Hochschule Esslingen
University of Applied Sciences

Entwicklung eines Spiels für einen Multitouch-Tisch am Beispiel von Cell Wars

Studienarbeit Sommersemester 2013

Fakultät Informationstechnik
Semester SWM6
Philipp Erler
21.07.2013

Inhalt

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

GUI: Graphical User Interface
TUIO: Tangible User Interface Objects
UDP: User Datagram Protocol

Vorwort

Die Arbeit an Cell Wars war und ist eine der interessantesten Erfahrungen meines ganzen Studiums. Die erste Idee für das Spiel hatte ich schon beim Global Game Jam 2013, konnte sie bei der Gelegenheit aber nicht umsetzen. Diese Gelegenheit bot sich mit dieser Studienarbeit.

Diese Studienarbeit baut auf früheren Studienarbeiten auf:

"Bau eines Multitouchprototypen" (08.01.2010) von Manuel Sailer (Sortiernummer: SA075)

- Physikalische Grundlagen
- Aufbau des Tisches
- Auswahl der Komponenten

"Softwareentwicklung für einen Multitouch-Tisch" (24.02.2011) von David Malek (Sortiernummer: SA74)

- Vergleich von CCV (Community Core Vision) und reacTIVision
- Erste Anwendung mit Unity 3D

"Multitouch-Anwendung" (25.07.2011) von Dominik Kimmel, Dirk Mezger und Simon Mezger (Sortiernummer: SA084)

- Verbesserung des Tisches durch Halterungen für den Umlenkspiegel und den Beamer

Für dieses Dokument wird Wissen über Programmierung, Multithreading, Objekt Orientiertes Programmieren und Containerklassen vorausgesetzt.

Ich danke:

- Patrick Tejada für Musik, Sounds und Ideen
- Den Mitarbeitern der Hochschule Esslingen Prof. Dr.-Ing. Reinhard Schmidt, Peter Dück und Fabian Müller für den reibungslosen Ablauf und für die Infrastruktur
- Meinen Kollegen bei Chasing Carrots KG für ihre Tipps für das Spiel und dessen Umsetzung
- Meinem Bruder Tim Erler fürs Testen

Für Fragen, Kritik, Verbesserungsvorschläge, oder Anderes schreiben Sie mir bitte eine E-Mail an pherti00@hs-esslingen.de.

1. Konzept

1.1. Aufgabenstellung

In einer früheren Studienarbeit wurde für die Hochschule Esslingen ein Multitouch-Tisch gebaut. Um die Möglichkeiten des Tisches zu zeigen, soll ein präsentierbares Spiel entwickelt werden.

Durch den Einsatz bei öffentlichen Veranstaltungen gibt es einige spezielle Anforderungen an das Spiel. Die Zielgruppe besteht aus potenziellen Studenten, die typischerweise um die 20 Jahre alt sind und wahrscheinlich Erfahrung mit Computerspielen haben und insbesondere Casual Games bereits kennen.

1.2. Spielidee

Cell Wars ist ein Echtzeitstrategiespiel. Die Spieler steuern Bakterien durch einen menschlichen Körper. Bakterien sammeln Nährstoffe, indem sie Körperzellen auffressen. Mit diesen Nährstoffen können sich die Bakterien durch Zellteilung vermehren oder sich weiterentwickeln. Ziel des Spiels ist, mit den Bakterien zu überleben, bis die Bakterienarmee zur Unterstützung kommt. Das Opfer wehrt sich mit Abwehrkräften (Makrophagen) gegen die Spieler.

Abbildung 1: Cell Wars Splash Screen

1.3. Anforderungen

Das fertige Spiel soll folgende Funktionalitäten beinhalten:

- Bakterien können Zellen und Makrophagen angreifen und vernichten.
- Makrophagen machen eine Bakterie kampfunfähig und verdauen sie langsam.
- Zellen geben Bakterien Ressourcen, wenn sie von ihnen angeknabbert werden.
- Bakterien können sich mit vollen Ressourcen verdoppeln oder weiterentwickeln.
- Makrophagen erscheinen in immer kürzeren Abständen in immer größerer Zahl.
- Zellen erscheinen in regelmäßigen Abständen.
- Wenn nach fünf Minuten noch eine Bakterie lebt, hat der Spieler gewonnen.
- Mehrere Bakterien können zu einer Auswahlgruppe zusammengefasst werden.
- Bakterien und Auswahlgruppen können durch Fingerbewegungen verschoben werden.
- Eine Aufgabenbeschreibung erklärt den Spielern, wie sie spielen können.
- Musik und Soundeffekte unterstützen das Spielerlebnis.
- Die Lautstärke ist einstellbar.

Das Spiel soll für öffentliche Veranstaltungen, wie den Tag der offenen Tür geeignet sein. Das heißt:

- Es darf nicht abstürzen.
- Es muss sehr viele Eingaben verarbeiten können.
- Es darf nicht bei Gesprächen stören.
- Es soll aber trotzdem auffallen.

1.4. Stil

Das Spiel ist in 2D, da die dritte Dimension für das Gameplay nicht relevant ist. Das Spiel soll organisch wirken, um zur Szenerie, dem Körper zu passen. Erreicht wird dies dadurch, dass alles bunt und in ständiger leichter Bewegung ist. Zu diesem Zweck pulsiert der Hintergrund, indem seine Größe mit einer Herzschlagkurve verändert wird. Diese Kurve ist ein Kosinus multipliziert mit der Gauss'schen Glockenkurve:

