

Abschlussdokumentation

Circuit

Teammitglieder:

Ben Kantwerk (Team lead)
Andrei Stein
Grigori Jordan
Philipp Hundelshausen

Inhaltsverzeichnis

1. Einleitung.....	1
1.1. Beschreibung des Projektes.....	1
1.2. Projektspezifische Kennzahlen.....	1
2. 2D und 3D-Prototypenphase.....	2
2.1. Ben Kantwerk.....	2
2.2. Andrei Stein.....	3
2.3. Grigori Jordan.....	4
2.4. Philipp Hundelshausen.....	5
3. Projektmanagement.....	6
3.1. Allgemeine Organisation des Projektes.....	6
3.2. Initiale Ideenfindung.....	7
3.3. Aufgabenbereiche des Teams.....	8
3.3.1. Ben Kantwerk.....	8
3.3.2. Andrei Stein.....	9
3.3.3. Grigori Jordan.....	10
3.3.4. Philipp Hundelshausen.....	11
3.4. Gegenüberstellung geplanter und erreichter Termine.....	12
3.4.1. Meilenstein I – 1.6.2012.....	12
3.4.2. Meilenstein II – 22.6.2012.....	12
3.4.3. Meilenstein III – 13.7.2012.....	13
3.4.4. Meilenstein IV / Testwochenversion – 3.9.2012.....	13
3.5. Analyse von Problemen und Projektabweichungen.....	14
3.5.1. Probleme mit GIT-Repositories.....	14
3.5.2. Unentschlossenheit bei der Wahl der Physik Engine.....	14
3.5.3. Probleme bei der Arbeit mit der Physik Engine.....	14
3.5.4. Probleme bei der Skelett-Animation.....	16
3.5.5. Probleme bei der Content Erstellung.....	17
3.6. Methoden der Qualitätssicherung.....	20
3.6.1. Qualitätssicherung innerhalb der Veranstaltungen.....	20
3.6.2. Qualitätssicherung in Teammeetings.....	20
3.6.3. Qualitätssicherung der Performance.....	20
3.6.4. Ergebnisse und Einflüsse der Testwoche.....	21

3.7. Verwendete Tools und Bibliotheken.....	22
3.7.1. BEPUphysics.....	22
3.7.2. Maya / 3ds Max.....	22
3.7.3. bfxr.....	22
3.7.4. Crazy Bump.....	23
3.7.5. Gimp.....	23
3.7.6. Spacescape.....	23
4. Technische Details.....	24
4.1. Glow-Shadereffekt.....	24
4.2. Waffensystem.....	25
4.2.1. Arsenal-Klasse.....	25
4.2.2. Factory-System.....	26
4.2.3. Fragmentwolken.....	27
4.4. Optimierungen.....	28
5. Ergebnisse des Projektes.....	29
5.1. Vergleich des Projektergebnisses mit dem Game Design Dokument.....	29
5.2. Begründung wichtiger Designentscheidungen.....	30
5.2.1. Feedback.....	30
5.2.2. Interface.....	30
5.2.3. Core Mechanics.....	30
5.2.4. Wahl der Physik-Engine.....	31
5.2.5. Änderungen am Game Design.....	32
6. Anhang.....	32
6.1. Referenzen.....	32
6.2. Begleitendes Bildmaterial.....	33
6.3. Konzeptzeichnungen.....	35
6.4. Screenshots alter Versionen.....	38

1. Einleitung

1.1. Beschreibung des Projektes

Circuit ist ein Third-Person Shooter, welcher im Rahmen des Acagamics Softwareprojektes 2012 entstanden ist. Zwei Spieler treten dabei im Splitscreen gegeneinander an.

Die nachfolgende Dokumentation dient dabei der abschließenden Zusammenfassung des Projektablaufes, sowie die Präsentation der Projektergebnisse.

1.2. Projektspezifische Kennzahlen

- Anzahl geschriebener Klassen: 111
- Anzahl Vertices des Charakter Modells: 2580
- Durchschnittliche Anzahl Frames pro Sekunde: 58
- Dauer des Projektes: 13.4.2012 – 28.9.2012

2. 2D und 3D-Prototypenphase

2.1. Ben Kantwerk

2D-Prototyp



Abbildung 1: 2D-Prototyp, Einfaches Reaktionsspiel

Ein Reaktionsspiel, bei dem der Spieler eine von neun eingeblendeten Tasten drücken muss. Bei rechtzeitiger Eingabe weicht der Abenteurer einer Falle aus, andernfalls endet das Spiel. Dieser Prozess wird solange fortgesetzt, bis eine Eingabe nicht rechtzeitig erfolgt. Das Thema "Feedback" äußert sich insbesondere durch die 2D-Animationen des Charakters, sowie das Aufleuchten

des erwarteten Inputs. Dieser Prototyp hatte aufgrund seiner geringen Komplexität jedoch nur wenig direkten Einfluss auf das eigentliche Softwareprojekt.

3D-Prototyp

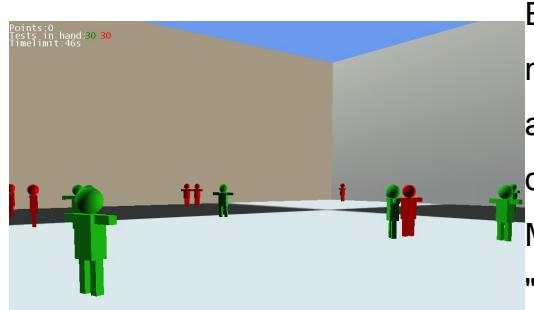


Abbildung 2: 3D-Prototyp, Einfacher Shooter

Ein First-Person-Shooter bei dem "Schüler" mit "Noten" der korrekten Farbe abgeschossen werden müssen. Durch drücken der linken Maustaste und rechten Maustaste werden grüne bzw. Rote "Noten" abgefeuert. Wenn die Farbe der "Noten" mit der Farbe der "Schüler" übereinstimmt, gibt es Bonuspunkte, andernfalls werden Punkte abgezogen. Im

Anschluss an das Spiel wird der Spieler anhand der erreichten Punktzahl selbst benotet. Das Thema "Belohnung und Bestrafung" äußert sich durch das Punktesystem und die abschließende Bewertung des Spielers.

2.2. Andrei Stein

2D-Prototyp



Abbildung 3: 2D-Prototyp, Bounce

Bounce ist ein Geschicklichkeitsspiel bei dem es darum geht einen Ball sicher durch das Level zu navigieren um ihn ans Ziel zu bringen. Gesteuert wird das Spiel durch Veränderung der Richtung der Erdanziehungskraft. Das besondere an Bounce ist, dass sich der Ball in einen Flummi, eine Metallkugel und in eine Seifenblase verwandeln kann. Jede Ballart verhält sich physikalisch anders. Es gibt für jede Ballart Objekte in der Umgebung, die der Ball nicht berühren darf. Für die Kollisionserkennung wurde eine pixelgenaue Kollisionserkennung implementiert.

Andreis Hauptaufgabe war es, den Screenmanager zu schreiben. Außerdem war er für die Level-Erstellung und die Soundeinbindung zuständig.

3D-Prototyp

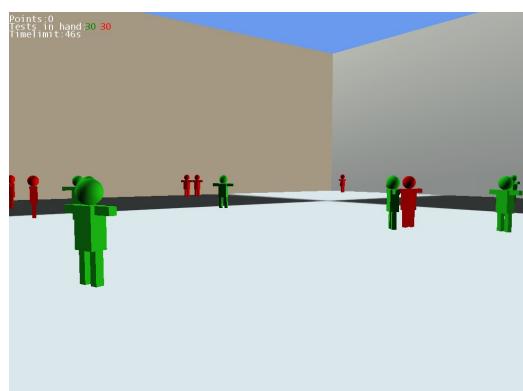


Abbildung 2: 3D-Prototyp, Einfacher Shooter

Der 3D-Prototyp ist ein einfacher First-Person-Shooter bei dem es darum geht, gute Schüler in Grün mit guten Zeugnissen zu belohnen und schlechte in Rot mit Schlechten Zeugnissen zu bestrafen. Der Spieler bekommt Punkte abgezogen, wenn er einem Schüler das falsche Zeugnis gibt. Die Schüler-Modelle sind dabei ständig in Bewegung. Andrei war für die Erstellung und das Verhalten der Schüler und für die Kollisionsbehandlung zuständig.

2.3. Grigori Jordan

2D Prototyp

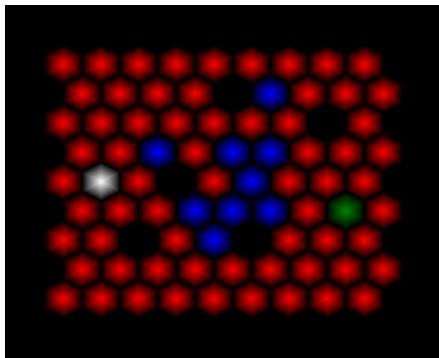


Abbildung 4: 2D-Prototyp, Ingame Screenshot

Das Konzept unseres 2D Prototypen bestand darin ein 2D Puzzlespiel zu machen, basierend auf hexagonalen Feldern. Der Spieler bewegt ein markiertes Feld über eine Map mit Hindernissen, diese sind zum einen statische hexagonale Felder und zum anderen sich um den Spieler befindliche, ihm folgende hexagonale Felder. Diese sind dabei hinderlich platziert und versperren das Zielfeld, wenn der Spieler sich darauf zu bewegt.

Sie lassen sich jedoch verschieben wenn sie gegen ein statisches Feld bewegt werden.

Grigori war für die visuell-technische Umsetzung verantwortlich. Besonders zu erwähnen ist, dass die Felder komplett auf Vertices basieren, die alle manuell gesetzt und indiziert wurden. So wird die Map in einer einzigen Methode erstellt. Das Aufleuchten der Felder bei einer Kollision wurde ebenfalls von Grigori implementiert.

3D Prototyp



Abbildung 5: 3D-Prototyp, Ingame Screenshot

Die Spielidee hinter dem 3D Prototypen war eine Art Puzzlespiel kombiniert mit Jump'n'Run. Ziel ist es die Spitze eines Turmes über Plattformen zu erreichen. Dabei muss die Kamera um die vier Seiten des Turms gedreht werden, um ganz nach oben zu gelangen. Auch gab es rote und grüne Power-Ups die den Spieler entweder belohnen oder bestrafen. Grigoris Anteil am Projekt bestand wiederum an der Grafik und Modellierung.

2.4. Philipp Hundelshausen

2D-Prototyp

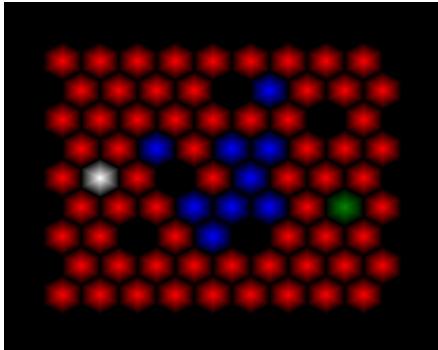


Abbildung 4: 2D-Prototyp, Ingame Screenshot

Der 2D Prototyp war für ein simples Puzzle-game. Der Spieler steuert den weißen Spielstein und muss an die grün markierte Position gelangen. Er kann sich jeweils um ein Feld in einer von sechs Richtungen bewegen, wobei blaue und schwarze Felder Hindernisse darstellen. Die blauen Steine versuchen sich jeweils in gleicher Manier wie der Spieler zu bewegen, sodass der Spieler diese durch geschicktes Manövrieren umgehen muss. Philipps Anteil an diesem Prototyp waren die Spielidee und das Gamedesign, sowie der technische Teil des Gameplays. Hierbei erwähnenswert ist das eventgesteuerte Verhalten der blauen Spielsteine und die Routine zur Erkennung und Differenzierung von Kollisionen, welche außerdem jeweils ein optisches Feedback an schwarzen Steinen auslösen. Das Gameplay war insofern im Sinne des Themas 'Feedback' designed, als dass das gesamte Spielfeld auf die Bewegungen des Spielers reagiert, indem alle blauen Hindernisse sich mitbewegen.

