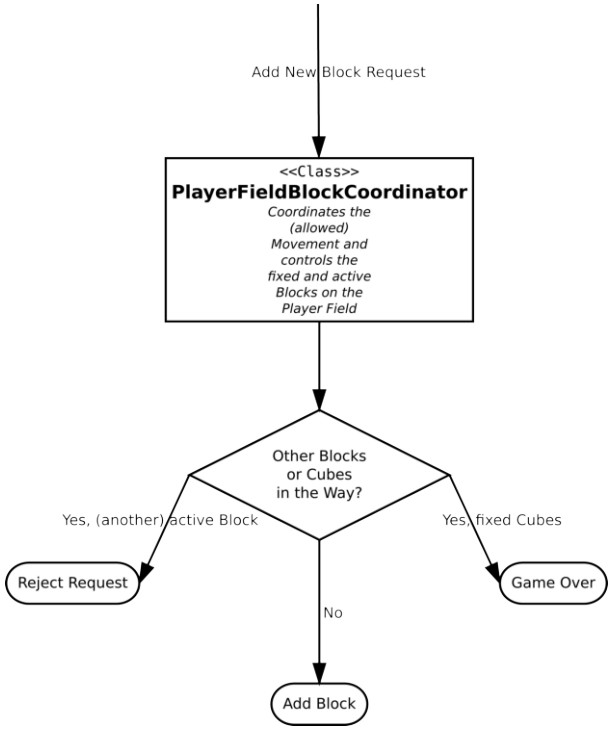


Abbildung 14.3: Grid-Movement: Kann ein neuer Block hinzugefügt werden? [53]
PlayerFieldBlockCoordinator prüft, ob der nachzuschiebende Block bereits Platz zum landen hat. Hat es keinen Platz, und die verhindernden Cubes sind als fixiert markiert, ist das Spiel vorbei. Ansonsten wird bei Platzmangel gewartet, bis Platz ist. Sobald Platz vorhanden ist, wird der Block dem Spielerfeld angefügt.

14.5.2 Unity3D Editor-Absturz

Hin und wieder stürzt der Unity3D Editor ab. Die Ursache hierführ ist unbekannt. Die Lösung besteht in einem Neustart des Editors, der danach meistens wieder gut funktioniert. Wenn dies nichts nützt, sollte überprüft werden, ob das Assembly (der Code) keine Fehler enthält.

Wenn diese Fehlersuche zu keinem Ergebnis führt, kann das Projekt in einem leeren Ordner neu ausgecheckt werden.

14.6 Bestehende Applikation Erweitern

14.6.1 Einrichten der Umgebung und erste Schritte

- Source Code vom Repository herunterladen (https://bitbucket.org/hixi/tetrisaeder.git)
- Download und Installation von Blender (Voraussetzung!) (http://www.blender.org/download/)
- Download und Installation von Unity3D (https://unity3d.com/unity/download)
- Unity3D starten
- heruntergeladesnes Projekt in Unity3D öffnen

Der erste Start kann eine Weile dauern, da alle Skripts kompiliert und Ordner indexiert werden müssen.

Falls eine Fehlermeldung bezüglich fehlender Assets angezeigt werden, so kann dem folgendermassen entgegengewirkt werden:

- Notwendige Assets installieren (menu -> Assets -> Import Package -> Custom Package...)
  - zum Source Folder “NeededUnityPackages” navigieren und die dort enthaltenen Assets installieren
- Ordner “ExampleTests” aus dem Unity3D Test Folder löschen, oder erst gar nicht importieren

14.6.2 Komponenten Digramm

14.6.3 Tipps und Tricks mit triDat

Skybox splitting

Sollte Skybox splitting (geteilter Hintergrund) mal nicht mehr erwünscht sein, so gibt es bereits eine vorbereitete Alternative mit nur einer Skybox, die ein ganzes (nicht gesplittetes Bild) über den Hintergrund laufen lässt. Um dies zu nutzen muss in jeder Szene die SkyboxSingle aktiviert und die SkyboxSplit deaktiviert werden.

Clone Objekte

Zur Laufzeit werden einige Objekte erstellt, welche dynamisch von Skripts erzeugt wurden. Diese werden zur besseren Identifizerung mit anghängten “clone” im Namen bezeichnet, was es erleichtert, die Objekte wieder aufzufinden.

In einem Skript können nur Objekte erstellt (instantiiert) werden, welche entweder eine Kopie eines bereits bestehenden Objektes sind, oder aber als Prefab existieren, welches im Ordner “Resources” unter Assets abgelegt ist.