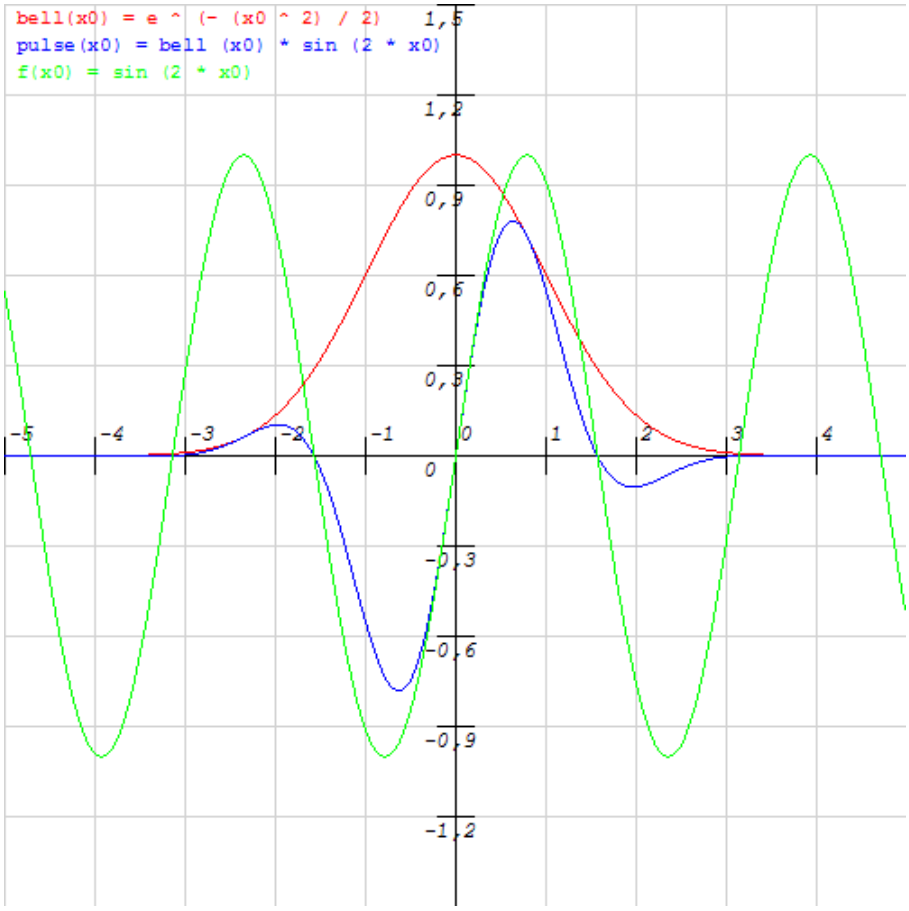

Abbildung 2: Pulsfunktion (hier Blau) = Glockenkurve * Sinus

Unterstützt wird das Organische von der Hintergrundmusik, die einen ständigen Herzschlag enthält. Außerdem erzeugt sie Spannung, indem sie immer schneller wird, nämlich von anfangs 80 bis zu 150 Schlägen pro Minute am Ende. Das Überblenden der GUI-Formulare verstärkt zusätzlich den weichen, organischen Eindruck.

Vergleichen lässt sich der Stil mit Spielen, wie Worms Armageddon[1] und World of Goo[2]. Die Bakterien und Zellen sollen als das erkennbar sein, was sie sind. Dabei müssen sie aber nicht sehr detailliert sein. Die Objekte sollen transparent sein, um realistisch zu wirken.

Da ein Körper zu einem großen Teil aus Wasser besteht, sollen sich die Objekte auch so verhalten, als ob sie durch Wasser schwimmen. Dies wird durch Dämpfung und einen Luftbläschen-Partikeleffekt bei der Bewegung erreicht.

[1] http://www.team17.com/games/worms/worms-armageddon#PC
[2] http://2dboy.com/games.php

2. Grundlagen

2.1. Multitouch-Tisch

Die primäre Plattform für das Spiel ist ein Multitouch-Tisch, der im Rahmen einer Studienarbeit für die HS-Esslingen gebaut wurde. Das Herzstück ist ein gewöhnlicher Windows-PC mit folgenden Daten:

Daten des eingebauten PCs:
Windows 7 64bit
AMD Athlon 64 X2 Dual Core Processor 4800+ (~2,5 GHz)
2GB Ram
Nvidia GeForce 7300 GT

Die grafische Ausgabe erfolgt über einen Beamer, der sein Bild gegen einen Spiegel strahlt, welcher wiederrum das Bild von unten an die Tischplatte reflektiert. Die Fingerberührungen werden von einer Kamera erkannt. Die Erkennung basiert auf Infrarotlicht, das von den Fingern auf der Tischoberfläche in eine Kamera reflektiert wird. Es wird Infrarotlicht verwendet, um die Reflexionen der Finger vom Licht des Beamers trennen zu können. Problematisch sind dabei vor allem IR-Reflexionen (Blobs), die nicht von Fingern stammen, sondern von Reflexionen an der Tischoberfläche von den Strahlern kommen. Es ist daher vorteilhaft, wenn die Tischplatte selbst das Infrarotlicht so wenig, wie möglich reflektiert. Im Moment erfolgt die Bestrahlung durch drei IR-Strahler innerhalb des Touchtable-Unterbaus, die die Tischplatte von unten beleuchten.

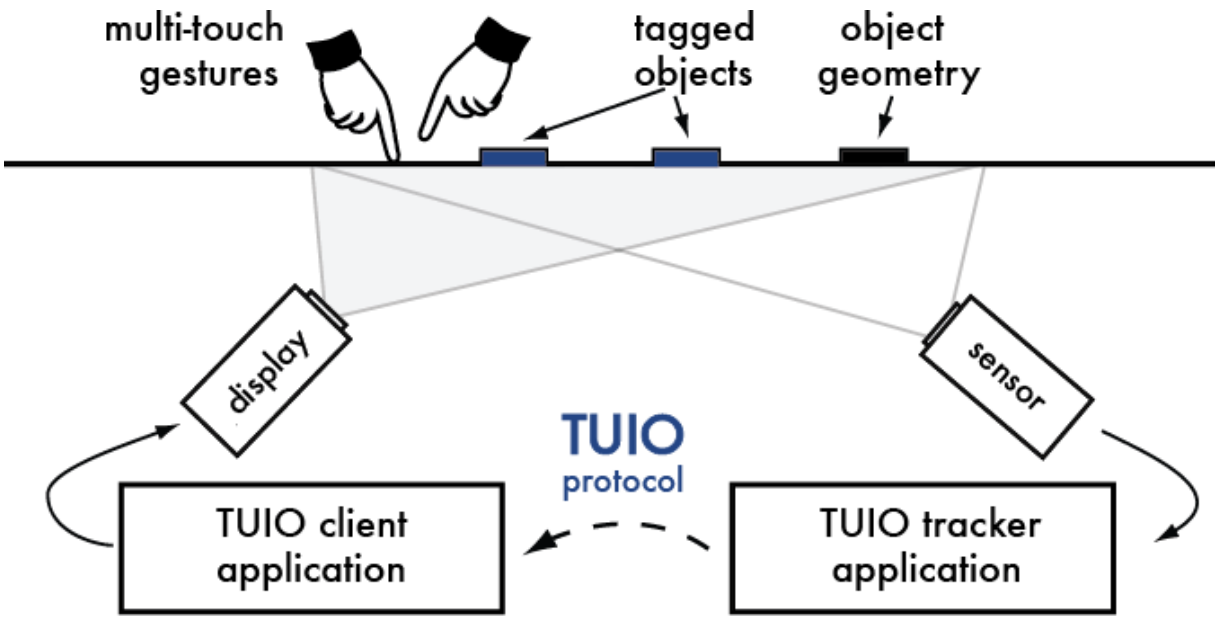

Abbildung 3: TUIO Tracking[3]

[3] Quelle: http://www.tuio.org/ (15.06.2013)

2.2. Tracking mit TUIO

Die Tracking-Software erkennt Finger und andere Objekte auf der Tischoberfläche durch Helligkeitsunterschiede auf dem Kamerabild. Ein TUIO (Tangible User Interface Objects)-Tracker enthält auch den TUIO-Server, welcher Eingabeinformationen mit dem TUIO-Protokoll über UDP versendet. Ein TUIO-Client kann diese Informationen auslesen. Er muss mit der Anwendung kommunizieren, sodass in dieser auf die Eingaben reagiert werden kann. Eine geeignete Lösung ist, den Client in die Anwendung zu integrieren, aber in einem eigenen Thread laufen zu lassen. Der Client kann dann bei Eingaben in der Anwendung Funktionen aufrufen, oder die Eingaben als Event in einer von einem Mutex geschützten Liste ablegen. Die Anwendung kann somit die Events in ihrem eigenen Takt abarbeiten. Da die UDP-Verbindung verlustbehaftet sein kann, werden Set- und Alive- anstatt Add- und Remove-Nachrichten verschickt.