3D-Prototyp



Abbildung 6: 3D-Prototyp, Menü

Der 3D Prototyp stellte eine digitale Portierung des Spiels Jenga dar. Klassischerweise versuchen zwei Spieler abwechselnd, einen Stein aus dem Turm zu ziehen und ihn wieder auf diesem zu plazieren, beides ohne dass der Turm einstürzt. Philipp implementierte den physikalischen Teil des Gameplays (und die Einbindung der Jitter Physics Engine^[1]) sowie die Einbindung der Assets. Das Gameplay sollte insofern das Thema 'Belohnung/Bestrafung' wiederspiegeln, als dass ein ungeschicktes Spielverhalten in der zunehmenden Instabilität des Turms resultiert.

3. Projektmanagement

3.1. Allgemeine Organisation des Projektes

Durch die Beteiligung von vier Personen an diesem Projekt kam es schon sehr früh im Prozess zu einer groben Aufteilung der Teammitglieder in einzelne Teilbereiche, jeweils zwei für die Bereiche Gameplay und Grafik, um individuelle Spezialisierungen und Interessen auszunutzen.

Um sicherzustellen, dass das Projekt sowohl den Anforderungen der Veranstaltung genügt, als auch den zeitlichen Rahmen erfüllt, wurden wöchentliche Teammeetings durchgeführt; in der vorlesungsfreien Zeit fanden diese sogar zweimal wöchentlich statt.

Ziel der Teammeetings war es dabei, den Fortschritt der Teammitglieder zu verfolgen, offene Fragen und Probleme innerhalb der Gruppe zu klären, sowie anstehende Meilensteine zu planen.

3.2. Initiale Ideenfindung

Ziel der Ideenfindung war es, neben einem spannenden Spielekonzept auch eine sinnvolle Einbindung der Physik-Engine in das Projekt zu erzielen.

Angestrebt wurde dabei ein schnelles, spaßiges Spielekonzept, das keine lange Einarbeitungszeit benötigt.

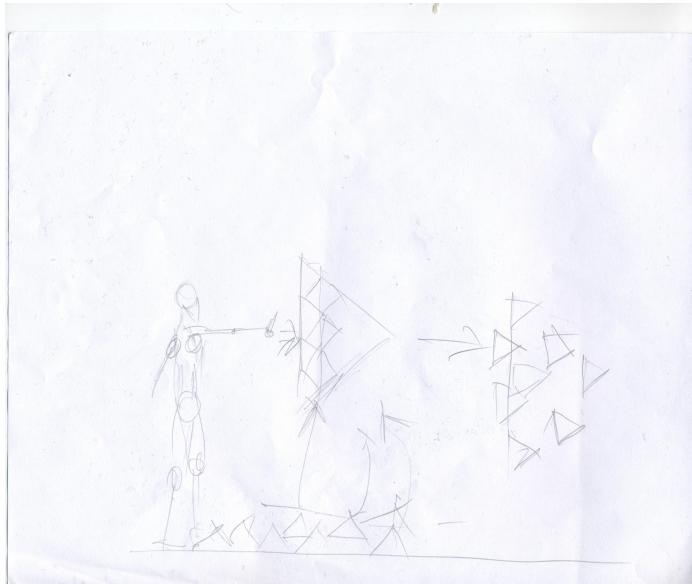


Abbildung 7: Erste Konzeptzeichnung des Projektes

Die Idee war ein kompetitiver Third-Person Shooter mit einfachen geometrischen Formen als Waffen. Zwei Spieler bekämpfen sich dabei im Splitscreen. Die simplen geometrischen Objekte eigneten sich sehr gut dafür, die Einflüsse der Physik zu visualisieren, wie die akkurate Kollision und rotatorische Einflüsse.

Die Entscheidung für ein passendes Setting fiel letztendlich auf ein steriles, futuristisches Sci-Fi Design. Ein wichtiges Merkmal dieses Settings war der Entschluss, auf die Beleuchtung großen Wert zu legen, was letztendlich auch wichtige Einflüsse auf das Charakter-Design selbst hatte. Insgesamt lässt sich sagen, dass aus dem Setting und der Ideenfindung bereits sehr früh Ideen entstanden sind, welche auch für das finale Projekt selbst verwendet wurden, insbesondere bezüglich des Feedbacks und des Interface.

Weitere Konzept Zeichnungen und Screenshots alter Projekt-Versionen befinden sich im Anhang (siehe Punkt 6.3. und 6.4.).

3.3. Aufgabenbereiche des Teams

3.3.1. Ben Kantwerk

Neben der Position als Team lead war Ben auch für das Game Design und somit für den Entwurf der Core Mechanics verantwortlich.

Sein Code Anteil liegt dabei hauptsächlich bei der Einbindung der Gamepad spezifischen Steuerung, sowie Implementierungen in den Bereichen Feedback und allgemeiner Spiel-Logik, auf welche im Folgenden kurz etwas näher eingegangen wird.

Besonders wichtig für das Feedback sind in diesem Sinne die Erstellung und Einbindung aller Soundeffekte, das Anpassen des Leuchteffektes an den aktuellen State des Spielers (aktuelle Lebenspunkte und benutzte Boost-Effekte), sowie das Einblenden von Nachrichten während des Spiels, z.B bei Ende einer Runde oder Tod eines Spielers.

Bens größter Anteil am Code liegt jedoch bei der Implementierung der Spiel-Logik. Wichtige Implementierungen in diesem Bereich sind die Systeme für Lebenspunkte, den Punktestand beider Spieler und das "Respawnen" nach Spieler-Tod. Des Weiteren ist Ben für die Implementierung der Credits, der "Instructions" (als Einführung für neue Spieler) und des Optionsmenüs (zum Anpassen der Kamera-Steuerung und der Spielregeln) verantwortlich.

3.3.2. Andrei Stein

Anrei interessierte sich sehr für die Shaderprogrammierung, mit der er bis dahin wenig Erfahrung hatte. Um sich in diesem Bereich weiterzuentwickeln, übernahm er die Position des Graphics-Programmers innerhalb des Teams.

Seine Hauptaufgabe bestand darin den Normal-Map-Effekt und den Glow-Effekt im Spiel zu implementieren. Aufgrund dessen war er auch für Erstellung der Hilfsklassen für die Einbindung der Effekte verantwortlich. Dazu gehören die Material-Klassen und die Entity-Klasse, die unter anderem für die grafische Repräsentation eines Objektes zuständig ist.

Für den Glow-Effekt wurde zusätzlich zum Shader ein Postprocessing-Framework erstellt, welches das Rendern in eine Textur erleichtert. Außerdem war er für die Erstellung des Levels und die Code-Einbindung der Animationen zuständig.

3.3.3. Grigori Jordan

Seine Aufgaben ließen sich am ehesten als die eines Technical Artists beschreiben. Also als jemand der im allgemeinen für die Grafik auf der Code-, als auch der Content-, sowie Artist-Ebene verantwortlich ist.

Er hat bezüglich des Artist-Anteils das konzeptuelle Design des Spieles geprägt. Auch war er für das Characterdesign verantwortlich, woraus die Game-Designentscheidung resultierte, das HUD auf den Character zu reduzieren und z.B. die Lebensanzeige auf diesem durch leuchtende Streifen darzustellen.

Um den Content-Anteil erfüllen zu können musste er sich fast von Grund auf in das 3D-Tool Maya^[2] einarbeiten. Dies war mit mehr Hindernissen verbunden als zunächst angenommen. Abschließend kann er sagen, dass er den Aufwand unterschätzt habe. Aber dennoch hat er auf diesem Gebiet abgesehen von der Levelerstellung alles bewältigen können.

Die Prozesse die bei der Charaktererstellung zu durchlaufen waren, das eigentliche Modellieren und die Texturierung, das "Rigging" (Erstellen eines Animationsskelettes) und anschließend das Erstellen der eigentlichen Animationen per "Key-Frame" Animation.

Die Texturierung war dabei wohl der am wenigsten problematische Teil.

Auch das Erstellen der Normal-Maps war angenehm, dank Crazy Bump^[3].

Technisch war er für das Implementieren und Einstellen der finalen Kamera verantwortlich. Dieser Teil war dadurch aufwändiger als notwendig, weil sich aufgrund des Fortschrittes in unserem Projekt bereits Abhängigkeiten von der Kamera gebildet haben.

Die finale Kamera orientiert sich an einer Third-Person Verfolgungskamera mit einem einfachem Federeffekt, der der Kamera Masse verleiht, welche sich somit realistischer anfühlt^[4].

Darauf folgte nur noch eine Justierung der Kamera, so dass diese in Schussrichtung blickte.

3.3.4. Philipp Hundelshausen

Philipp erfüllte im wesentlichen die Rolle des Lead-Programmers.

In diesem Rahmen stellte er die Code-Konventionen auf und entwarf realisierte die grundlegende Architektur des Spiels.

"Grundlegende Architektur" bezieht dabei das Folgende ein: Verwaltung der Gamestates, grundlegende Datenstrukturen für Spielobjekte, Einbindung der Physik-Engine, grundlegende Struktur der Arena- und Spieler Klassen.

Weitere Arbeitsgebiete waren das Waffensystem, die physische Representation des Spielers und der Arena sowie die zerstörbaren Objekte.

Er war es, der den Wunsch bzw. den Vorschlag vorbrachte, eine Physik-Engine in das Spiel einzubinden (aus persönlichem Interesse). Erfahrungen in diesem Gebiet beschränkten sich bis dahin auf einige Gehversuche im Rahmen der Veranstaltung "Introduction to 3D Game Development" im Wintersemester 2011/2012, sowie im 3D Prototyp der der Entwicklung dieses Spiel vorgelagert war (siehe Punkt 2.4.). Auch beschränkten sich diese Gehversuche auf eine andere (ebenfalls Open Source) Physik-Engine, namens Jitter Physics.

3.4. Gegenüberstellung geplanter und erreichter Termine

Im Folgenden soll anhand der Meilenstein Termine aus der Veranstaltung verglichen werden, inwieweit geplante Features rechtzeitig implementiert wurden. Eine Analyse der aufgetretenen Probleme und Verzögerungen befindet sich im Punkt 3.5.

Desweiteren befindet sich eine Abbildung des ersten Projektplans, sowie Screenshots des Projektes zu Meilensteinpräsentationen im Anhang (siehe Abbildung 14, bzw. Punkt 6.4.)

3.4.1. Meilenstein I – 1.6.2012

Ziel dieses Meilensteines war die Anfertigung und Präsentation des Technical Design Dokumentes. In diesem Punkt kam es zu keinen Verzögerungen sodass dieser Meilenstein rechtzeitig erfüllt werden konnte.

3.4.2. Meilenstein II – 22.6.2012

Für Meilenstein II waren folgende wichtige Features geplant:

- eine vollständig implementierte Waffe mit allen Funktionen (Shot, Boost, Trap)
- vollständig implementierte Kamera, inklusive Splitscreen
- anfängliche Menüstruktur
- eine Test-Stage inklusive eingebundener Physik
- Normalmapping-Effekt
- erste Version des Spielermodells
- Steuerung des Charakters vollständig implementieren

Bis auf zwei Ausnahmen, das Spielermodell und die Implementierung der ersten Waffe, konnten diese Anforderungen erfüllt werden.

3.4.3. Meilenstein III – 13.7.2012

Die wichtigste Zielsetzung für diesen Meilenstein war es, den Prototypen spielbar zu machen. Dafür mussten folgende Punkte realisiert werden:

- vollständige Implementierung der Spielregeln (Lebenspunktesystem, Respawn nach Spieler-Tod, Munitionssystem, Siegbedingungen)
- eine "spielbare" Arena
- Aufholen der Verzögerungen aus Meilenstein II

Diese Punkte wurden vollständig implementiert; durch die Verzögerungen in Meilenstein II war die erste Version des Charaktermodells jedoch untexturiert. Des Weiteren wurden folgende Features über den geplanten Rahmen hinaus implementiert:

- Optimierung der Performance durch Frustum Culling
- Ausrichtung der Kamera am Spielermodell
- Zerstörbare Objekte
- Weitere Verbesserungen an der Physik

3.4.4. Meilenstein IV / Testwochenversion – 3.9.2012

Für den letzten großen Meilenstein des Projektes ging es insbesondere um Feedback sowie das Ausarbeiten der Menüs und des Gameplays. Insgesamt waren folgende Punkte geplant:

- Wichtiges Feedback (z.B. Soundeffekte, Glow-Effekt als HUD-Ersatz)
- Ausarbeitung der Menüs
- Fertigstellung des Charakters (Texturen und Animationen)
- Implementation von Shader-Effekten (Glow-Effekt, Depth of Field)
- Fertigstellen der zweiten Waffe

Auch dieser Meilenstein konnte zu einem Großteil realisiert werden, jedoch kam es wiederum zu Verzögerungen bei der Erstellung und Einbindung der Charakter Animationen. Auch hier konnten Features über den geplanten Rahmen hinaus implementiert werden: rechtzeitig zur Testwoche wurden erste Implementierungen der dritten und vierten Waffe fertiggestellt.