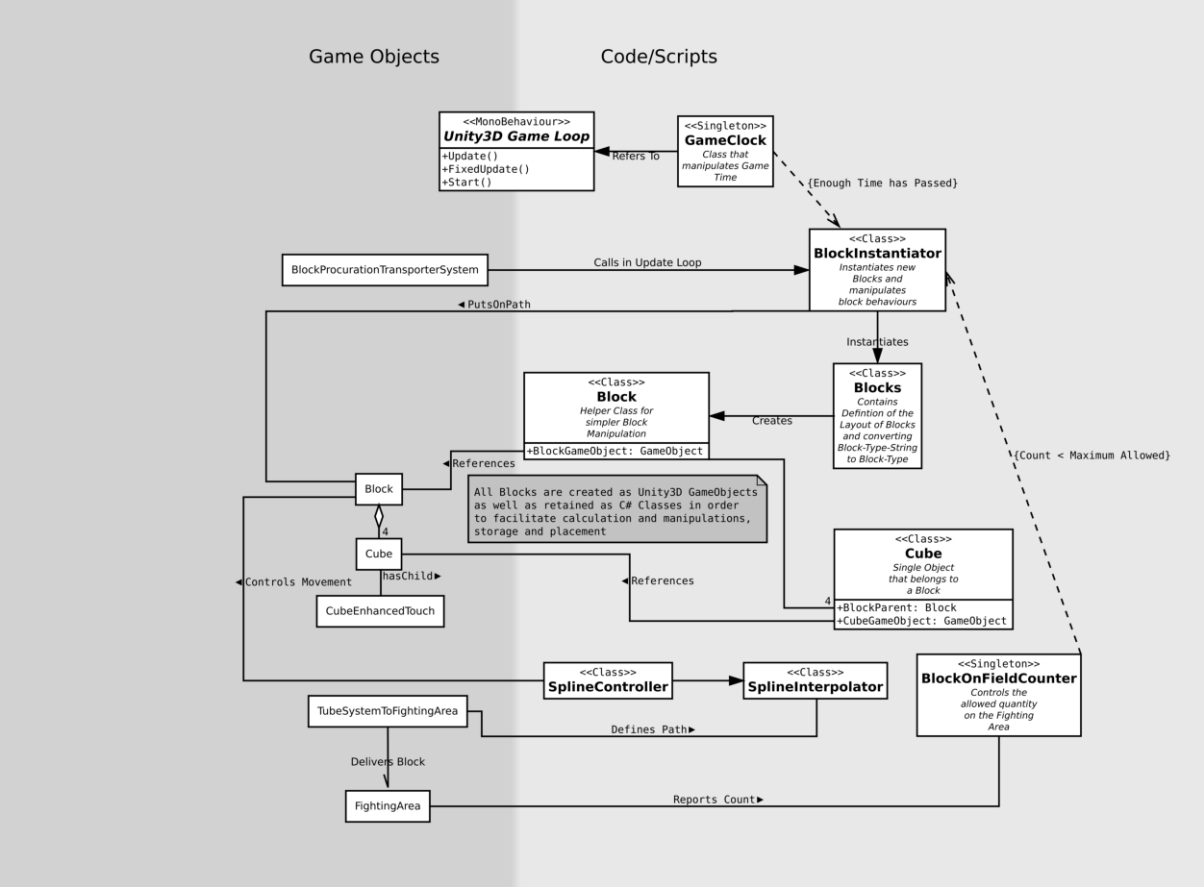


Abbildung 14.4: Komponenten Diagramm (folgt keinem UML Standard) [54]
Das Diagramm beschreibt den groben Zusammenhang, Interaktion und Ablauf eines Spielzyklus bis zur Ankunft des Blocks im Fighting-Bereich.

Make it a Prefab!

Aus allem (was geht) sollte ein Prefab erstellt werden! Dies erhöht die Laufgeschwindigkeit des Spiels, da die Objekte nur noch instantiiert werden müssen und falls ein Objekt, das zum Beispiel in verschiedenen Szenen verwendet wird zB eine neue Textur erhalten soll, so kann dies an einem einzigen Ort (dem Prefab) angepasst werden und alle Objekte, die aus dem Prefab erstellt wurden, übernehmen die Änderungen selbständig.

14.7 Nächste Schritte

Anschliessend findet sich eine Auflistung, was aus jetziger Entwicklersicht als nächstes gemacht werden sollte und wo noch Verbesserungen durchgeführt werden können.