Der TUIO-Server verschickt Message Bundles. Diese bestehen aus einer optionalen Source-, einer Alive- Nachricht, einer Set-Nachricht für jedes zu aktualisierende Objekt und einer Sequence-Nachricht.

Die Source-Nachricht enthält die Informationen, von welchem Programm über welche Adresse das Paket verschickt wurde. Damit können mehrere Clients die Paket herausfiltern, die ein bestimmter Server verschickt hat.

Die Alive-Nachricht enthält alle IDs von den Objekten, die im aktuellen Frame erkannt wurden. Wenn bei einer solchen Alive-Nachricht eine ID fehlt, die der Client noch in seinen Datenstrukturen gespeichert hat, kann er davon ausgehen, dass das Objekt von der Tischoberfläche gehoben wurde und damit ein Object-Up-Event auslösen.

Eine Set-Nachricht enthält alle passenden Daten zu einem TUIO-Objekt bzw. Cursor: Unter Anderem ID, Position, Geschwindigkeit und Beschleunigung. Die Anzahl und Art der Parameter hängen vom gewählten Profil ab. Das Profil bestimmt, um was für eine Art von Touchoberfläche es sich handelt. Die Sequence-Nachricht enthält einen Zähler, der mit jedem verschickten Paket hochgezählt wird. Der Client sammelt einige Daten, wie die einzelnen Objekt-Positionen, die zu einem Bewegungspfad zusammengesetzt werden, sowie die Gesamtzeit, seit der ein Objekt auf der Oberfläche erkannt wurde.

Es gibt mehrere Tracking-Softwares mit integriertem TUIO-Server. Der Vergleich von CCV (Community Core Vision) und reacTIVision von David Malek in seiner Studienarbeit "Softwareentwicklung für einen Multitouch-Tisch" auf den Seiten 12 bis 20 zeigt, dass CCV durch viele Filtereinstellungen eine bessere Fingererkennung bietet. Da das Spiel nur die Fingereingaben verwendet und die Ausleuchtung der Touch-Oberfläche mit nur drei IR-Strahlern kritisch ist, kommt nur CCV für Cell Wars in Frage.

3. Umsetzung

3.1. Projektablauf

Die Umsetzung erfolgte in mehreren Phasen:

1. Spielidee entwickeln
2. Projektplan aufstellen
3. Wahl der Game-Engine
4. Prototyping (Verknüpfung von TUIO-Client mit Spiel testen)
5. Programmieren der Grundsysteme
6. Parallele Erstellung der Assets (3D-Modelle gerendert, Grafiken und Sounds)
7. Alle Features implementieren
8. Testen

Dabei wurden folgende Programme genutzt:

- Visual Studio 2010 für die Programmierung
- Bazaar im lokalen Einsatz für Versionssteuerung
- Zim Desktop Wiki für Notizen und Ideen
- Microsoft Project für den Projektplan
- TUIO Simulator zum Testen
- 3D Studio Max für die 3D Modelle
- Photoshop und Paint.NET für Grafiken und Bearbeitung der gerenderten Modelle
- FreeFileSync zum Synchronisieren des Projekts mit USB-Sticks für den Tisch
- QuickCalculator für Visualisierung von Funktionen

3.2. Wahl der Engine

Die Engine ist das Herzstück jedes Spiels. Sie enthält alle für die Entwicklung nötigen Funktionen und Datenstrukturen. Sie beschleunigt die Entwicklung, weil sie die Abstraktionsebene von z.B. Matrix-Berechnungen zu Verschiebungen hebt. Professionelle Engines übernehmen sogar einige Aufgaben des Betriebssystems in kleinerem Umfang, wie beispielsweise die Speicherverwaltung.

Alle Vorgänger haben Unity3D benutzt. Da das Spiel die dritte Dimension nicht benötigt und 2D typischerweise einfacher und performanter ist, soll eine passendere Engine gefunden werden. Da auf diese Weise die bereits gesammelten Erfahrungen mit Unity wertlos werden, ist es wichtig, dass die neue Engine einfach zu erlernen, aber trotzdem umfangreich ist. Im Gegensatz zu 3D- sind viele 2D-Engines Open Source und können daher relativ leicht um fehlende Features erweitert werden.

Eine kurze Internetrecherche bringt zahllose mögliche Engines ans Tageslicht. Durch Empfehlungen der Kollegen und einschätzen der Kurzbeschreibungen lässt sich die Liste auf 3 Kandidaten beschränken, die genauer beurteilt werden sollen: Angel2D, IndieLib und Löve. Anhand der Engine-Websites mit beworbenen Features und Tutorials lässt sich eine Feature-Tabelle aufstellen:

Name	Angel2D	IndieLib	Löve
Website	**http://angel2d.com/**	**http://www.indielib.com/**	**https://love2d.org/**
Anzeigeeinstellungen	Ja	Ja	Ja
Objekte anzeigen, rotieren, skalieren	Ja	Ja	Ja
Animation	Ja, Spriteanimation auf Actors	Ja, Spriteanimation	Ja
Partikel	Ja	Ja	Ja
Inputsystem	Ja, Keybinding von Tastatur, Maus, Gamecontroller und Accelerometer	Ja, Tastatur, Maus	Ja, Tastatur, Maus, Gamecontroller
GUI	Ja	Nein, es gab aber Ansätze dazu	Ja
Fonts	Ja	Ja	Ja
Sound	Ja, Fmod	Nein	Ja
Kollisionserkennung	Ja	Ja, Kollisionsformen: Kreis, Rechteck, Polygon	Ja, Kollisionsformen: Kreis, Rechteck
Physik	Ja, Box2D	Nein	Ja, Box2D
Kamera	Ja	Ja, Splitscreen mit ViewPorts	Ja
Dokumentation	Ja, Super Tutorials und Beispiele, http://docs.angel2d.com/	Ja, Tutorials, Beispiele, Wiki, Forum	Ja, Super Tutorials und Beispiele
Kostenlos	Ja	Ja	Ja, Spendenbasiert
Wegfindung	Ja	Nein	Ja

Levelleditor	Kein Editor, aber Configfiles für Levels	Nein, aber es gibt Tutorial für einen In-Game Map Editor	Nein, http://www.mapeditor.org/ empfohlen im Forum
Open Source	Ja	Ja	Ja
Scripting	Ja, Lua	Nein	Ja, Lua
Konsole	Ja	Ja	Nein, im Forum aber fertig
Lizenz	BSD 3	LGPL	zlib/libpng
Standard Gameentity	Ja, Actor	Ja	Nein
Sonstiges	Messages u.A. für Eventhandling	Verzerrung von Bildern mit Surface Grids	Komplette Spiele können als .love – Datei exportiert werden (sind wie .zip)
	Unterstützt Accelerometer	Lighting	.love – Dateien können mit der Engine abgespielt werden
	Unterstützt Gamecontroller (Xbox-Controller)	Testsuite	.love – Dateien sind zip-Archive
			Multithreading
Fazit:	Wirkt einfach, aber umfangreich, optimal	Kein Sound, Tutorials nicht so gut	Sehr ausgereift, wirkt nicht so komfortabel