3.5. Analyse von Problemen und Projektabweichungen

3.5.1. Probleme mit GIT-Repositories

Besonders zu Beginn des Projektes traten Probleme bei der Arbeit mit GIT-Repositories auf, da keines der Teammitglieder Erfahrung beim Umgang mit diesen hatten. Dies führte dazu, dass Code-Stücke im Teammeeting von Hand "germerged" werden mussten. Diese Probleme wurden mit zunehmender Erfahrung geringer, aber hatten dennoch einen Effekt auf den Anfang des Projektes.

3.5.2. Unentschlossenheit bei der Wahl der Physik Engine

Obwohl bereits sehr früh in der Ideenfindung die Entscheidung fiel eine Physik-Engine einzubinden, gab es Unschlüssigkeiten darüber welche Engine konkret verwendet werden sollte. Erst auf das Anraten der Veranstaltungsleiter, früh im Prozess eine Entscheidung zu fällen, kam es letztendlich zur endgültigen Festlegung, die BEPUphysics^[5] Physik-Engine zu verwenden (anders als im Game Design und Technical Design).

3.5.3. Probleme bei der Arbeit mit der Physik Engine

Aufgrund des Mangels an Erfahrung mit Physik Engines war die Arbeit mit derselben bisweilen ein langwieriger Prozess. In manchen Fällen wurden weite Codesegmente verworfen, da die entsprechenden Features entweder nicht physikalisch umsetzbar waren, oder schlicht die Kenntnis von nötigen Strukturen der Engine nicht vorhanden oder unzureichend war. Häufig mussten Klassenstrukturen der Engine recherchiert werden.

Diesen Problemen wurde verschiedener Art begegnet:

Im Falle des Cloudsystems (siehe 4.2.3) (sowohl für die Clouds selbst als auch der CloudManager) fehlte die Kenntnis effektiver Methoden, physische Objekte an bestimmte Positionen zu zwingen (Motoren) oder deren Bewegungsfreiheit einzuschränken (Constraints). Mit dem jetzigen Kenntnisstand wäre dieses System anders umgesetzt worden.

Im Falle der physikalischen Representation und Bewegung des/der Spieler wurde ebenfalls zunächst (eingeschränkt erfolgreich) experimentiert. Die Lösung dieses Problems bestand in der Assimilierung eines derartigen Systems welches in einer Demonstration der Engine eingesetzt wurde.

Ein weiteres Problem stellte das Kollisionsverhalten von Geschossen dar. In einem hohen Anteil der Fälle wurde eine Kollision zwischen Geschoss und Spieler oder, häufiger, zwischen Geschoss und Arenamodel nicht von der Physiksimulation erkannt. Gelöst wurde dies einmal seitens der Geschosse selbst, indem deren physikalische Representation vergrößert wurde, sowie die Geschwindigkeit herabgesetzt wurde.

Der andere Teil der Lösung lag im Kollisionmodel der Arena. Bis dahin wurde für dieses nur eine Representation verwendet, die mit Methoden der Engine aus dem grafischen Model erstellt wurde. Weite Teile der Arena wurden durch physikalische Primitive (Quader) erweitert, welche (unsichtbar) in Wände und Boden integriert wurden.

Performance bezogene Probleme entstanden vor allem durch die zerstörbaren Objekte des Spiels. Diese Objekte bestehen aus einer hohen Zahl von Fragmenten welche in einer geometrischen Formation angeordnet sind. Kommt es zu einer Kollision eines Fragments mit einem Geschoss so wechselt es von einem statischen zu einem dynamischen Verhalten. Dabei kam es zu Performanceeinbrüchen, da sehr viele Kollisionen zur gleichen Zeit zwischen Fragmenten und Arena sowie Fragmenten untereinander kam. Dem wurde dadurch begegnet dass durch Festlegung einer Regel in der Simulation die Kollision zwischen Fragmenten untereinander unmöglich gemacht wurde. Dadurch konnten die Einbrüche zumindest stark abgeschwächt werden. Weitere Probleme entstehen allerdings durch die hohe Anzahl der zu zeichnenden Objekte in Form der Fragmente.

Eine Verbesserung der Performance wurde außerdem erreicht durch ein Engine internes Feature welches Multithreading ermöglicht.

Ansonsten kam es an verschiedenen Stellen zu unvorhergesehem Verhalten von physischen Objekten, durch Phänomene wie extremer Geschwindigkeitsanstieg und ähnliches.

Als ein Fazit ist die Arbeit mit der Engine im Rahmen dieses Projekts als sehr ertragreich und lehrreich zu bewerten.

3.5.4. Probleme bei der Skelett-Animation

Der Content-Pipeline-Processor von XNA 4.0^[6] kann keine 3D-Modelle mit Skinned-Animationen verarbeiten. Aus diesem Grund musste Andrei den Content-Pipline-Processor mit Hilfe des Beispiels von (XNA Creator's Club – Skinned Model Sample^[7]) erweitern.

Für das Animieren des Models ist der SkinEffekt-Shader zuständig, der schon in XNA enthalten ist. Da der SkinEffekt-Shader schon in XNA verankert ist, kann man diesen nicht einfach erweitern oder anpassen. Damit ergibt sich das Problem, dass der Spieler-Charakter zwar animiert werden kann, jedoch würde der Glow-Effekt nicht mehr funktionieren.

Nach eingehender Recherche wurde dann festgestellt, dass es zwar möglich ist einen eigenen SkinEffekt-Shader zu schreiben, dies jedoch viel Zeit in Anspruch nehmen würde. Aufgrund des Zeitmangels stand das Team vor der Wahl, den Glow- oder Skin-Effekt (für die Animation) für den Spieler zu benutzen. Der Glow-Effekt ist maßgeblich für das Feedback des Spiels zuständig. Deswegen hat sich das Team für den Glow-Effekt entschieden.

3.5.5. Probleme bei der Content Erstellung

Die Qualitätssicherung des Contents, für den er den Großteil seiner Zeit aufgebracht hat, erfolgte bei ihm in erster Linie durch eine dritte Person, die im Umgang mit dem 3D-Softwaretool mehr Erfahrung hatte.

So wurde ihm zum einen das Überarbeiten des Modells und später die Überarbeitung des Modell Riggs vorgeschlagen. Dies geschah zugunsten eines effizienteren Ergebnisses für die Erstellung von Animationen.

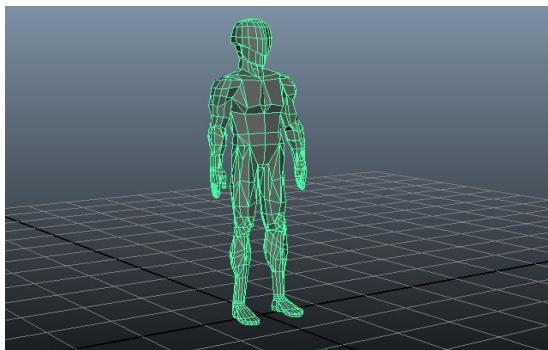


Abbildung 8: Fehlerhaftes Charakter-Modell

Der Fehler den er beim Modellieren gemacht hatte, war, dass Dreiecksformen im Bereich der Gelenke benutzt wurden, was dazu geführt hätte, dass nach dem Skinnig beim Beugen der z.B. Knie Spitzen herausragen würden.

Zuerst hatte er nach Möglichkeiten der Neuanordnung der Geometrie gesucht, wie z.B. das "Retopologising", bei der neue "gut" strukturierte Geometrie (aus Quads bestehend) an die alte entweder falsch strukturierte oder zu hoch augelöste angelegt und angepasst wird. Da in seinem 3D-Modellierungstool diese Möglichkeit nur durch Umwege erreicht werden konnte, die sehr zeitaufwändig gewesen wären, hat er sich entschlossen das Modell komplett neu zu modellieren.

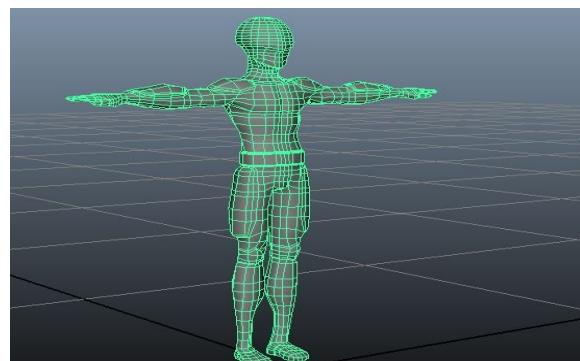


Abbildung 9: Verbessertes Charakter-Modell

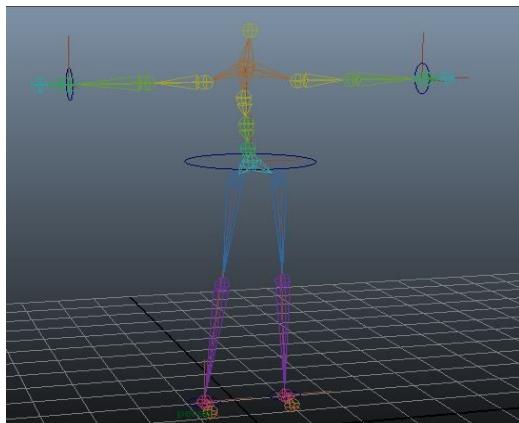


Abbildung 10: Fehlerhaftes Rig

Der zweite Fehler, beim Rig, war, dass der Root Joint war falsch gesetzt worden war. Statt im Beckenbereich am Übergang des Rückens zum Nacken.
Dieser Fehler hätte das Animieren dahingehend unnötig schwer gemacht, dass bei jeder Beugung des Oberkörpers jeder Joint der Wirbelsäule durch Translation UND Rotation hätte angepasst werden müssen.

Bei der korrigierten Version kann mit einer Rotation des Becken-Joints der ganze Oberkörper ab diesem Joint mitgebeugt werden und es sind nur noch kleine rotatorische Anpassungen an den anderen Wirbelsäulen-Joints nötig, um jede beliebige Beugung des Oberkörpers schnell darzustellen.

Ein weiteres Problem das sich nach den Animationen ergab, war das Exportieren des Modells mit seiner Textur und den Animationen in das korrekte Format für die Integration in das Spiel. Dafür kamen in diesem Projekt zwei Formate in Frage, zum einen das ".x"-Format und zum anderen das ".fbx"-Format.

Zuerst schien das ".x"-Format die bessere Wahl zu sein, da es nach dem Export keine Probleme mit den Animationen gab und man diese beim Export sehr bequem unterteilen und benennen konnte. Allerdings kam es hierbei zu Problemen mit dem Texturpfad, der angeblich immer falsch angegeben war.

Um nicht noch mehr Zeit zu verlieren wurde mit dem ".fbx"-Format weitergearbeitet. Hier war es anfangs genau umgekehrt, die Texturen machten keine Probleme, jedoch die Animationen (siehe Abbildung 12). Nach vielen Fehlversuchen gelang es letztendlich doch, dass alles in diesem Format funktionierte, sowohl die Textur, als auch die Animationen.