- Entsprechenden Sound und Soundeffekte einbinden
- Logo in Splash-Screen und als Icon hinzufügen (in Unity -> Build Settings -> Player Settings)
- Zwei Konfigurationen für Development Build und Release Build
  - wobei zB beim Development Build der Touch Input mit einer Maus simuliert werden kann und beim Release Build [2] keine Maus Unterstützung vorhanden ist.

[2] Wenn das Spiel auf dem Tisch mit Maus Support aktiviert gespielt wird, so scheint die Mausemulation vom Windows 7 dazu zu führen, dass es die Touch-Inputs von Unity blockiert.

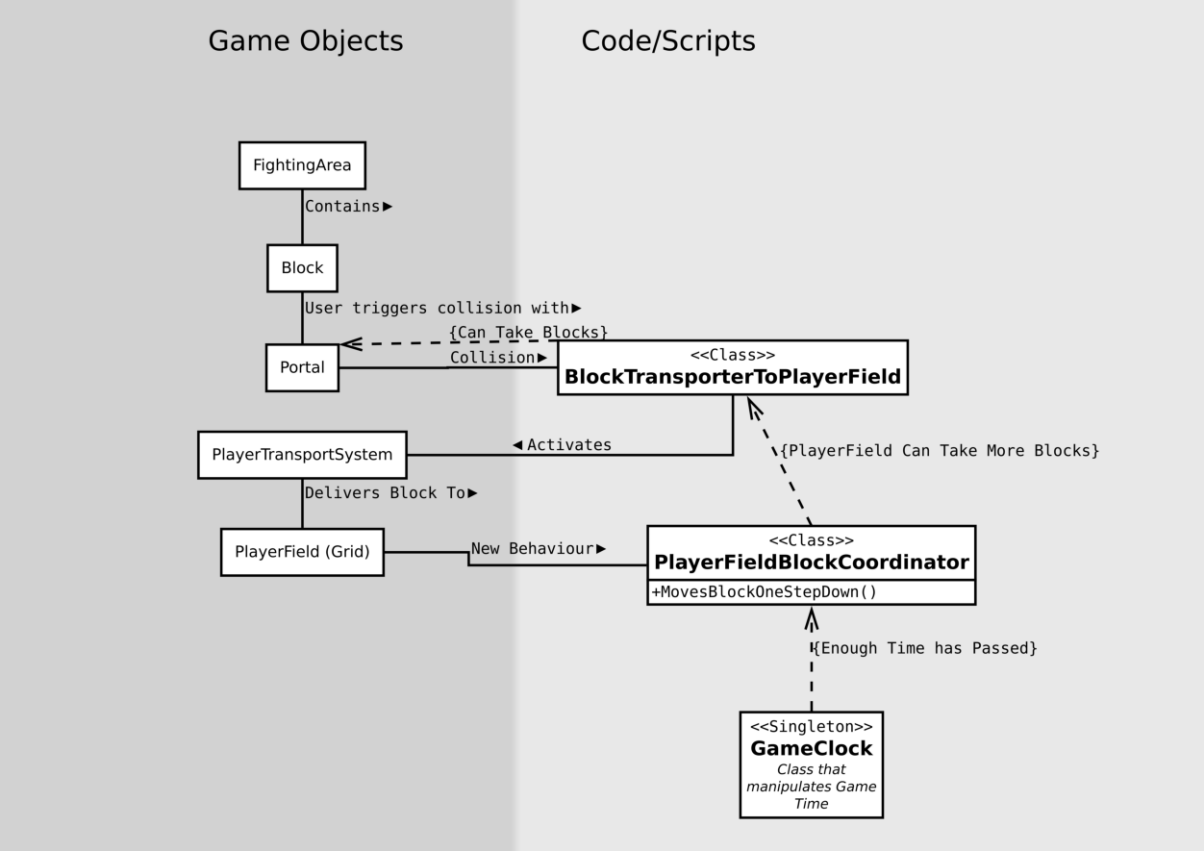


Abbildung 14.5: Komponenten Diagramm ab Fighting-Bereich (folgt keinem UML Standard) [55]

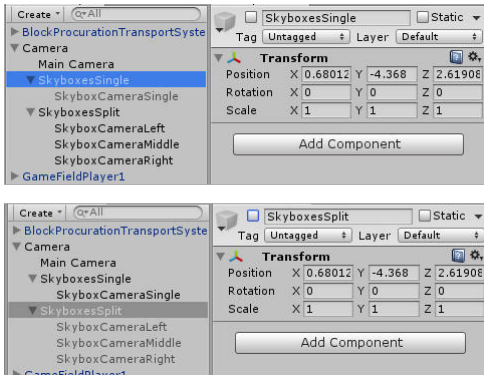


Abbildung 14.6: Skybox Splitting verwerfen [56]

14.8 Best Practices Unity3D

14.8.1 Structure

Die Organisation der Scenen, Projektordnern und Scriptordnern sollen ähnlich strukturiert sein, so dass man sich einfacher darin zurecht finden kann.