Tabelle 1: 2D Game Engine Vergleich

Die Wichtigkeit der Features hängen immer vom geplanten Projekt ab. So ist das Inputsystem für Cell Wars nicht wichtig, da es für TUIO sowieso nicht ausgelegt wäre und entweder extrem umgeschrieben oder ersetzt werden müsste. Ebenso werden Scripting, Physik und Künstliche Intelligenz für Cell Wars nicht benötigt. Eine Standard-Entity, von der die Spielobjekte abgeleitet werden können und eine Spielwelt, die sich um die Verwaltung der Entities kümmert, sind dagegen Features, die für das Projekt sehr wichtig sind. Diese Struktur bietet Löve nicht an. IndieLib fehlt dagegen eine eingebaute Soundunterstützung. Zum Erlernen ist zudem eine gute Dokumentation, Tutorials und eine lebendige Community sehr hilfreich und damit ein wichtiges Qualitätsmerkmal, bei dem vor allem Angel2D punktet. Zusammenfassend kann man sagen, nur Angel2D enthält alle für Cell Wars wichtigen Features und hat darüber hinaus eine überragend gute Dokumentation.

Angel2D ist auf Prototyping und Game-Jams ausgelegt, das heißt sie enthält viel Funktionalität und bleibt dabei einfach zu benutzen. Selbst der Low-Level Code, also beispielsweise das Rendern der Objekte aus internen Datenstrukturen heraus, ist verständlich geschrieben, damit er verbessert oder sogar ersetzt werden kann. Schon in der Standardversion kommt die Engine in einem fertig eingerichteten Visual Studio 2010 Projekt mit Publishing-Skript und ist auf diese Weise innerhalb von wenigen Minuten einsatzbereit. Insgesamt ist Angel2D die beste Engine für Cell Wars.

3.3. Verknüpfung des TUIO-Clients mit Angel2D

Erste Schwierigkeiten beim Einbinden des TUIO-Clients in die Engine traten schon bei den Projekteinstellungen auf. So muss der Client durch die Engine aufgerufen werden, damit er seinen eigenen Thread starten kann. Dazu muss aber der Client als statische oder dynamische Bibliothek gelinkt werden können. Dynamisches Linken, unter Windows mit DLLs (Dynamic Link Libraries), hat unter anderem die Vorteile, dass die Bibliothek bereits geladen sein kann, was die Startzeit verkürzt. Außerdem kann die eigene Anwendung diese Bibliothek mit anderen Anwendungen teilen, was Ressourcen spart. Dynamisches Linken hat auch den Vorteil, dass die Bibliothek einfach ausgetauscht werden kann, ohne die Anwendung zu ändern. Auf diese Weise kann allerdings auch eine inkompatible Bibliothek die Anwendung lahmlegen. Statisches Linken bietet den großen Vorteil, dass der Code beim Kompilieren vollständig bekannt ist, die Kontrolle über das ganze System also beim Entwickler bleibt. Da die Anzahl der Programme, die eine TUIO-Client-DLL benutzen doch sehr gering ist, bietet dynamisches Linken für Cell Wars keine Vorteile. Daher wird das statische Linken verwendet. Dafür muss im Visual Studio Projekt die richtige Einstellung gesetzt werden, und die TUIO-Client-Bibliothek als Linker Input eingetragen sein. Da der TUIO-Client seinerseits statisch in die TUIO-Bibliothek linkt, muss auch diese im Cell Wars Projekt als Linker Input stehen.

Normalerweise ist die Verknüpfung von zwei eher zusammenhanglosen Bibliotheken eine deutlich aufwändigere Aufgabe als die Projekteinstellung. Im Fall des C++ TUIO-Clients mit Angel2D ist dies jedoch nicht so schlimm. Das liegt vor allem daran, dass beide Programme mit vollständig offenem C++ - Quellcode vorhanden sind und Angel2D bereits einen Multitouchmanager beinhaltet. Dieser verwaltet Berührungseingaben und berührte Spielobjekte, jedoch für die Verwendung mit TUIO stark geändert oder sogar ersetzt werden. Ein Grund dafür ist auch, dass beide Bibliotheken eigene Datenstrukturen für bestimmte Informationen, wie beispielsweise Vektoren, besitzen. Dies führt dazu, dass in jedem Frame für alle TUIO-Cursor die Pfade, also eine Liste von Positionsvektoren konvertiert werden müssen, was einen beachtlichen Rechenaufwand bedeutet. An dieser Stelle könnte Cell Wars optimiert werden.

Die erste Version des Multitouchmanagers sah vor, für die drei wichtigen TUIO-Events, also CursorDown, CursorMove und CursorUp jeweils eine Liste zu verwalten. Diese Listen sind durch Mutexe geschützt, die ermöglichen, dass der TUIO-Client aus seinem Thread Events zu den Listen hinzufügen und die Spiele-Engine in der Update-Phase die Events an die entsprechenden Objekte weiterleiten kann, ohne dass es dabei zu Inkonsistenz kommt. Die drei Listen wurden von der Engine nacheinander abgearbeitet. Später hat sich herausgestellt, dass bei schnellen und vielen Fingerbewegungen eine CursorID mehrfach auftreten kann. Dies hatte die Folge, dass die Engine beispielsweise erst zwei FingerUp-Events zu einem Objekt weitergeleitet hat und das FingerDown-Event in der zweiten Liste auf ein anderes Objekt erst danach ausgewertet wurde. Dadurch kam es nicht nur zu falschen Zuständen der Objekte, sondern auch zu Sprüngen der Cursor, sowie fehlgeleiteten Events. Die Lösung dafür ist, dass alle Events genau in der Reihenfolge, in der sie aufgetreten sind, auch ausgewertet werden müssen. Eine aufwändige Implementierung wäre, die Events nach einem Timestamp sortiert auszuwerten. Die beste Möglichkeit ist aber, alle Events in einer Queue zu speichern. Dabei muss beachtet werden, dass die Art des Events, am besten als Aufzählung, auch in den Queue-Einträgen gespeichert werden muss. Bei der Weiterleitung der Events ist dann eine Fallunterscheidung nötig.

3.4. Koordinatensysteme

In Cell Wars spielen drei verschiedene Koordinatensysteme eine Rolle.

Das erste ist die Basis des TUIO-Trackers. Es hat in der Standardeinstellung den Ursprung links oben in der Ecke der getrackten Fläche. Die X- und Y-Werte gehen von Null bis Eins. Eine TUIO-Koordinate (1|1) ist also die rechte untere Ecke. Dabei ist zu beachten, dass eine Längeneinheit auf der X-Achse nicht gleich lang wie auf der Y-Achse ist.