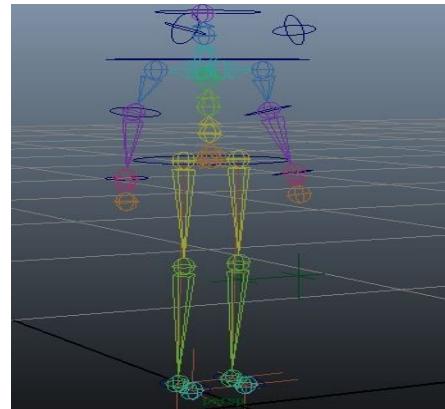


Abbildung 11: Verbessertes Rig

Der Grund weshalb es zu diesen Fehlern im Export in das ".fbx"-Format sind uns nicht ganz klar, aber um diesen Fehler zu vermeiden hat es sich als hilfreich herausgestellt einfach alles in das gewünschte Format zu exportieren und sukzessiv all das zu entfernen, was nicht benötigt wurde, um es am Ende nochmal in das gewünschte ".fbx"-Format korrekt exportieren zu können.



Abbildung 12: Fehlerhafte Animation

Seine Schwierigkeiten lagen in den ersten beiden beschriebenen Fehlern bei der Contenterstellung in typischen Anfängerfehlern, wie er von seinem Ansprechpartner bezüglich des Contents erfahren konnte. Die Plan- abweichungen

bezüglich seines Arbeitsanteils hingegen wurden durch seine Fehleinschätzung des Aufwandes der Charaktererstellung zum einen, sowie durch persönliche unvorhergesehene Probleme zum anderen notwendig.

Auch kam es bei Maya anfangs häufig zu abstürzen und Performance-problemen, was öfter die Fortschritte zunichte machte. Dieses Problem rührte zum einen von anfänglichem Unwissen her, sowie zum anderen von einer instabilen Version von Maya, wie im Nachhinein klar wurde.

3.6. Methoden der Qualitätssicherung

3.6.1. Qualitätssicherung innerhalb der Veranstaltungen

Nach jeder Meilenstein-Präsentation wurde Feedback von René Hoyer und Enrico Gebert eingeholt. Bei den Gesprächen wurde der aktuelle Status des Projektes beurteilt.

Außerdem wurde auf zu erwartende Probleme hingewiesen und es wurden Ratschläge gegeben um diese Probleme zu umgehen. So wurde z.B. das Team davor gewarnt, dass die Kombination des Split-Screens mit der Third-Person Kamera in einer eingeschränkten Sicht resultieren könnte. So konnte dieses Problem frühzeitig umgangen werden, indem Lösungsansätze in bestehenden Spielen mit ähnlichen Konstellationen recherchiert wurden.

3.6.2. Qualitätssicherung in Teammeetings

Es wurde wöchentlich ein Teammeeting abgehalten. Dabei wurden die Ergebnisse der Teammitglieder vom gesamten Team kritisch beurteilt. Bei mangelnder Qualität wurden Verbesserungsvorschläge gesammelt und von der verantwortlichen Person umgesetzt.

Dieses Vorgehen war sehr nützlich, weil es nicht immer leicht ist für eine Person, die Qualität der eigenen Ergebnisse zu beurteilen. Außerdem wurden beim Meeting Erfahrungen und Probleme der durchgeföhrten Aufgaben besprochen.

3.6.3. Qualitätssicherung der Performance

Zur Sicherung der Performance wurde ein FPS-Counter in das Spiel eingebaut. Damit konnte die Anzahl der Frames pro Sekunde überwacht werden.

Dies war sehr nützlich um festzustellen, in welchen Situationen im Spiel die Framerate einbricht. Außerdem konnte man durchgeföhrte Optimierungen am Spiel leichter erkennen.

3.6.4. Ergebnisse und Einflüsse der Testwoche

Im Test-Wochen Feedback haben wir weitestgehend positive Rückmeldungen erhalten, besonders in Bezug auf unsere Spielemechanik, dass dieses funktioniere und Spass mache.

Sehr wertvoll für uns waren auch die Kritikpunkte. Der größte Kritikpunkt bei uns war wohl das Feedback, welches z.B. nach dem Gewinn oder Verlust eines Spieles fehlte, sowie wenn man eine neue Runde begann.

Weitere Kritikpunkte verwiesen auf die noch teils fehlerhafte Funktionsweise und sich daraus ergebende Dysbalance des Waffensystems: Die Zweitfunktionen einiger Waffen waren noch nicht intakt und unterlagen somit den Waffen die funktionierten. Auch waren einige Waffen noch nicht in der Lage nach unten zu schießen, so dass man von höheren Plattformen einen sich tiefer befindenden Spieler nicht treffen konnte.

Dank dieser Kritik konnten wir die Qualität unseres Spieles durch Behebung dieser Bugs steigern.

3.7. Verwendete Tools und Bibliotheken

3.7.1. BEPUphysics

Die verwendete Physik Engine BEPUphysics ist ein freies Open Source Projekt; die von uns genutzte Version war 1.2.0.

Sie beeinhaltet zahlreiche klassische Features wie verschiedenste RigidBody Primitive, Joints und Constraints, Motoren, Raycasting, Multithreading und Weitere.

Sie wurde generell zur physikalischen Simulation des Spiels verwendet, sowie für die Kollisionserkennung zwischen Spielern und Geschossen u.Ä.

Die Website der Engine enthält Erklärungen zur Verwendung der verschiedenen Features und bietet online kompletten Einblick in den Source Code (selbstverständlich auch zum Download angeboten). Desweiteren sind Demo- und Tutorial-Projekte erhältlich.

3.7.2. Maya / 3ds Max

Autodesk Maya 2013 und Autodesk 3ds Max 2013 waren die 3D-Tools unserer Wahl um den Charakter und das Level zu erstellen. Dabei wurden alle notwendigen Schritte vom Modellieren bis hin zu Exportieren des Charakters über Maya geregelt, wogegen mit 3ds Max das Level erstellt wurde. Anfängliche Performanceprobleme gab es mit der früheren Version von Maya, weshalb der Wechsel auf Maya 2013 erfolgte.

3.7.3. bfxr^[8]

bfxr ist eine Erweiterung des Tools Sfxr, einem Programm zur Erstellung von Soundeffekten. Die Soundeffekt im Projekt wurden mit Version 1.3.2. erstellt.

3.7.4. Crazy Bump

Crazybump ist ein sehr hilfreiches Tool um schnell verschiedenste Maps zu generieren, von Normal Maps über Specular Maps bis hin zu Gloss Maps. In unserem Fall haben wir damit die Normal Maps für Charakter und Level erstellt. Es wurde die Demo der Version 1.2. verwendet.

3.7.5. Gimp^[9]

GIMP wurde von uns für das Erstellen der Farbtexturen der Modelle verwendet. Dieses Tool ist ein Zeichen- und Malprogramm mit vielen erweiterten Funktionen, ähnlich Adobe Photoshop, jedoch im Umfang stark verringert, aber dennoch für unsere Zwecke ausreichend. Es wurde Version 2.6.11 verwendet.

3.7.6. Spacescape^[10]

Spacescape ist ein Tool zur Erstellung von Grafiken für eine Skybox. Verwendet wurde die Version 0.3.

4. Technische Details

4.1. Glow-Shadereffekt

Der Glow-Effekt ist ein Post-Processing-Effekt der den Anschein eines leuchtenden Objektes bewirkt. Die Durchführung des Effektes ist in mehrere Schritte aufgeteilt. Dabei wird die Multi-Pass-Rendering Technik eingesetzt. Dabei werden zuerst alle Objekte die eine zusätzliche Glow-Textur haben mit dieser in eine Textur gerendert.

Die resultierende Textur wird mithilfe eines Pixelshaders separat in horizontaler und vertikaler Richtung verwischt. Die Separierung der beiden Richtungen im Gegensatz zur Berechnung in nur einem Schritt hat den Vorteil, dass es aufgrund der nicht verschachtelten Schleife schneller berechnet werden kann.

Das entstandene unscharfe Bild wird ebenfalls als Textur namens „Glow_Rendering_Textur“ gespeichert. Anschließend wird die ganze Szene mit den normalen Texturen in die Textur „Normal_Rendering_Textur“ gezeichnet. Im finalen Schritt werden die beiden Texturen „Normal_Rendering_Textur“ und „Glow_Rendering_Textur“ mit additivem Blending nacheinander gerendert.

Das additive Blending bewirkt, dass die Stellen bei den sich die beiden Texturen überschneiden, heller werden. Das Resultat ist ein leuchtend wirkendes Objekt.



Abbildung 13: Szene mit und ohne Glow-Effekt im Vergleich

4.2. Waffensystem

4.2.1. Arsenal-Klasse

Diese Klasse bildet das Herzstück des Waffensystems. Beide Spieler verfügen über eine eigene Instanz. Die wichtigsten Member der Klasse sind jeweils ein Array von Waffentyp-Factories, FragmentClouds und Munitionsmengen.

Weitere sind der Boostmanager, der Cloudmanager sowie die Prog (kurz für Programm) Struktur.

Letztere repräsentiert die aktuell ausgewählte Waffe und enthält somit die aktuelle Fabrik, Wolke, Typ, etc.

Der Boostmanager ist für die Verwaltung der verschiedenen Boosts zuständig, welche von den Fabriken produziert werden. Er steuert die Aktivierung und Deaktivierung der Effekte der Boosts, aktualisiert deren Timer und zerstört abgelaufene Boosts.

Der Cloudmanager verwaltet die Positionierung, Ausrichtung und Aktualisierung der FragmentClouds.

Außerdem verfügt das Arsenal über eine Referenz auf die Arena, da wiederum deren Referenzen auf GameScreen und den (physikalischen) Space zur Erstellung neuer Spielobjekte benötigt werden.

Die Kommunikation zwischen Spieler und Arsenal besteht zum einen Teil aus dem Weiterreichen der Input-Events von Spieler zu Arsenal. Dies schließt ein: Schuss abfeuern, Falle abfeuern, Boost "abfeuern", Rotieren der Waffentypen (in zwei Richtungen) und die Aktualisierung von Spieler Position und Ausrichtung. Bei den ersten drei Methoden wird jeweils zwischen 'Drücken und Halten', sowie 'Freigabe' der entsprechenden Taste unterschieden.

Die Kommunikation der Position erfolgt streng genommen nicht zwischen Spieler und Arsenal sondern zwischen dem Spieler und der Member des Arsenals (jeweils über Events), die da wären die Factories und der Cloudmanager.

4.2.2 Factory-System

Die Produktion der verschiedenen Waffeninstanzen im Spiel (Shot, Trap, Boost) ist nach dem AbstractFactory-Pattern organisiert (siehe GoF^[11], Seite 5).

Das Arsenal über ein Interface um von einer Factory jeweils eine Instanz produzieren bzw. abfeuern zu lassen. Die Instanz wird dabei jeweils dem grafischen und physikalischen Raum hinzugefügt bzw. dem Boostmanager übergeben.

Im Anhang befindet sich eine Abbildung einer [Waffentyp]Factory (siehe Abbildung 15).

Offensichtlich verfügt jede Fabrik über je eine Methode die vom Interface des Arsenals aufgerufen wird. Zusätzlich enthalten die meisten Fabriken eine Methode um Shots "aufzuladen", sprich, ihre Wirksamkeit zu erhöhen (grafisch dargestellt durch eine wachsende Skalierung des Geschoss-Modells).

Desweiteren abonniert jede Fabrik das Move-Event des Spielers dem sie zugewiesen ist. Anhand der Positions- und Ausrichtungsdaten des Spielers wird dementsprechend die aktuelle Shot oder Trap-Instanz (in dem Zusammenhang mit dem Run-Interface verwendet) positioniert und ausgerichtet.

Spezialfälle

Einen Spezialfall bildet die GoToFactory, welche zusätzlich das Move Event des Gegenspielers abonniert. Die entsprechenden Positionsdaten werden benötigt für Parameter für den zielverfolgenden Effekt der GoToShots.

Einen weiteren Spezialfall bildet die FirewallFactory. Sie verfügt über eine Instanz der MotorizedGrabSpring Klasse, in diesem Kontext Grabber genannt. Dieser verfügt über jeweils einen linearen- und einen winkelbasierten-Motor. Ein Motor ist ein Konstrukt gegeben in der Framework der Physik-Engine; Zweck

eines Motors ist es, eine physikalische Entität zu einer Bewegung an eine bestimmte Position oder in eine bestimmte Ausrichtung zu zwingen. Diese Funktion wird benötigt für die Verwendung der FirewallTraps und des FirewallBoosts. Die MotorizeGrabSpring Klasse wurde aus einer Demonstration der Engine entnommen und an die Zwecke des Spiels angepasst.