14.8.2 Gebrauch von Offsets für GUI Komponenten minimieren

Offsets sollten nur im Parent-Object einer Layout-Komponente gesetzt werden und so nicht von der Position ihrer Grand-Parents abhängen. Offsets sollten einander nicht ausschliessen. Grundsätzlich sollte folgendes nicht vorkommen:

> Parent Objekt hat Position (100, -50) Child Object sollte auf Position (10,10) zu liegen kommen Child Objekt wird relativ zum Parent Object auf Position (90,60) gesetzt

14.8.3 World Floor Position auf y = 0 setzen

Wenn der World Floor in der Y-Achse auf 0 positioniert ist, so ist es viel einfacher Objekte auf demselbigen zu positionieren und anschliessend die ganze World als 2D Space zu behandeln um sie zu konstruieren.

14.8.4 Spiel von allen Scenes aus startbar halten

Dies reduziert den Aufwand fürs Testen sehr, da jede Scene einzeln getestet werden kann. Um eine Scene generell runnable zu halten gelten folgende zwei Punkte:

- alle Daten, die von vorhergehenden Scenes benötigt werden müssen mockbar sein falls sie nicht vorhanden sind
- Objekte die von einer früheren Scene in die aktuelle mitgenommen werden, müssen wieder gespawned werden

```
myObject = FindMyObjectInSCene();

if (myObject == null)
{
    myObject = SpawnObject();
}
```

14.8.5 Scaling default immer 1

Alle Textures, Models etc sollten immer so designed sein, dass sie mit einem Scalefaktor von 1 importiert werden können. Ein GamoObject von Unity3D, zum Beispiel ein Cube kann als Referenz für das Scaling verwendet werden. Die World sollte auch durchgehend in Unity3D Einheiten gehalten sein und nicht in Pixel oder Centimeter.

14.8.6 Prefabs verwenden

Grundsätzlich sollen alle Objekte in einer Scene Prefabs sein, auch Objekte die nur einmal vorkommen. So wird es einfacher Änderungen an einzelnen Komponenten durchzuführen ohne gerade die ganze Scene verändern zu müssen.

14.8.7 Prefabs spezialisieren, nicht die Instanzen davon

Gibt es ein Prefab eines Objektes (zum Beispiel Cube), und es werden Spezialisierungen dieses Objektes gebraucht (zum Beispiel ein gelber Cube und ein blauer Cube), so sollen zwei neue Prefabs dafür erstellt und spezialisiert werden und nicht die Instanzen (des Cubes hier).

14.8.8 GetSaveComponent statt GetComponent verwenden

```
public static T GetSafeComponent<T>(this GameObject obj)
where T: MonoBehaviour
{
    T component = obj.GetComponent<T>();

    if(component == null)
    {
        Debug.LogError("Expected to find component of type "
            + typeof(T) + " but found none", obj);
    }

    return component;
}
```

14.8.9 Eigene Clock implentieren

Es ist von Vorteil eine eigene GameClock zu implementieren um Pausieren, Restart und auch Geschwindigkeitsanpassungen des Spiels einfacher zu gestalten.

14.8.10 Singletons

```
public class Singleton<T> : MonoBehaviour where T : MonoBehaviour
{
    protected static T instance;

    // Returns the instance of this singleton

    public static T Instance
    {
        get
        {
            if(instance = null)
            {
                instance = (T) FindObjectOfType(typeof(T));
                if(instance == null)
                {
                    Debug.LogError("An instance of " + typeof(T) + " is
                    needed in the scene, but there is none.");
                }
            }
            return instance;
        }
    }
}
```

- Vermeide Singletons für einmalige Instanzen von Prefabs, die nicht Manager sind
- Definiere static Properties und Methoden für public Variablen und Methoden, die häufig von ausserhalb der Klasse genutzt werden

```
GameManager.MyObject
```

statt

```
GameManager.Instance.MyObject
```

14.8.11 Public Variablen von Komponenten

Variablen einer Komponente, die aus dem Inspector heraus nicht verändert werden sollen, nicht public setzen, denn ansonsten wird sie ein Designer *bestimmt* verändern.