Das zweite ist das Welt-Koordinatensystem und wird von Angel für die Positionen von gewöhnlichen Spielobjekten verwendet. In Cell Wars sind solche Objekte beispielsweise Bakterien und Zellen. Das Angel Koordinatensystem hat standardmäßig den Ursprung in der Mitte des Bildschirms. Die X-Achse geht nach rechts und die Y-Achse noch oben. Die X- und Y-Längeneinheiten bleiben dabei immer gleich groß. Angel simuliert eine Kamera, die senkrecht von Oben auf die X-Y-Fläche gerichtet ist. Durch Heben und Senken wird das Koordinatensystem gleichförmig skaliert und durch Längsbewegungen wird der Ursprung verschoben.

Angel2D verwendet das dritte Koordinatensystem für Objekte, die direkt und pixelgenau auf dem Bildschirm unabhängig von der Kamera angezeigt werden sollen. Zu diesen Objekten gehören auch in Cell Wars GUI-Elemente wie Menüs, Texte und Buttons. Dieses Koordinatensystem ist das Übliche für Bildschirmausgaben, das heißt dass der Ursprung links oben ist, die X-Achse geht nach rechts und die Y-Achse nach unten. Die Längeneinheiten sind in X- und Y-Richtung normalerweise gleich groß, nämlich die Pixellänge bzw. -breite. Die Maximalwerte für X und Y sind von der Auflösung abhängig. Bei Cell Wars heißt das, dass durch den Full-HD-Beamer im Tisch X von 0 bis 1920 und Y von 0 bis 1080 geht.

Es wird eine Konvertierungsfunktion für Punkte von jedem Koordinatensystem in jedes andere benötigt. Da das zweite und dritte Koordinatensystem Teile von Angel2D sind gibt es in der MathUtil-Klasse bereits Funktionen dafür. Die anderen Konvertierungsfunktionen müssen selbst geschrieben werden. Theoretisch könnte der TUIO-Tracker auch passend zum Angel-Welt-Koordinatensystem eingestellt werden, allerdings kann man sich bei Fremdsystemen nicht auf diese Einstellungen verlassen. Außerdem müsste der TUIO-Tracker bei jeder Kameraänderung angepasst werden. Um die verschiedenen Koordinaten zusammenzubringen, ist es am besten, die Tracking-Punkte an der Schnittstelle zur Spieleengine zu konvertieren. Für Cell Wars bedeutet dies, dass die TUIO-Punkte sofort bei der Übergabe als Event-Parameter an den Inputmanager konvertiert werden. Die Punkte können dann für das Spiel als Angel-Punkte gespeichert und weiter verwendet werden.

3.5. Klassen und Spielobjekte

Bei der Software-Architektur von Cell Wars wird intensiver Gebrauch von Vererbung gemacht. Auf diese Weise lässt sich der Code sehr gut in zusammengehörige Teile aufspalten. Beispielsweise erben die *Bacterium*- und die *Button*-Klassen von der abstrakten *Touchable*-Klasse. Dadurch muss im Code der Bakterien nicht mehr darauf geachtet werden, dass sie sich beim *MultiTouchManager* anmelden. Ähnliche Strukturen existieren auch um die *MobileMicrobe*- und die *Damageable*-Klassen.

Anfangs wurde überlegt, ob der *MultiTouchManager* nicht als Singleton implementiert werden sollten. Dieses Design-Pattern eignet sich allerdings nicht, da es nicht nur zu unsauberem Code führt, wie beispielsweise zu versteckten Abhängigkeiten (siehe Jim09[4]), sondern auch inkompatibel mit Vererbung ist. Stattdessen sind die Manager ganz normale Klassen. In Angel2D sind dennoch einige Klassen wie die Kamera oder der Soundmanager als Singleton implementiert. Dies ist normalerweise unproblematisch, da solche Klassen nicht abgeleitet werden müssen. Von den selbst entwickelten Klassen ist nur der *CellWarsGameManager* ein Singleton.