4.2.3. FragmentCloud

Die FragmentClouds bilden grafische Repräsentationen des Munitionsvorrats über den ein Spieler verfügt. Sie wurden designed im Zuge der Bestrebung möglichst viele Feedback bzw. GUI oder HUD Elemente in die Spielwelt zu integrieren. Ein Fragment repräsentiert hierbei ein Primitiv der Waffe (Programme genannt) der es angehört. Fragmente können um physikalische Eigenschaften bereichert in der Spielwelt plaziert werden und durch Annäherung vom Spieler aufgenommen werden.

Die Cloud Klasse verfügt schlicht über eine Liste von Fragmenten, die sie enthält und die sie aktualisiert sowie Methoden zum Hinzufügen und Entfernen von Fragmenten zu dieser Liste.

Die Animation der Fragmente ist in der Fragment Klasse selbst gekapselt und bedarf nur der Aktualisierung. Parametrisiert wird die Animation durch zufällig generierte Werte (diese steuern Radius des Bewegungskreises, Geschwindigkeit, Ausrichtung, etc.)

Die Verwaltung der Clouds erfolgt in der Cloudmanager Klasse. Diese steuert die Ausrichtung der Clouds relativ zur Spielfigur. Abhängig ist diese Ausrichtung von der Anzahl an aktiven FragmentClouds. Aktiv ist eine Wolke dann, wenn sie Fragmente enthält (sprich nicht leer ist). Die Konstellation der Clouds kann gegen und mit dem Uhrzeigersinn rotiert werden um zu erreichen dass die der aktuell ausgewählte Waffe zugeordnete Wolke sich rechts von der Spielfigur befindet. Die Methoden die diesen Vorgang initialisieren sind für das Arsenal

sichtbar und werden bei Drücken der entsprechenden Waffenwechseltaste aufgerufen.

Wenn der Fall eintritt dass eine FragmentCloud leer wird, reagiert der Manager mit der Rejustierung der Positionen der übrigen Clouds.

4.2.4. Optimierungen

Um die Framerate zu optimieren, wurde ein Frustum-Culling im Spiel implementiert. Beim Laden eines 3D-Modelles wird eine passende Bounding-Sphere erstellt. Diese wird beim Zeichnen daraufhin überprüft, ob sie im Frustum der Kamera liegt. Dadurch werden nur sichtbare Objekte an die Grafikkarte geschickt. Die Framerate hat sich durch das Frustum-Culling von 30fps auf ca. 55fps erhöht.

Desweiteren wurde das Multithreading Feature von BEPUphysics genutzt. Die Engine erkennt selbstständig, wie viele CPU-Kerne zur Verfügung stehen und nutzt dementsprechend weitere CPU-Kerne für Physik Berechnungen.

5. Ergebnisse des Projektes

5.1. Vergleich des Projektergebnisses mit dem Game Design Dokument

Die nachfolgende Tabelle dient der Gegenüberstellung der erreichten Leistungen mit den ursprünglich geplanten Features. Im Punkt 5.2. werden sowohl wichtigsten Entscheidungen im Game Design Dokument als auch Abweichungen von dieser Tabelle begründet.

Game Design Dokument	Realisierte Leistungen
Jitter als geplante Physik Engine	BEPUpysics als genutzte Physik Engine
5 verschiedene Waffen	4 Waffen implementiert; Teile der fünften Waffe mit anderen Waffen kombiniert
6 unterschiedliche Arenen	Eine Arena
Waffe 1/DELETE regeneriert den eigenen Munitionsvorrat	Keine Regeneration
Schüsse von Waffe 2/FIREWALL können beim Aufladen als Schutzschild verwendet	Idee verworfen
Waffe 3/OPTIMIZE ist Blau	Farbe auf Weiß geändert
Verlangsamungseffekt der Schüsse von Waffe 3 sind kumulativ	Effekt ist nicht kumulativ
Falle von Waffe 3 explodiert im Umkreis	Leicht abgeändert; funktioniert stattdessen ähnlich der Falle von Waffe 1
Falle von Waffe 4/GOTO ist eine Bombe; wird vom Spieler gerollt	Fehlende Animationen; Wird stattdessen wie Falle von anderen Waffen platziert
4 Bonus-Programme als Powerups	Nicht implementiert
"Quick Setup" und "Customize Game" Spielmodi	Beide Modi kombiniert; Spielregeln sind stattdessen im Optionsmenü Regelbar
4 definitiv geplante Animationen (Laufen, Springen, Schießen, Schuss aufladen)	Animationen erstellt; nicht ins Projekt integriert
Munitionspickups und zerstörbare Objekte	Im Code vorhanden, aber stark performancelastig
D-Pad des Kontrollers zum schnellen Waffenwechsel	Unnötig, da Waffenwechsel durch Schultertasten
Diverse visuelle Feedback Elemente	Nicht Implementiert
Diverse geplante Shader Effekte (Depth of Field, Motion Blur)	Nicht implementiert

Tabelle 1: Vergleich von Game Design und erreichten Projektleistungen

5.2. Begründung wichtiger Designentscheidungen

5.2.1. Interface

Durch die frühe Entscheidung, Wert auf grafische Leuchteffekte beim Charakter-Modell zu legen, wurde bereits früh im Projektlauf beschlossen, möglichst viele Teile des Interfaces am Charakter darzustellen. Für diesen Zweck eigneten sich die Lebenspunkte des Charakters sowie die derzeit aktiven Boost-Effekte besonders gut.

5.2.2. Feedback

Ähnlich dem Interface, sollte besonders visuelles Feedback von großer Bedeutung sein. Aus diesem Grund kam es auch zur Festlegung, den implementierten Waffen unterschiedliche Farben zu geben. Zum einen, um Geschosstypen während des Spielens leicht voneinander zu unterscheiden, zum anderen, um den Glow-Shader um entsprechende Farbwechsel zu erweitern.

Auch Soundeffekte waren von großer Bedeutung, da Soundeffekte für jedes Spiel von essentieller Bedeutung sind.

5.2.3. Core Mechanics

Die Entscheidung über das Genre des Spiels ist darin begründet, dass das Team bereits in der Veranstaltung "2D Game Project" einen 2D-Shooter erstellt hat und gesammelte Erfahrung in einem 3D-Projekt einbringen wollte.

Das Design des Waffen-Systems (mit aufladbaren Schüssen, Boost-Function und Trap-Function) entstand aus dem Beschluss, die Anzahl unterschiedlicher Waffen gering zu halten. Dafür sollte dem Spiel eine größere Tiefe verliehen werden, indem sich diese Waffen stark voneinander unterscheiden und vielfach einsetzbar sind.

5.2.4. Wahl der Physik-Engine

Die Anforderungen bei der Suche nach der passenden Physik-Engine waren:

- Open Source, aufgrund der Abwesenheit eines Budgets. Außerdem war maximaler Einblick und Einfluss auf den Code wünschenswert.
- Leichtgewichtigkeit. Eine zu komplexe Engine wäre schlicht unangemessen gewesen und hätte möglicherweise längere Einarbeitungszeit erfordert.
- Gute Dokumentation.

Die Kandidaten der engeren Auswahl waren Jitter Physics, BEPUphysics und Havoc. Aufgrund dieser Faktoren fiel die Wahl letztendlich auf BEPUphysics, da die Dokumentation von Jitter nicht ausreichend genug war.

5.2.5. Änderungen am Game Design

Generell lässt sich feststellen, dass ein Großteil der nicht realisierten Leistungen, z.B nicht implementierte Shader-Effekte, Animationen und Gameplay-Elemente, aufgrund mangelnder Zeit, sowie der im Punkt 3.5. beschriebenen Probleme zu erklären sind.

Im Bezug auf das Waffensystem sind zwei Designentscheidungen von größerer Bedeutung: die Änderung der Farbe von Waffe 2, sowie die Änderung der Eigenschaften von Waffe 1.

Da Waffe 3 laut Game Design ursprünglich Blau gefärbt war, können die Geschosse dieser Waffe sehr leicht mit den cyan-farbigen Geschossen von Waffe 2 verwechselt werden, was durch die Farbänderung auf Weiß umgangen wird. Die Änderungen von Waffe 1 sind einfach begründet: Ursprünglich war es geplant, dass sich der Munitionsvorrat dieser Waffe regeneriert, damit einem Spieler nie die Munition ausgeht. Diese Problem wird jedoch durch einen großen Vorrat an Startmunition behoben. Alternativ ist es möglich, die Spieldauer im Optionsmenü über ein Zeitlimit zu regeln.

6. Anhang

6.1. Referenzen

- [1] <http://jitter-physics.com/wordpress/>
- [2] <http://www.autodesk.de/>
- [3] <http://www.crazybump.com/>
- [4] 3D Graphics with XNA Game Studio 4.0, by Sean James
- [5] <http://bepu.squarespace.com/>
- [6] <http://msdn.microsoft.com/en-us/centrum-xna.aspx>
- [7] http://xbox.create.msdn.com/en-US/education/catalog/sample/skinned_model
- [8] <http://www.bfxr.net/>
- [9] <http://www.gimp.org/>
- [10] <http://alexcpeterson.com/spacescape>
- [11] *Entwurfsmuster. Elemente wiederverwendbarer objektorientierter Software*