KAPITEL 15

Erfahrungsberichte

> "One of my most productive days was throwing away 1000 lines of code."
>
> —Ken Thompson

15.1 Erfahrungsbericht Linda Urech

Mich persönlich reizte an diesem Projekt vor allem die Tatsache, dass es sich um eine Gameentwicklung handelte. Im SE2-Projekt entwickelten wir bereits einen Bomberman-Klon, allerdings in Java und 2D, was sehr viel Spass machte. Daher freute ich mich auf die Herausforderung an einem grösseres Projek beteiligt sein zu dürfen und dabei mit einer mir neuen Technologie ein 3D Game zu entwickeln. Ich glaube auch nicht, dass ich so schnell wieder die Gelegenheit erhalte für eine so spezielle wie auch eindrückliche Hardware wie der Multitouch-Tisch Pixelsense zu entwickeln.

Die Aufgabenstellung liess uns viele Freiheiten bezüglich der Gestaltung der Benutzeroberfläche und der Auswahl zusätzlicher Features. Ideen hatten wir viele, nur mussten diese auf ihre Umsetzbarkeit und ihren Nutzen überprüft und entsprechend priorisiert werden. Dafür kamen Papierprototypen, Usability Tests am Multitouch-Tisch selbst und diverse andere Methoden zum Zug.

Ich fand toll, dass ich die im Modul UserInterfaces 2 erlernten Praktiken betreffend Usability Testing in diesem Projekt anwenden konnte. Dabei erkannte ich einerseits die Vorteile, die ich auch bereits im Unterricht mitbekam, anderseits - was für mich persönlich fast wertvoller war- auch die Limiten zum Beispiel von Papierprototypen. Auch erlebte ich, dass in der Softwareentwicklung (wie in fast allen Bereichen) die Theorie und Praxis deutlich divergieren. Softwareentwicklung nach SE oder nach Scrum lassen sich oft nicht kompromisslos nach einem vorgefertigen Muster durchführen, sondern müssen immer etwas den Gegebenheiten angepasst werden.

Überrascht haben mich persönlich der hohe administrative Aufwand und die anfänglichen Kommunikationschwierigkeiten bis sich das Team eingespielt hatte, obwohl ich bereits oft davor "gewarnt" wurde, dass dies auftreten würde. Auch in einem Zweierteam sind viele Absprachen und Abmachungen nötig um eine reibungslose Zusammenarbeit zu gewährleisten. Die hieraus gewonnenen Erkenntnisse werde ich in der folgenden Bachelorarbeit gut umsetzen können. Auch bezüglich Dokumentation eines Projektes und der nötigen Toleranz Änderungen gegenüber habe ich wichtige Erfahrungen gesammelt.

15.2 Erfahrungsbericht Nicola Jordan

Was mir am meisten gefallen hat an diesem Projekt, ist die Zusammenarbeit im Team. Obschon wir ziemlich diametrale Arbeitsweise und Umsetzungsstrategien angewendet haben, konnte mit viel Kommunikation jede Hürde gemeistert werden. Für mich war es ein persönliches Highlight, diese Gruppendynamik zu erleben und zu sehen, dass eine Zusammenarbeit nicht nur möglich ist, sondern auch gut funktioniert und Spass machen kann.

An Herausforderungen möchte ich vorallem die geänderte Herangehensweise an das Testing erwähnen: An der Schule wird gelehrt, dass "ungetesteter Code schlechter Code" sei, was üblicherweise mit Unittests und ähnlichen automatisierbaren Tests gemacht wird und nach Möglichkeit noch mit Continuous Integration ergänzt wird. Als

klar wurde, dass dies im Verhältnis zum Nutzen zuviel Zeit in Anspruch nehmen würde, und sich dieser Mehraufwand nicht rechtfertigen liess, wurde auf einen strukturierte manuellen Tests ausgewichen, so dass der Code und Funktionalität nicht komplett ungestet blieben und mit der Zeit entwickelte sich sogar ein gewisses Vertrauen in diese Methode.

Dass der Tisch in einer heissen Phase für zwei Wochen ausgefallen ist, hatte einen Anstieg in der kreativen Lösungsfindung zur Folge, und erhöhte Zeitweise die Unsicherheit ob das Projekt wie erhofft abgeschlossen werden kann. Darum war es eine gute Erfahrung, zu sehen, dass mit einem etwas erhöhtem Arbeitseinsatz dieser kritische Punkt abgefangen werden konnte.