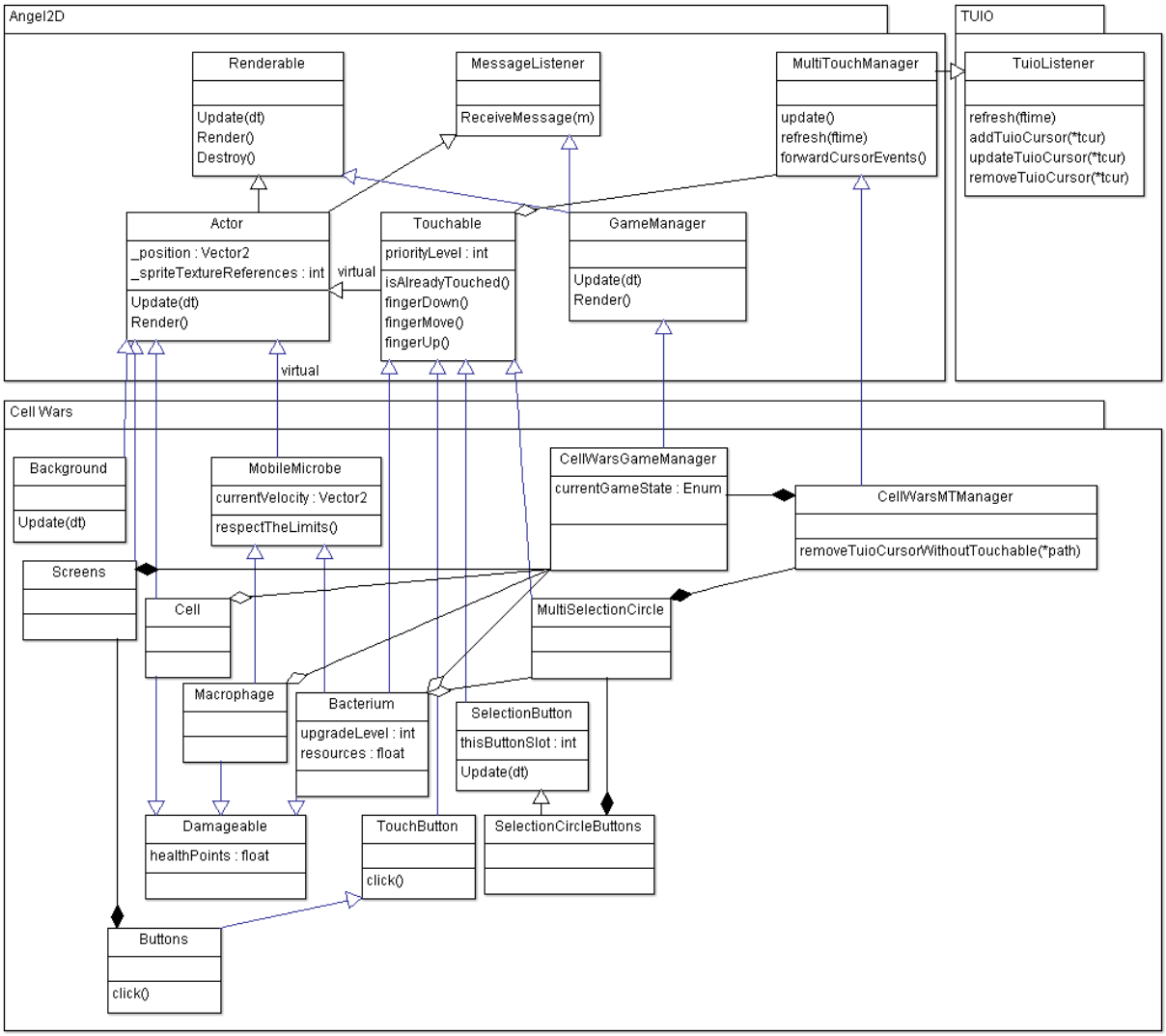

Abbildung 4: Teil des Klassendiagramms

[4] http://stackoverflow.com/questions/137975/what-is-so-bad-about-singletons

Die *Bacterium*-Klasse erbt von *Touchable* und *MobileMicrobe*, welche jeweils von der *Actor*-Klasse erben. An dieser Stelle entsteht das Deadly-Diamond-Problem. Es basiert darauf, dass die Daten der *Actor*-Klasse durch die Mehrfachvererbung doppelt im *Bacterium* vorkommen und bei einem Zugriff darauf nicht klar ist, welche Version davon gemeint ist. Lösen lässt sich dieses Problem z.B. dadurch, dass eine Klasse nicht von *Actor* erbt, sondern eine Instanz davon enthält. In Cell Wars hat sich dieser Ansatz nicht durchgesetzt, da er zu anderen Problemen und höherem Aufwand geführt hat. Stattdessen wird das Deadly-Diamond-Problem mit virtueller Vererbung gelöst. Das heißt, eine Klasse enthält nur eine, nämlich die erste Version von vererbten Daten, auch wenn sie durch Mehrfachvererbung mehrmals vorkommen müssten. Virtuelle Vererbung führt allerdings auch zu mehr Overhead und sollte daher mit Bedacht eingesetzt werden.

In Angel2D ist die World-Klasse das Zentrum der Klassenstruktur. Ihr wird der Game Manager des Spiels zugewiesen, der seine Spielobjekte, also alle von der *Renderable*-Klasse abgeleiteten Objekte, also *Actors* wie z.B. Bakterien zur Verwaltung übergibt. Die World enthält die Mainloop des Spiels, in welcher alle *Renderables* aktualisiert und gerendert werden. Dazu muss sie die Zeit seit dem letzten Frame, DeltaTime genannt, den Update-Methoden der *Renderables* als Parameter übergeben werden. Die Spielobjekte können damit ihren Zustand, also z.B. das angezeigte Bild einer Sprite-Animation anzeigen. Am Ende des Frames werden die Objekte in ihrem aktuellen Zustand an ihrer aktuellen Position gezeichnet. Die Spielwelt steuert nicht nur das Aktualisieren der Spielobjekte, sondern löscht sie auch, wenn beispielsweise das Level gewechselt oder das Spiel beendet wird. Da in der Update-Phase Spielobjekte eventuell auf alle anderen Zugreifen wollen, müssen in dieser Phase alle Objekte existieren. Damit dennoch Objekte gelöscht werden können, weil sie beispielsweise aufgefressene Zellen sind, können sie als zerstört markiert werden. Dabei wird deren Destroyed-Methode aufgerufen, sodass sie ihre Daten aufräumen können. Beispielsweise wird in dieser-Methode von Bakterien deren *ParticleActors* als zerstört markiert. Die ganzen markierten Objekte werden von der World nach dem Update aber vor dem Rendern endgültig gelöscht. Diese Struktur ist typisch für die meisten Spiele-Engines.

Die exzessive Nutzung von Vererbung funktioniert in Cell Wars sehr gut. Sie sorgt dafür, dass einzelne Funktionalitäten nur einmal implementiert und damit an nur einer Stelle konzentriert sind. Dadurch bleibt die Gesamtkomplexität begrenzt. In größeren Projekten, ist es möglicherweise nötig, die Abhängigkeiten zwischen den Klassen flexibler zu gestalten. Wie beispielsweise Steve Rowe in seinem Blog[5] schreibt, sollte Komposition der Vererbung meist vorgezogen werden.

[5] http://blogs.msdn.com/b/steverowe/archive/2008/04/28/prefer-composition-over-inheritance.aspx

3.6. Grafische Nutzeroberfläche

Angel2D beinhaltet bereits ein GUI-System. Da dieses allerdings für die Bedürfnisse von Cell Wars überdimensioniert sein dürfte und die Anbindung an die Multitouch-Events aufwändig ist, wurden für das Spiel die nötigen Teile selbst entwickelt. Damit sich die GUI vom Hintergrund abhebt, ist ein Hintergrund für die Formulare nötig. Dieser ist teilweise transparent, um nicht das ganze Spielfeld zu verdecken, was bei der Bakterieninvasion nach einem Sieg schade wäre. Es werden die bereits in Angel vorhandenen Textactors für die Textanzeige verwendet. Zur Anzeige von statischen Bildern werden die Standard-Actors verwendet. Damit die Spieler mit der GUI interagieren können, sind Buttons entwickelt worden. Diese Buttons sind wie die Bakterien von Touchable abgeleitet. Im Gegensatz zu den Bakterien bewegen sie sich aber nicht, sondern rufen beim CursorDown-Event die Click-Methode auf. Die Buttons müssen beim Click unterschiedliche Funktionen ausführen, wie das Erhöhen der Lautstärke, einen Wechsel des Spielzustands (und damit des Formulars), sowie das Beenden des Spiels. Dazu hat die TouchButton-Klasse die rein virtuelle Click-Methode, die von den abgeleiteten Buttons implementiert wird. So bleibt das Event-zu-Click interpretieren an einer Stelle und die aufgerufenen Funktionalitäten können beliebig verschieden sein. In der aktuellen Implementierung erzeugt der CellWarsGameManager das Formular (Screen), das zu seinem aktuellen Zustand gehört. Das Formular erzeugt seinerseits die dazugehörigen Steuerelemente und setzt deren Eigenschaften, z.B. Position, Bild, Größe und Text entsprechend. Wenn das Formular gelöscht wird, löscht es seine Steuerelemente mit.

Anfangs wurde ein Button-Klick-Event so implementiert, wie es auch bei Mausklicks auf Buttons ist. Also wird das Klick-Event erst ausgelöst, wenn das Down-Event auf dem Button ausgelöst wurde, der Cursor über dem Button bleibt und am Ende das Up-Event ausgelöst wurde. Wie auch bei dem Mausäquivalent sollte der Button nach dem Finger-Down-Event eine gedrückte Version anzeigen. Wenn diese Version fehlt, wird die fehlende Reaktion von den Spielern als Inaktivität interpretiert. Als Alternative für verschiedene Button-Zustände kann das Klick-Event auch schon beim Finger-Down ausgelöst werden. Dies hat auch den Vorteil, dass gerade in hektischen Situationen die Buttons leichter gedrückt werden können. Der Nachteil dabei ist, dass ein versehentliches Drücken eines Buttons nicht abgebrochen werden kann.