6.2. Begleitendes Bildmaterial

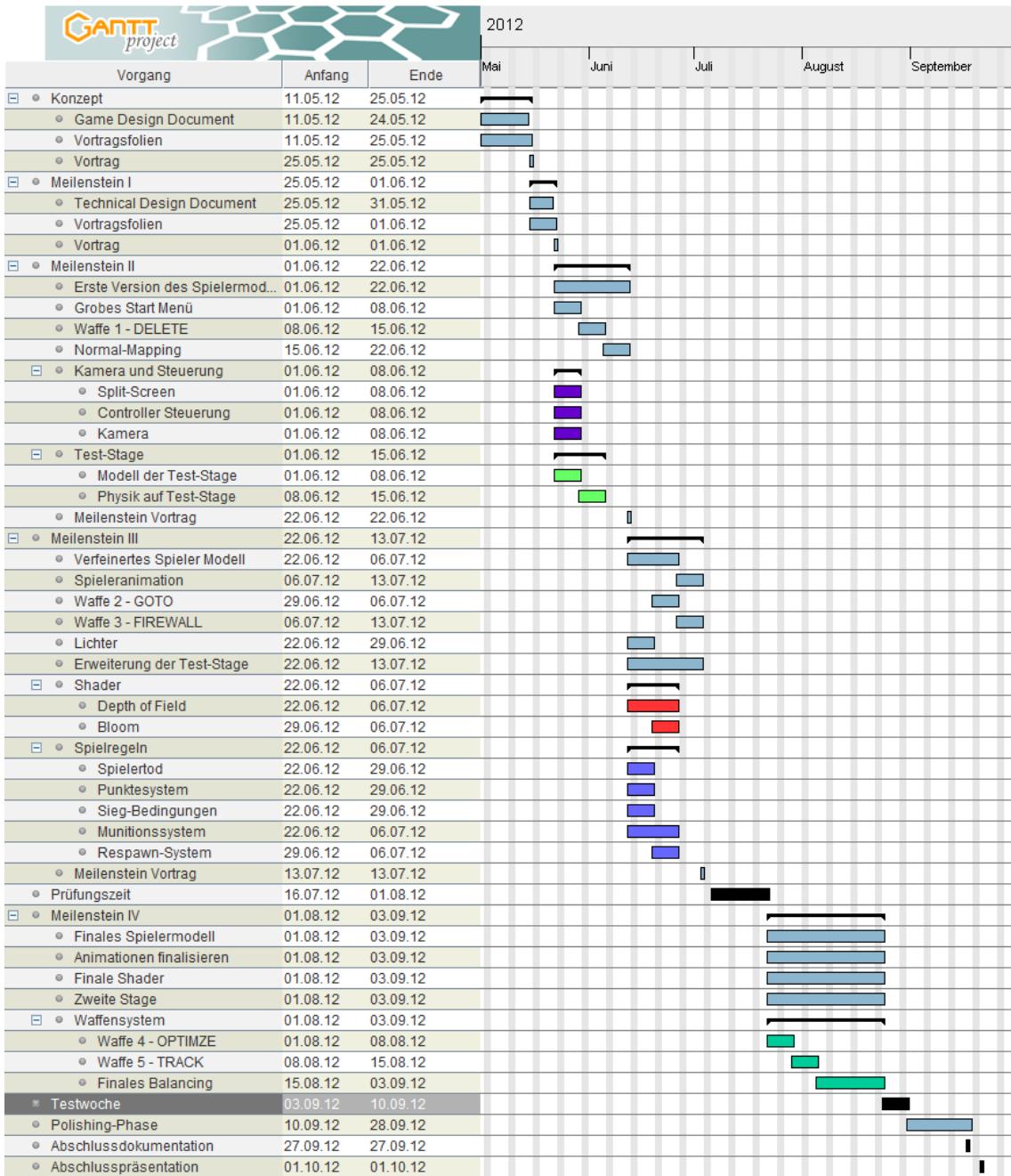


Abbildung 14: Gantt-Chart des ursprünglichen Projektplanes

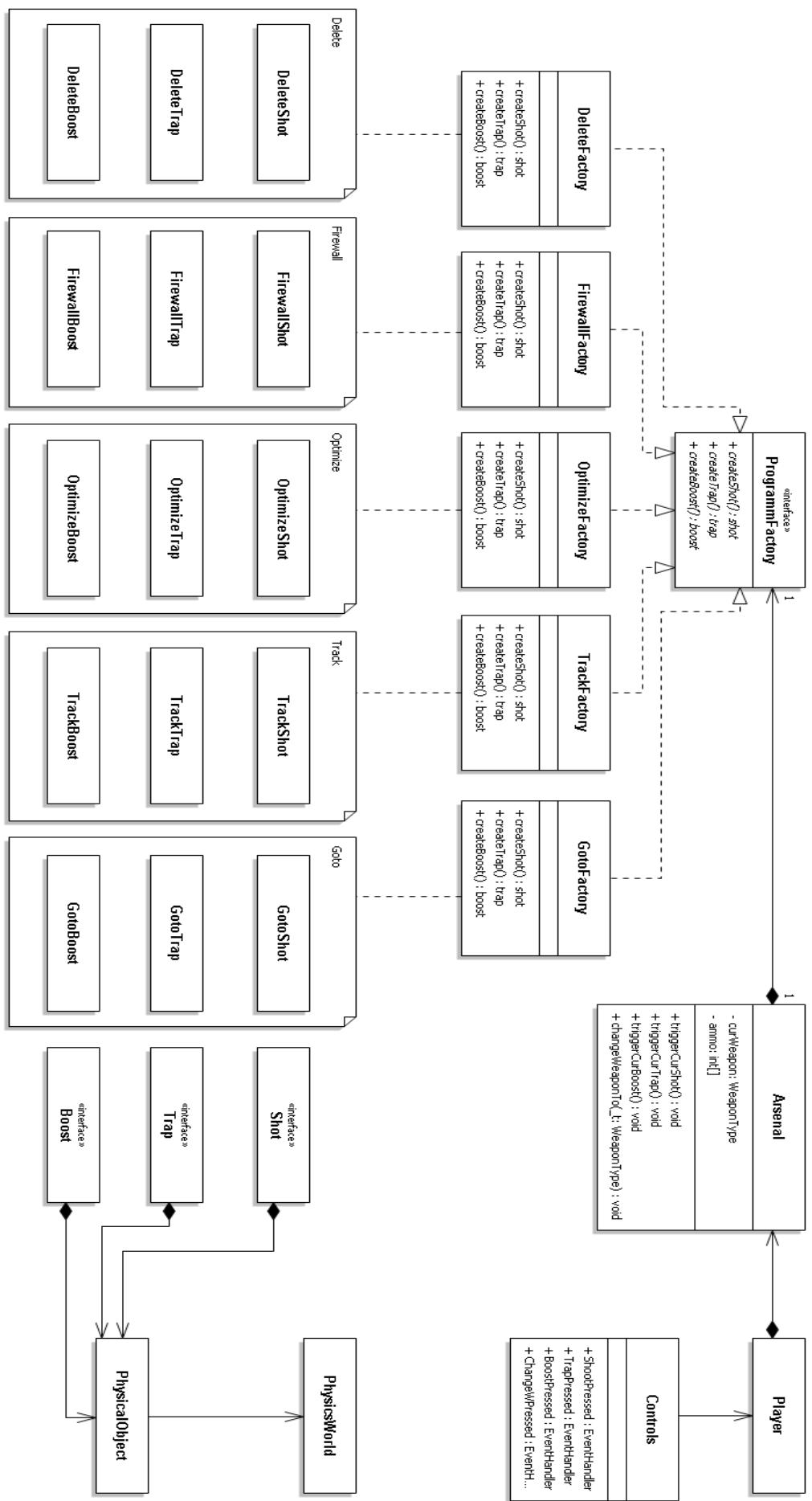


Abbildung 15: Designpattern des Waffenarsenals

6.3. Konzeptzeichnungen

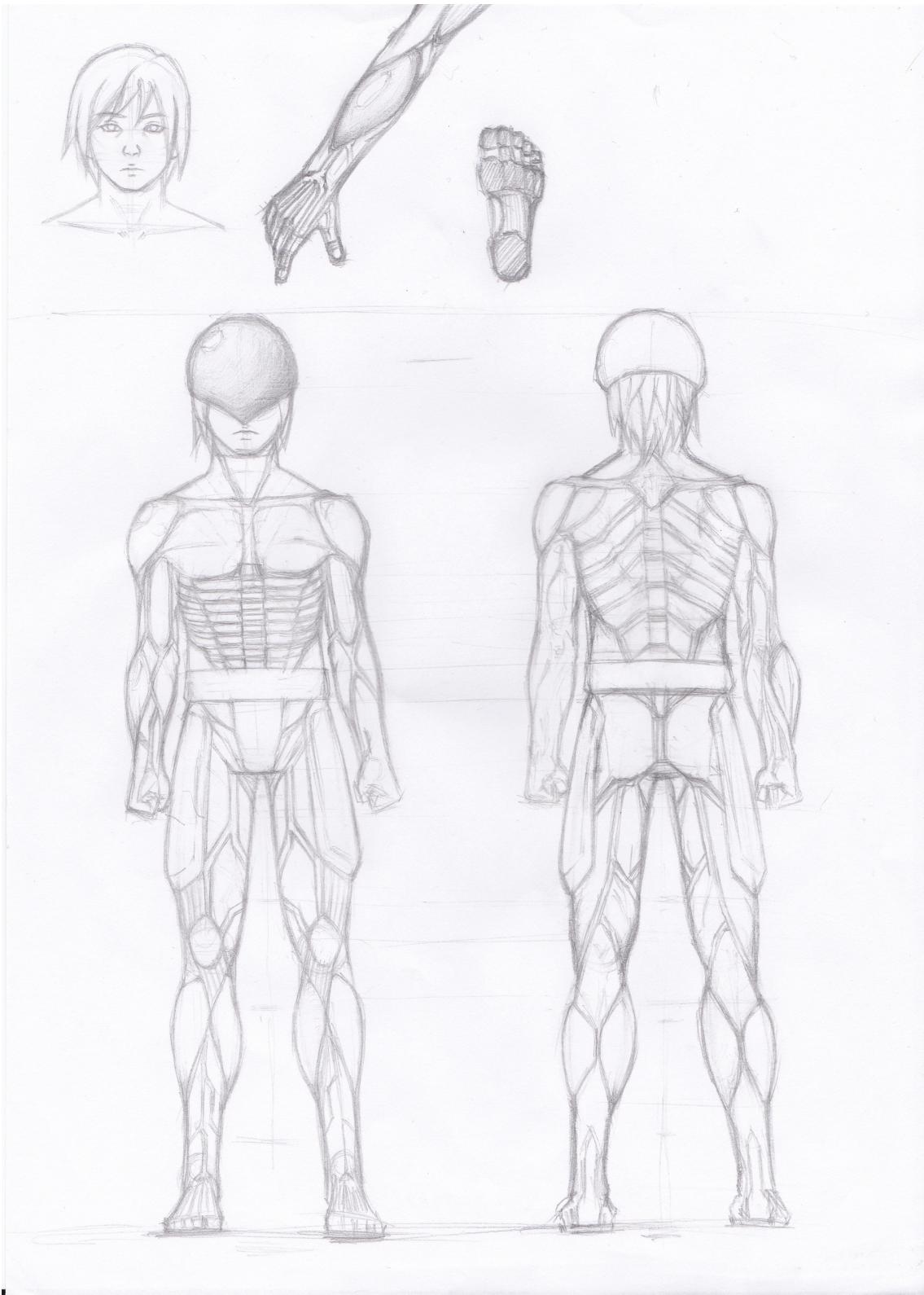


Abbildung 16: Fortgeschrittenes Charakter-Konzept

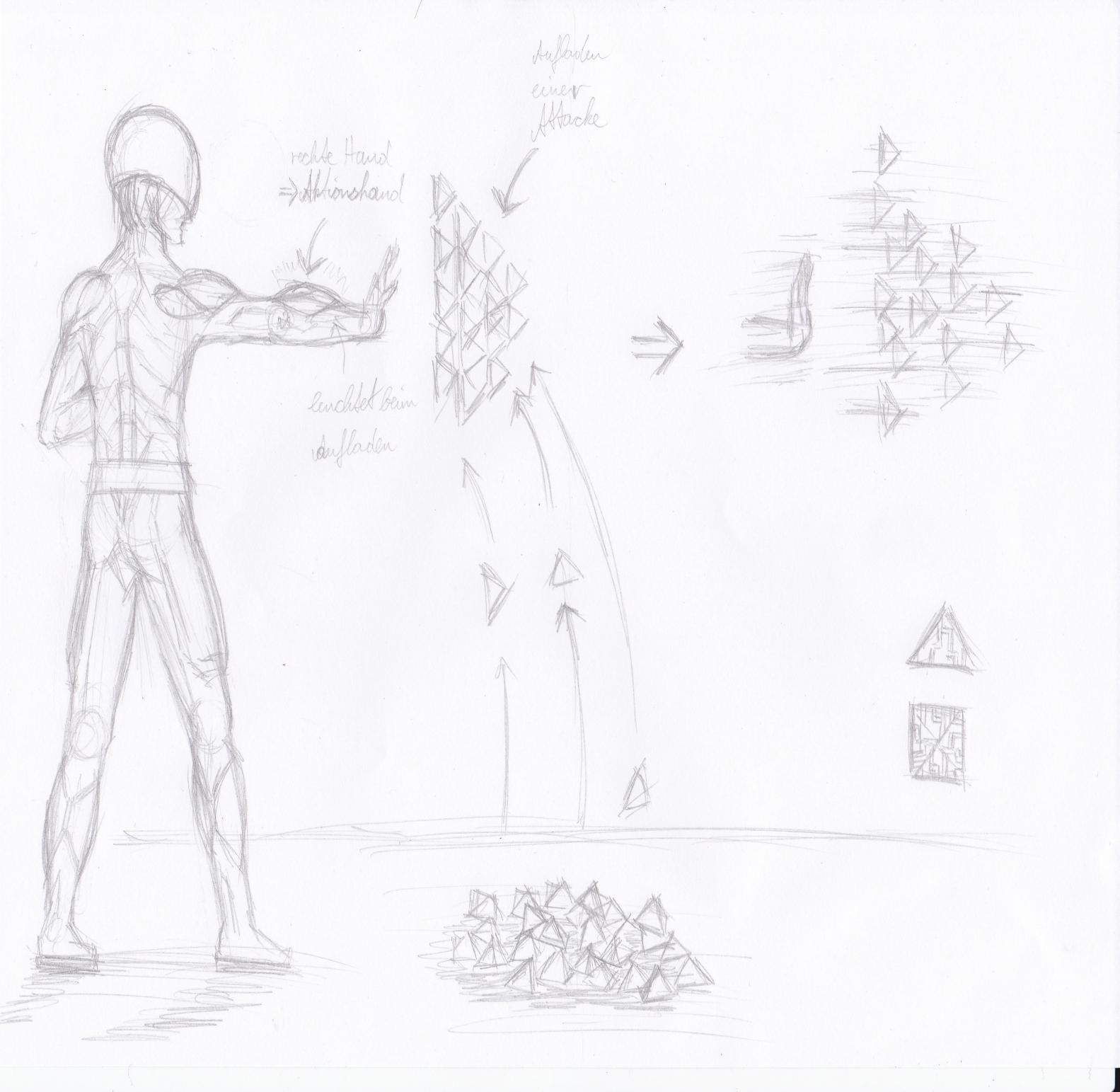