Die offene Aufgabenstellung und das Neuland der Game-Programmierung waren für mich eine willkommene Abwechslung zu den stark strukturierten Übungsaufgaben an der HSR; neue Ansätze für Probleme und Flexibilität gegenüber Limitis bei der verwendeten Software und Hardware sowie Anpassungenfähigkeit der verwendeten SE-Methodik waren eine wertvolle Erfahrung für mich.

Anhang

16.1 Glossar

Tabelle 16.1: Glossar

Begriff	Erklärung
Blender	Eine Software, welche Modelle in 3D Erstellen kann
Block	Spielelement, besteht aus 4 Cubes, kann gedreht und gezogen werden
Block-Typ	L-Block, LR-Block, I-Block, S-Block, SR-Block, Q-Block, ...
Cube	Basiselement, aus welchem Blöcke bestehen
Fighting-Bereich	Bereich in welchem sich Blöcke frei bewegen und drehen lassen, hier wird um gute Blöcke gekämpft
Grid	Raster, in welchem sich die Blöcke auf dem Spielfeld bewegen
Line	eine Reihe im Spielfeld, die komplett mit Cubes gefüllt ist
lowlevel	tiefes Abstraktionslevel, Grundlegende Funktionen müssen selber programmiert werden
Multiplayer	Multiplayer auf einem Gerät
Multitouch Tisch Pixelsense	primäre Hardware, auf der die Appliktion laufen soll
Netzwerk-Multiplayer	Multiplayer auf mehrere Geräte verteilt, die via Netzwerk miteinander kommunizieren
Portal	Kontextabhängig zwei Bedeutungen: Teleporter-Transporter beim Wolken-Design des Fighting-Bereichs oder Kurzform von Wurmlocheingangsportal
Queue	die Queue dient als Vorschau für die nächsten Blöcke und befindet sich im letzten Drittel des Wurmlochs
Script	in einem Script wird das Verhalten des zugehörigen Objekts in Form von Code beschrieben
Spawn-Bereich	Bereich im Fighting-Bereich, in welchem die Blöcke landen
Spieler	Benutzer der Applikation
Spielfeld	Bereich, in welchem der Benutzer die Blöcke innerhalb des Grids drehen und ziehen und in Linien anordnen kann; auch als Spielerfeld bekannt
Team	eine Gruppe von Spielern, die dasselbe Ziel verfolgt
triDat	Tetris-ähnliches Spiel, das entwickelt wird
Unity	Unity 3D (kurz Unity) ist ein 3D-Entwicklerwerkzeug für Spiele
Viewport	Sichtbarer Teil der virtuellen Welt, welcher von der Kamera definiert wird.
Vorschau	zeigt den nächsten Block, der im Spielfeld erscheinen wird
Wurmloch	Transportsystem für Blöcke vom Fighting-Bereich zum Spielfeld in Form von Röhren
Wurmlocheingangsporta	Eingang zum Wurmloch, hat zwei Stati: aktiv und inaktiv

[1] Nicola Jordan and Linda Urech. Soll/ Ist Stunden. PNG Image, 2014.

[2] Nicola Jordan and Linda Urech. Stundenaufteilung nach Kategorie. PNG Image, 2014.

[3] Nicola Jordan and Linda Urech. Stundenaufteilung nach Kategorie von Nicola Jordan. PNG Image, 2014.

[4] Nicola Jordan and Linda Urech. Stundenaufteilung nach Kategorie von Linda Urech. PNG Image, 2014.

[5] Nicola Jordan and Linda Urech. Total aufgewendete Stunden. PNG Image, 2014.

[6] Nicola Jordan and Linda Urech. Stunden aufgeschlüsselt auf Kategorien. PNG Image, 2014.

[7] Wikipedia. Lean IT. http://en.wikipedia.org/wiki/Lean_IT, 2014.

[8] Wikipedia. Agile Manifesto. http://en.wikipedia.org/wiki/Agile_software_development, 2014.

[9] Nicola Jordan and Linda Urech. Nutzwertanalyse: Kandidaten. PNG Image, 2014.

[10] Nicola Jordan and Linda Urech. Nutzwertanalyse: Kriterien. PNG Image, 2014.

[11] Nicola Jordan and Linda Urech. Nutzwertanalyse: Gewichtung. PNG Image, 2014.

[12] Nicola Jordan and Linda Urech. Nutzwertanalyse: Bewertungserklärungen. PNG Image, 2014.

[13] Nicola Jordan and Linda Urech. Nutzwertanalyse: Auswertung. PNG Image, 2014.

[14] Nicola Jordan and Linda Urech. Prototype 1: Gestures Capture. JPG Image, 2014.