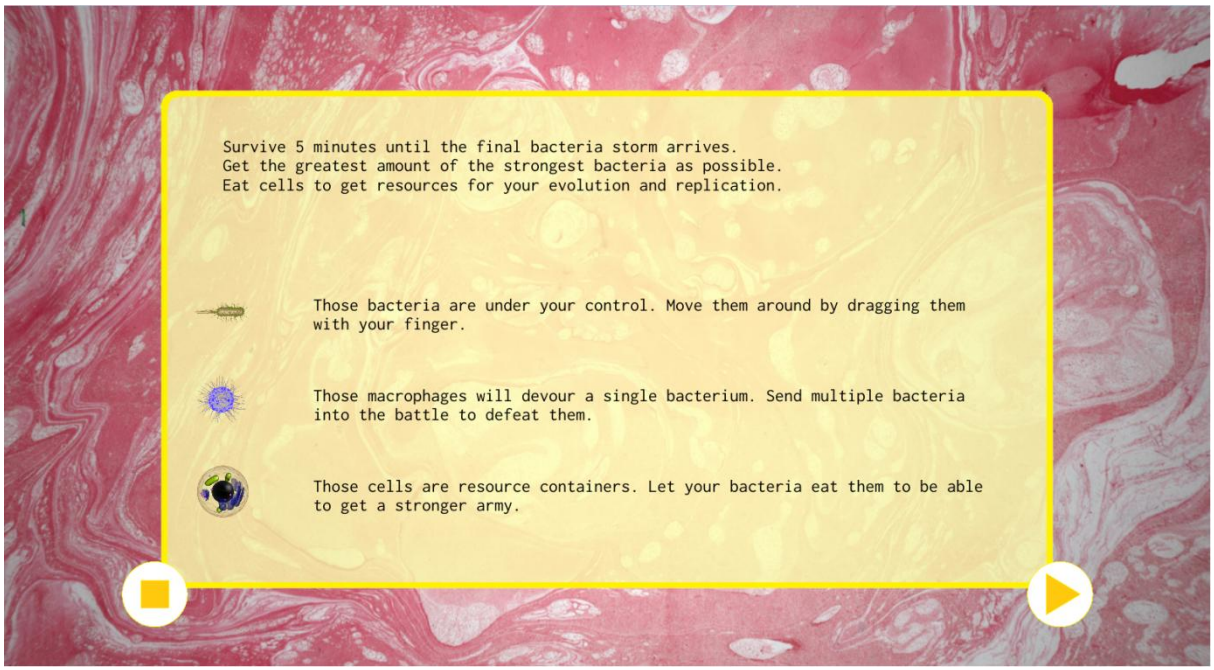

Abbildung 5: Hilfeseite als GUI-Beispiel mit Texten, Bildern und Buttons

Ein wichtiges Ziel aller GUI-Systeme ist, auf allen unterstützten Geräten die gleiche oder wenigstens eine gute GUI darzustellen. In Cell Wars wird die GUI mit der Auflösung skaliert. Das funktioniert allerdings nicht bei den Textactors, deren Font eine feste Größe hat und in festen Pixelangaben Positioniert werden. Die Positionen lassen sich umrechnen (siehe 3.4 Koordinatensysteme), aber die einzelnen Zeichen behalten ihre Größe. Die Fontgröße lässt sich nicht zuverlässig skalieren, da Schriftarten nicht in allen Größen verfügbar sind. In Cell Wars ist das Problem mit den Texten nur teilweise gelöst. Die Zeilenumbrüche innerhalb der Textactors werden dynamisch über die Auflösung, Fontgröße und TextActor-Breite berechnet. Es bleiben aber möglicherweise unpassend große Abstände zwischen den Steuerelementen.

3.7. Erstellung der Assets

Das beste Spiel nützt nichts, wenn es nichts zu sehen und zu hören gibt. Deshalb müssen für Cell Wars wie für jedes andere Spiel auch Assets erstellt werden. Die Assets sind 3D-Modelle, Bilder, Musik und Sounds. Da Cell Wars ein 2D-Spiel ist, können die 3D-Modelle aus 3D Studio Max heraus gerendert und als Bilder im Spiel verwendet werden.

Bei den Modellen ist es wichtig, die in Kapitel 1.4 Stil angesprochenen Eigenheiten zu beachten: Organisches, transparentes und nicht zu detailliertes Aussehen. Die zu erstellenden Modelle sind Zelle, Bakterie und Makrophage.

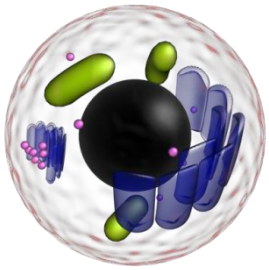

Abbildung 6: Zelle

Die Zelle soll an die Tierischen Zellen erinnern, die man vielleicht im Biologie-Unterricht unter dem Mikroskop gesehen hat. Das heißt, sie sind größtenteils transparent, mehr oder weniger rund und enthalten einen Kern. Da die Zelle so noch zu leer wäre, werden Speicher- und Stoffwechsel-Organellen hinzugefügt. Die Einzelteile der Zelle werden eher Opak eingestellt, damit die ganze Zelle nicht zu schwer zu erkennen ist.

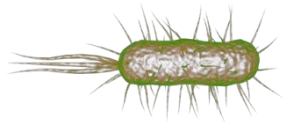

Abbildung 7: Bakterium

Für das Bakterium wird die bekannteste Form, die Stäbchenform gewählt, damit sie von den Spielern auch als Bakterie erkannt wird. Viele Bakterien haben sogenannte Geißeln. Das sind kleine Härchen, die zur Fortbewegung dienen. Da sich die Bakterien in Cell Wars ständig bewegen sollen, müssen auch viele Geißeln zu sehen sein. Dazu wird ein kleines Modell für eine einzelne Geißel erstellt. Dieses wird von einem Partikeleffekt auf der Oberfläche der Bakterie vielfach angezeigt. Es wird eingestellt, dass sich die Partikel vom Bakterienzentrum weg bewegen. Der Geschwindigkeitsvektor wird dann allerdings nicht zum Bewegen verwendet, sondern für die Ausrichtung der Härchen. Zusammen mit einem Noise-Modifier auf der Bakterie führt das zu natürlichen Variationen der Geißelausrichtung. Damit die Bakterie ein Vorne und Hinten bekommt, werden drei besonders große Geißeln an das hintere Ende der Bakterie kopiert. Die diffuse Farbe und Transparenz werden als Falloff eingestellt, was dazu führt, dass die Bakterie vor allem in der Mitte Durchsichtig ist. Die Bakterien sind grün bis bräunlich, damit viele von ihnen einen kranken Eindruck entstehen lassen.