Abbildung 17: Konzept-Zeichnung einer Angriffsanimation

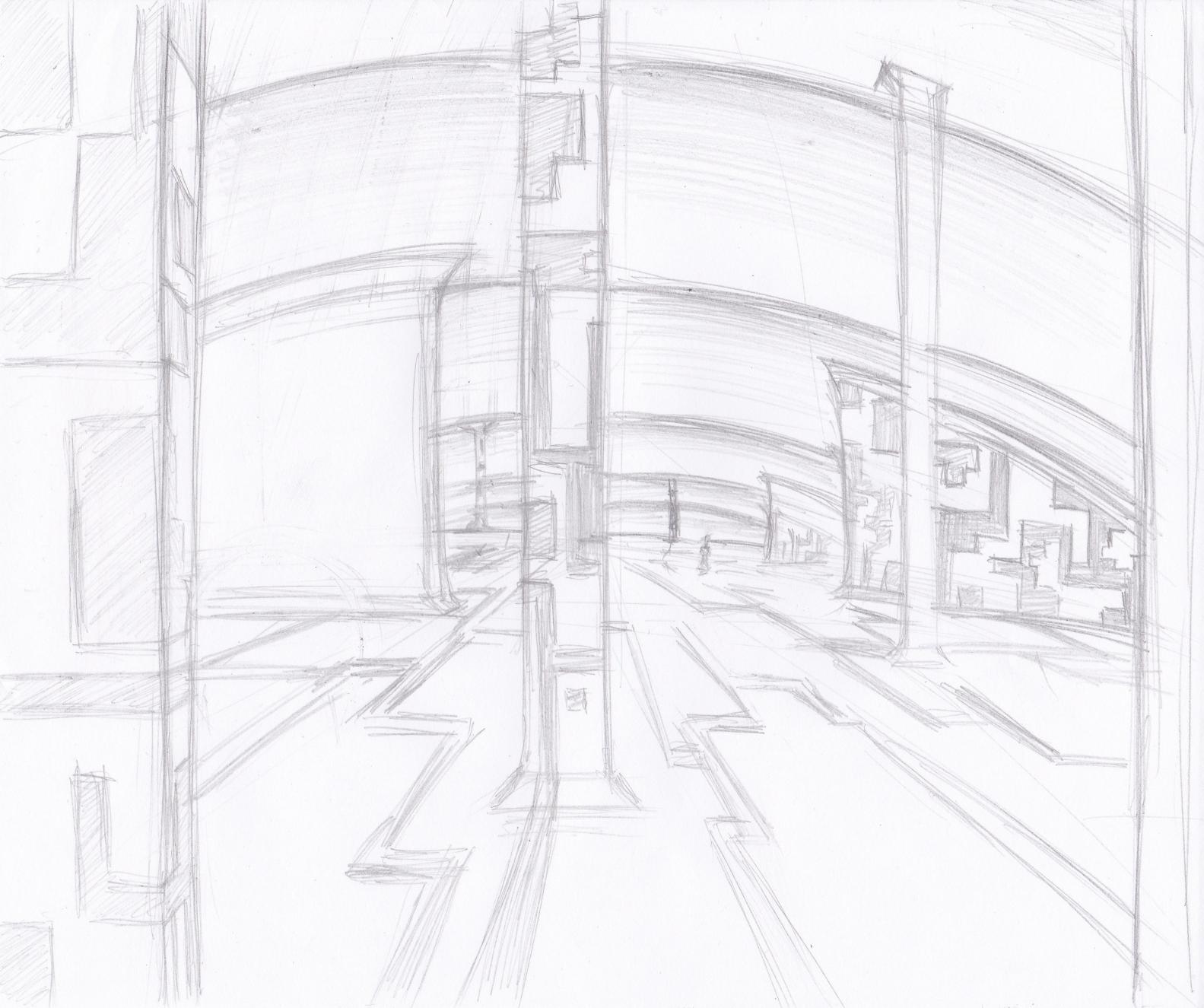


Abbildung 18: Frühes Arena-Konzept

6.4. Screenshots alter Versionen

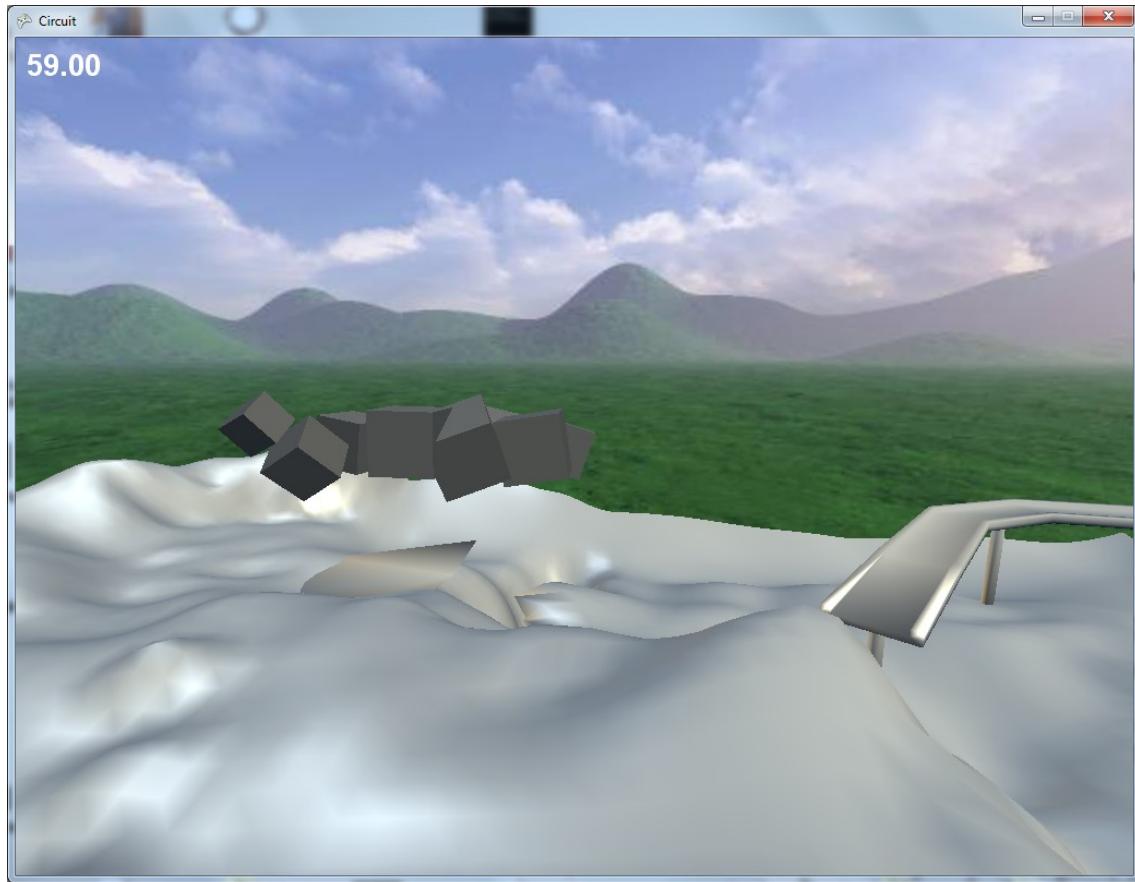


Abbildung 19: Erster lauffähiger Prototyp

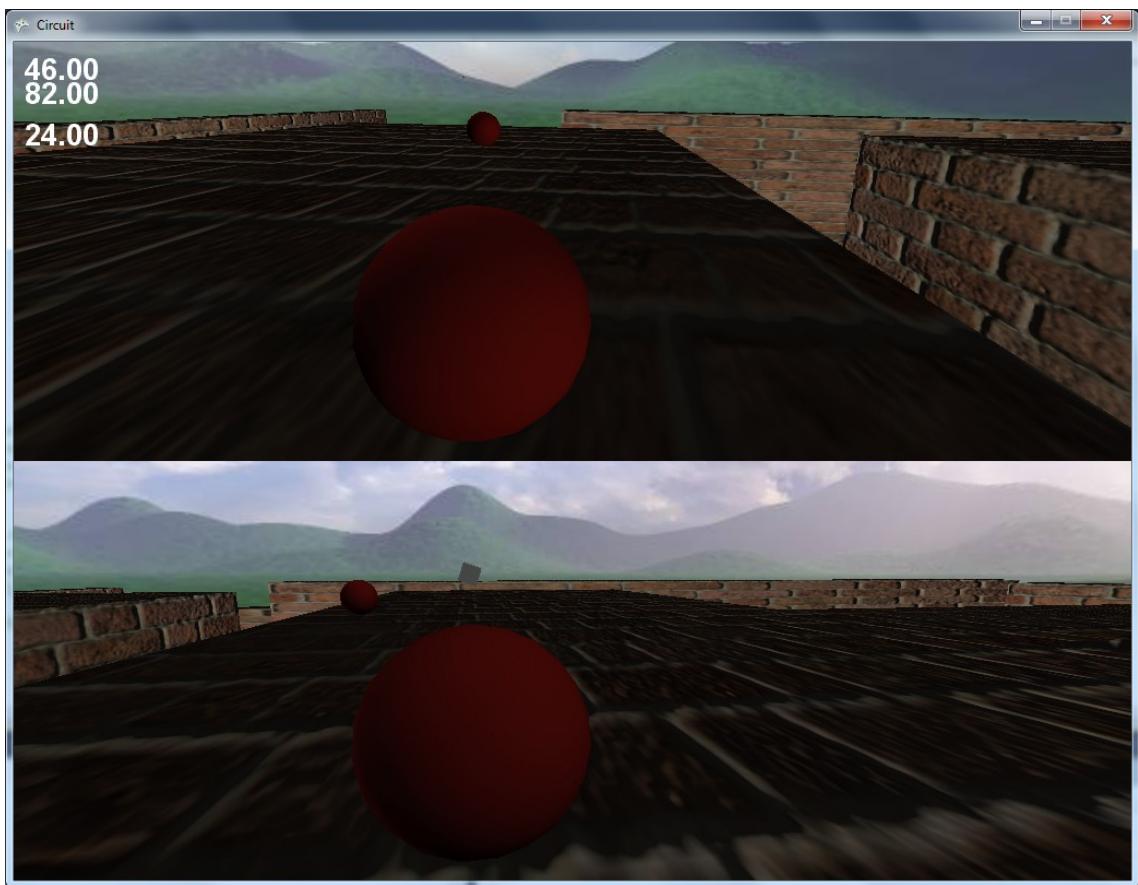


Abbildung 20: Stand zur Präsentation von Meilenstein II - 22.6.2012

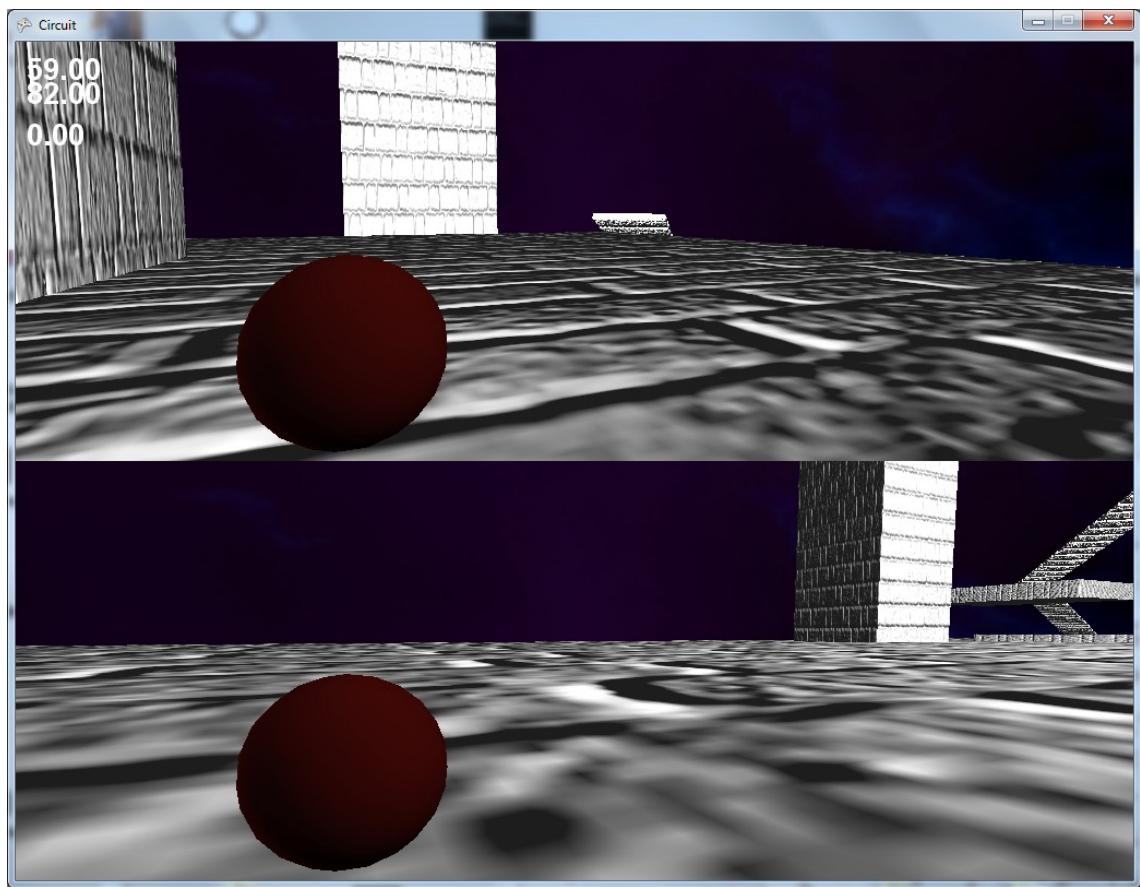