[15] Nicola Jordan and Linda Urech. Prototype 2: 3 Kameras vs schiefe Ebene. JPG Image, 2014.

[16] Nicola Jordan and Linda Urech. Prototype 3: Pong. PNG Image, 2014.

[17] Nicola Jordan and Linda Urech. Prototype 3: Finger Slip Erklärung. PNG Image, 2014.

[18] Nicola Jordan and Linda Urech. Beispiel eines Tetris-Multiplayer (Twinflix). PNG Image, 2014.

[19] Nicola Jordan and Linda Urech. Multiplayer via Netzwerk (Twinflix). PNG Image, 2014.

[20] Nicola Jordan and Linda Urech. Beispiel eines Netzwerk-Tetris-Multiplayer (Twinflix). http://hvaudit.blogspot.ch, 2014.

[21] Nicola Jordan and Linda Urech. Beispiel eines Netzwerk-Tetris-Multiplayer (Twinflix). http://www.hsr.ch/T-Touch-3D-Tetris.9218.0.html, 2014.

[22] Nicola Jordan and Linda Urech. Multitouch-Tisch Pixelsense. http://www.touchwindow.com/c/SamsungSUR40.html, 2014.

[23] Nicola Jordan and Linda Urech. Fighting Bereich. PNG Image, 2014.

[24] Nicola Jordan and Linda Urech. Wurmloch Portal. PNG Image, 2014.

[25] Nicola Jordan and Linda Urech. Röhren System. PNG Image, 2014.

[26] Nicola Jordan and Linda Urech. triDat Queue System Capture. PNG Image, 2014.

[27] Nicola Jordan and Linda Urech. Menu. PNG Image, 2014.

[28] Nicola Jordan and Linda Urech. Menu im Demomodus. PNG Image, 2014.

[29] Nicola Jordan and Linda Urech. triDat Spielfeld Capture. PNG Image, 2014.

[30] Nicola Jordan and Linda Urech. triDat Demo Mode Capture. PNG Image, 2014.

[31] Nicola Jordan and Linda Urech. triDat CoOp Mode Capture. PNG Image, 2014.

[32] Nicola Jordan and Linda Urech. triDat Vs Mode Capture. PNG Image, 2014.

[33] Nicola Jordan and Linda Urech. triDat verschiedene Blöcke. PNG Image, 2014.

[34] Nicola Jordan and Linda Urech. Planning Board. PNG Image, 2014.

[35] Linda Urech. Personas. Eingescannte Skizze., 2014.

[36] Nicola Jordan and Linda Urech. Vor Redesign: Quit Button. PNG Image, 2014.

[37] Nicola Jordan and Linda Urech. Start Over Funktion von Pixelsense. http://technet.microsoft.com/en-us/library/gg680399.aspx, 2014.

[38] Nicola Jordan and Linda Urech. Swipable Menu. PNG Image, 2014.

[39] Nicola Jordan and Linda Urech. Wolken Design Prototype. PNG Image, 2014.

[40] Nicola Jordan and Linda Urech. Risiko Matrix. http://www.peterjohann-consulting.de/index.php?menu-id=risk, 2014.

[41] Nicola Jordan and Linda Urech. Risiko Behandlungsstrategien. http://www.peterjohann-consulting.de/index.php?menu-id=risk, 2014.

[42] Nicola Jordan and Linda Urech. Risiko Prozess. http://www.peterjohann-consulting.de/index.php?menu-id=risk, 2014.

[43] Nicola Jordan and Linda Urech. Risiko Fingerslip. JPG Image, 2014.

[44] Nicola Jordan and Linda Urech. Architektur Übersicht. JPG Image, 2014.

[45] Nicola Jordan and Linda Urech. GameEngine, Script und GameObjects. JPG Image, 2014.

[46] Interactive Lab. Übersicht TouchPoint. https://github.com/InteractiveLab/TouchScript/wiki/The-Journey-of-a-Touch-Point, 2014.

[47] Interactive Lab. Übergang TouchManager - GestureManager. https://github.com/InteractiveLab/TouchScript/wiki/The-Journey-of-a-Touch-Point, 2014.

[48] Interactive Lab. Zuordnung eines TouchPoints. https://github.com/InteractiveLab/TouchScript/wiki/The-Journey-of-a-Touch-Point, 2014.

[49] Interactive Lab. Zuordnung eines TouchPoints in einer Hierarchie. https://github.com/InteractiveLab/TouchScript/wiki/The-Journey-of-a-Touch-Point, 2014.

[50] Wikipedia. Spline Interpolation. http://de.wikipedia.org/wiki/Spline-Interpolation, 2014.