Abbildung 21: Zwischenstand für Meilenstein III mit fertiger Levelarchitektur

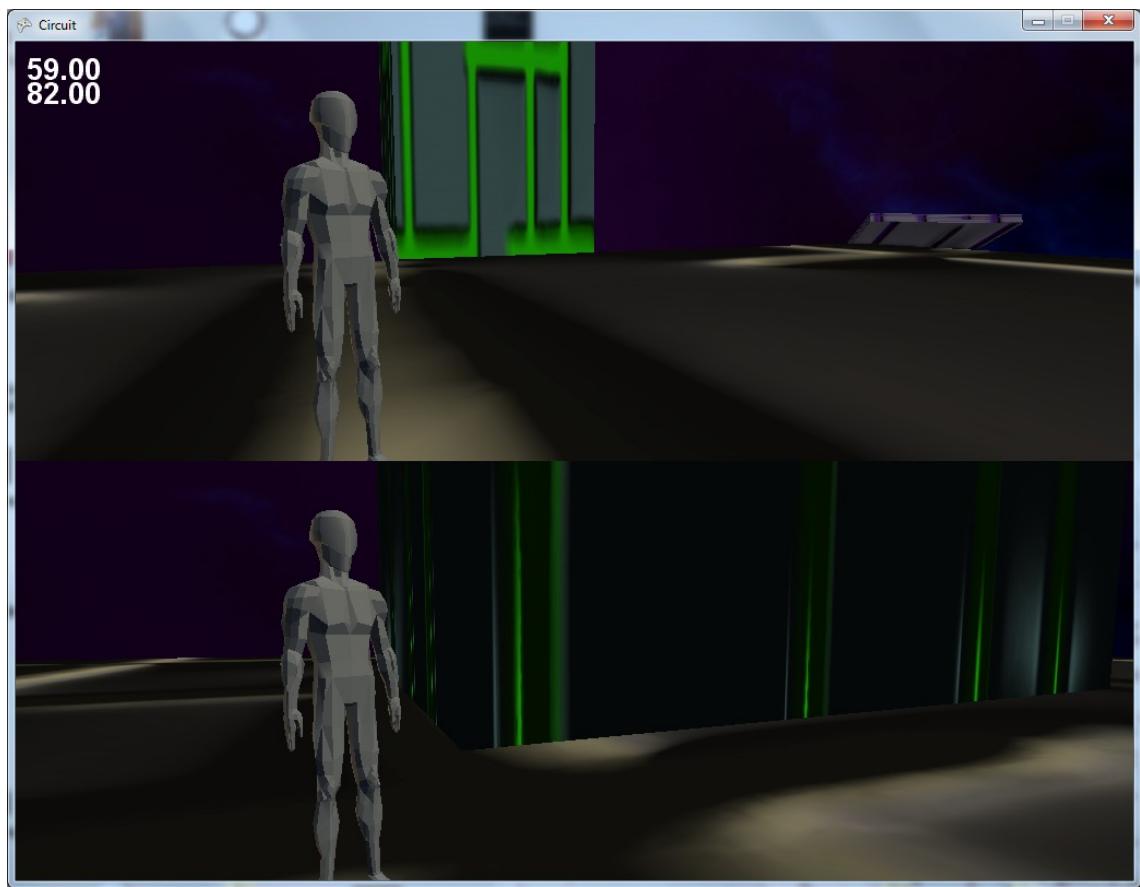


Abbildung 22: Stand zur Präsentation von Meilenstein III - 13.7.2012

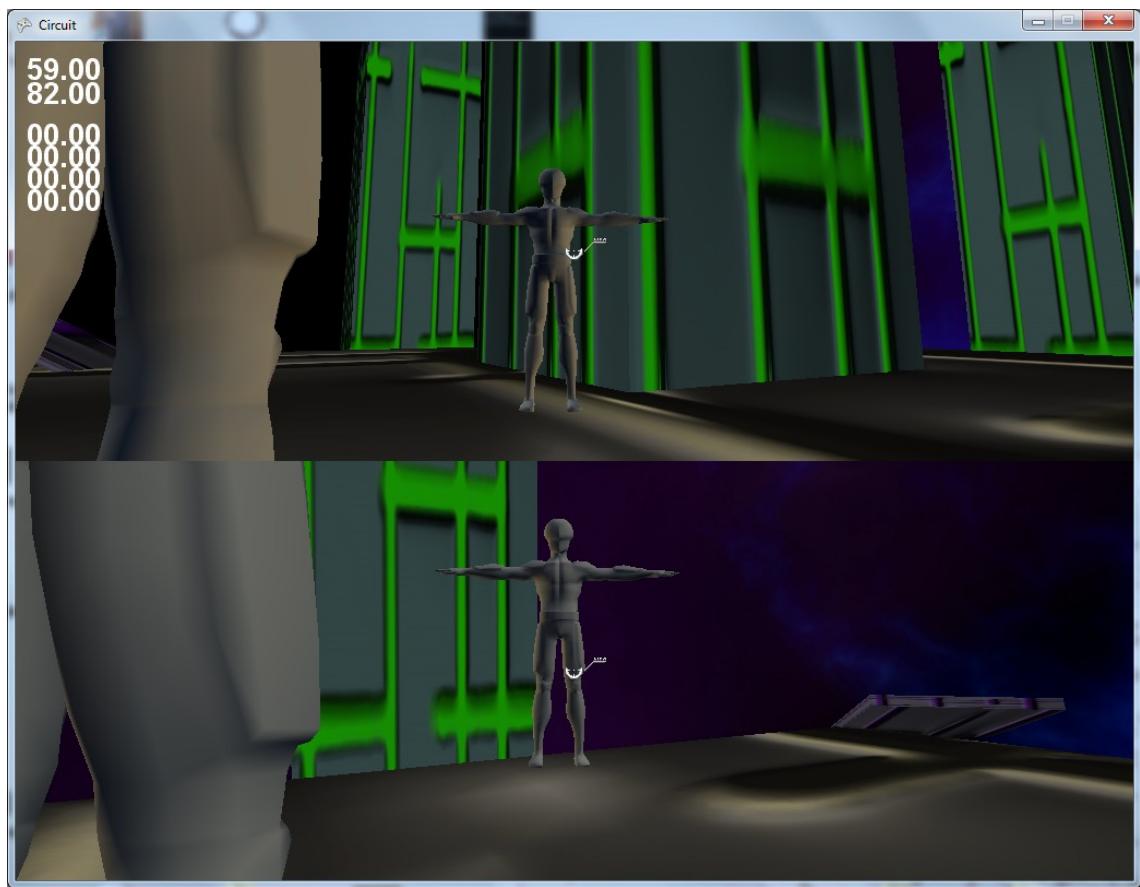


Abbildung 23: Einbindung des fertigen Charakter Modells; fehlerhafte Skalierung

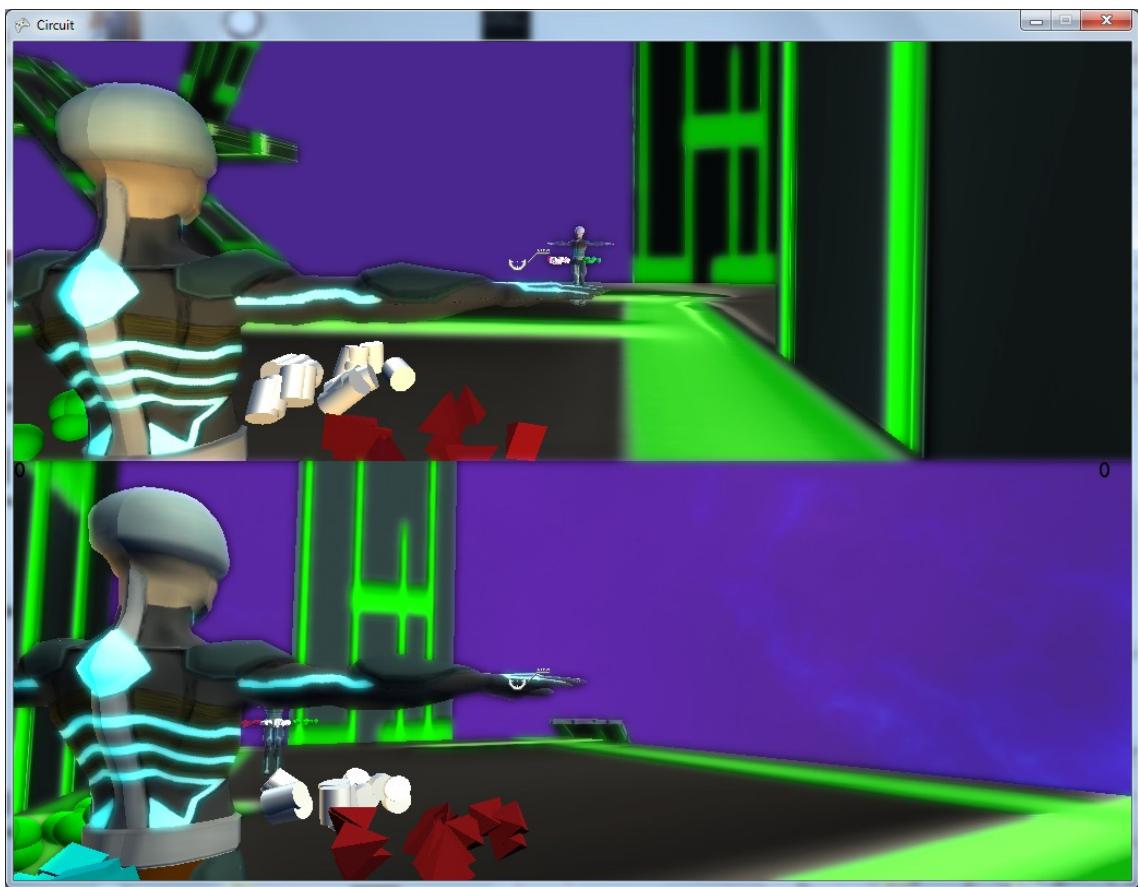


Abbildung 24: Stand zur Testwoche - 3.9.2012