[51] Nicola Jordan and Linda Urech. Grid Movement. PNG Image, 2014.

[52] Nicola Jordan and Linda Urech. Grid Movement Abklärungskette. PNG Image, 2014.

[53] Nicola Jordan and Linda Urech. Grid Movement New Block. PNG Image, 2014.

[54] Nicola Jordan and Linda Urech. Komponenten Diagramm. PNG Image, 2014.

[55] Nicola Jordan and Linda Urech. Komponenten Diagramm ab Fighting-Bereich. PNG Image, 2014.

[56] Nicola Jordan and Linda Urech. Skybox Splitting verwerfen. PNG Image, 2014.

Abbildungsverzeichnis

2.1 Soll/Ist Stunden [1] . . . 5
2.2 Stundenaufteilung nach Kategorie [2] . . . 5
2.3 Stundenaufteilung nach Kategorie von Nicola Jordan [3] . . . 5
2.4 Stundenaufteilung nach Kategorie von Linda Urech [4] . . . 6
2.5 Total aufgewendete Stunden [5] . . . 6
2.6 Stunden aufgeschlüsselt nach Kategorien [6] . . . 6

6.1 Nutzwertanalyse: Kandidaten [9] . . . 21
6.2 Nutzwertanalyse: Kriterien [10] . . . 21
6.3 Nutzwertanalyse: Gewichtung [11] . . . 21
6.4 Nutzwertanalyse: Bewertungserklärungen [12] . . . 22
6.5 Nutzwertanalyse: Auswertung [13] . . . 22
6.6 Prototype Gestures [14] . . . 27
6.7 Prototype 2 - 3 Kameras vs schiefe Ebene [15] . . . 28
6.8 Prototype 3: Pong [16] . . . 28
6.9 Fingerslip Erklärung [17] . . . 29

7.1 Beispiel eines Tetris-Multiplayer (Twinflix) [18] . . . 32
7.2 Multiplayer via Netzwerk (Twinflix) [19] . . . 32
7.3 Blocks auf dem alten Surface I Tisch [20] . . . 32
7.4 T-Touch [21] . . . 33
7.5 Multitouch-Tisch Pixelsense [22] . . . 34
7.6 Fighting-Bereich [23] . . . 36
7.7 Wurmloch Portal [24] . . . 37
7.8 Röhren-System [25] . . . 37
7.9 Queue [26] . . . 37
7.10 Menu [27] . . . 38
7.11 Menu im Demomodus [28] . . . 38
7.12 Spielfeld [29] . . . 38
7.13 DemoScreen [30] . . . 39
7.14 Co-Op Mode [31] . . . 39
7.15 Vs Mode [32] . . . 40
7.16 Verschiedene Blöcke [33] . . . 40
7.17 Planning Board für UserStories, Features und Tasks [34] . . . 40

8.1 vor Redesign: Quit Button [36] . . . 47
8.2 “Start Over” Funktion von Pixelsense [37] . . . 47
8.3 nach Redesign: SwipeableMenu [38] . . . 48
8.4 vor Redesign: Wolke [39] . . . 48

9.1 Risiko Matrix [40] . . . 52

9.2 Risiko Behandlungsstrategien [41] . . . 52
9.3 Risiko Prozess [42] . . . 52

10.1 Risiko Fingerslip [43] . . . 53

11.1 Architektur Übersicht [44] . . . 57
11.2 GameEngine, Script und GameObjects [45] . . . 58
11.3 Übersicht TouchPoint [46] . . . 59
11.4 Übergang TouchManager - GestureManager [47] . . . 59
11.5 Zuordnung eines TouchPoints [48] . . . 59
11.6 Zuordnung eines TouchPoints in einer Hierarchie [49] . . . 59

14.1 Grid-Movement: Übersicht [51] . . . 76
14.2 Grid-Movement: Abklärungskette [52] . . . 77
14.3 Grid-Movement: Kann ein neuer Block hinzugefügt werden? [53] . . . 77
14.4 Komponenten Diagramm (folgt keinem UML Standard) [54] . . . 79
14.5 Komponenten Diagramm ab Fighting-Bereich (folgt keinem UML Standard) [55] . . . 80
14.6 Skybox Splitting verwerfen [56] . . . 80

Tabellenverzeichnis

9.1 Risikoanalyse 51

12.1 Testlog der Usability Tests Teil 1 63
12.2 Testlog der manuellen Test Teil 1 66
12.3 Testlog der manuellen Test Teil 2 67

16.1 Glossar 87