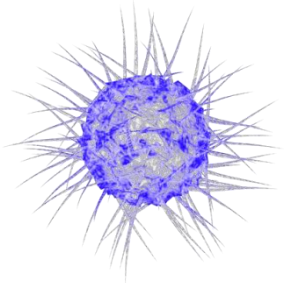

Abbildung 8: Makrophage

Makrophagen sind die gefährlichen Gegner aller Bakterien. Um dies zu verdeutlichen hat ihre Farbe einen starken Kontrast zu der grün-braunen Färbung der Bakterien. Da Makrophagen auch in vielen anderen Illustrationen weiß bis bläulich sind, bietet sich diese Farbe auch in Cell Wars an. Um ihre Gefährlichkeit zu verdeutlichen sind die Fangarme der Makrophage länger als die der Bakterien. Die Makrophagen haben ähnliche Einstellungen für Transparenz und einen ähnlichen Partikeleffekt für die Fangarme. Ein Tentakel wird gesondert heraus gerendert, weil es für den Angriffs-Partikeleffekt im Spiel benötigt wird.

Für den Hintergrund wird ein Bild benötigt, das verdeutlicht, dass das Spiel im Inneren eines Körpers ist. Am besten würde sich dafür ein Querschnitt durch ein Tierisches Gewebe eignen. Da man relativ schwer an solche Bilder mit passender Qualität kommt, eignet sich alternativ auch ein bearbeitetes abstraktes Bild. Die Textur für den Hintergrund wird von innen auf einen Halbkugel gelegt, verstärkt durch eine Schwarz-Weiß-Version als Bump-Map. Durch die Krümmung und die Beleuchtung soll ein Tiefeneindruck entstehen.

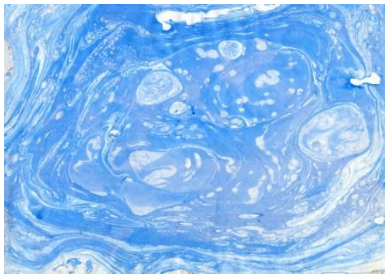

Abbildung 9: Ursprüngliche Version des Hintergrundbildes

Quelle: [6]

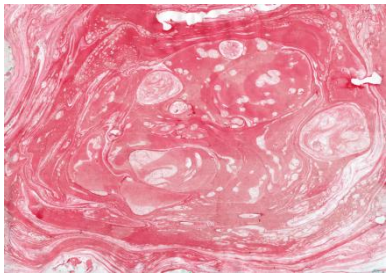

Abbildung 10: Hintergrundtextur durch Farbendrehen

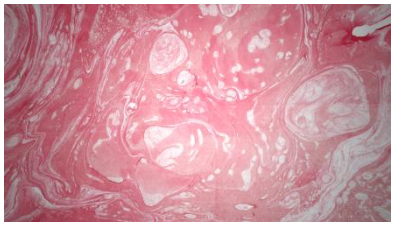

Abbildung 11: Gerendertes Hintergrundbild im 16:9 Format

Sobald die Modelle fertig gerendert sind, können sie als Bilder weiterverarbeitet werden und landen auch als Bilder im Spiel. Einige Assets werden aber direkt als Bild erstellt. Dazu gehören der Formularhintergrund, die Gesundheits- und Resourcenanzeige, Bewegungspartikel und die ganzen Buttons. Dabei muss beachtet werden, dass die Bilder auf dem Hintergrund genügend auffallen und die Qualität für FullHD ausreichend ist, auch wenn sie gestreckt werden.

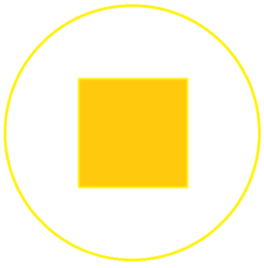

Abbildung 12: Beenden

Abbildung 13: Starten

Abbildung 14: Hilfeseiten

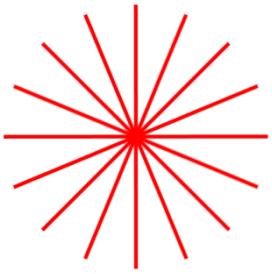

Abbildung 15: Leben

Abbildung 16: Verdoppeln

Abbildung 17: Upgrade

Abbildung 18: Hintergrund für Formulare

Abbildung 19: Auswahl

Die Musik und Sounds werden in vielen Iterationsschritten immer weiter verfeinert, bis die gewünschte Qualität und der passende Stil schließlich erreicht ist. Während die Musik meist nur einmal abgespielt wird, muss beispielsweise der Bakterien-Bewegung-Sound ständig wiederholt werden. Deshalb muss dieser Sound am Ende so ähnlich klingen, wie am Anfang.

[6] http://cgtextures.com/texview.php?id=31697&PHPSESSID=ufr9gjmapcbh3e5r643om9rcs1

3.8. Gamedesign

Obwohl Cell Wars ein eher kleines Spiel ist, soll es trotzdem eine schöne Erfahrung für die Spieler bieten. Das Kampfsystem hilft, ein einzigartiges Spielgefühl zu erzeugen.

Die Kernspielmechanik ist: Die Spieler starten mit ein paar Bakterien. In regelmäßigen Abständen erscheinen Zellen, die von Bakterien gefressen werden können, um ihre Ressourcen aufzufüllen. Sobald sie voll sind, können sie sich entweder weiterentwickeln oder vermehren. Weiterentwickeln macht die betreffende Bakterie widerstandsfähiger, stärker, schneller und größer. Wenn sie sich stattdessen vermehren, werden die Bakterien mit allen Weiterentwicklungen, die sie bereits haben, verdoppelt. In immer kürzeren Abständen kommen immer mehr Makrophagen auf die Spielfläche und greifen Bakterien in ihrer Nähe an. Die Bakterien gewinnen das Spiel, wenn mindestens eine von ihnen fünf Minuten überlebt. Denn dann kommt die Bakterieninvasion und zerstört den Organismus vollständig.

Während Bakterien ständig der nächsten in Reichweite befindlichen Makrophage Schaden zufügen, macht eine Makrophage ihr Opfer vollständig kampfunfähig, während sie es verdaut. Diese Form der asymmetrischen Kriegsführung zwingt die Spieler dazu, ihre Bakterien immer in Gruppen in den Kampf zu schicken. Immer nur Bakterien zu verdoppeln, ist keine Erfolgsgarantie, da sich so am Ende Bakterienhäufchen bilden, die von Makrophagen nach und nach ausgelöscht werden. Außerdem werden gerade am Ende Weiterentwicklungen benötigt, damit sich die Bakterien schnell und effektiv wehren können. Nur auf diese Entwicklungen zu setzen führt genauso wenig zum Erfolg, da die wenigen Bakterien durch das kampfunfähig Machen der Makrophagen einfach überrannt werden. Es liegt also an den Spielern taktisch abzuwägen, wie oft und wie viele ihrer Bakterien sie verbessern, bevor sie zur Reproduktion übergehen.

Da die Makrophagen immer häufiger und mehr von ihnen auftauchen, steigt deren Gesamtkampfkraft exponentiell. Dies hat den angenehmen Effekt, dass Anfänger am Anfang einen leichten Einstieg mit einer flachen Lernkurve haben, das Spiel aber zum Ende hin auch für erfahrene Spieler spannend bleibt. Hätten die Bakterien unbegrenzt Ressourcen, könnten auch sie sich exponentiell vermehren. Da die Ressource in Form von Zellen nur in regelmäßigen Abständen vorkommen, steigt ihre Kampfkraft nur linear. Dadurch, dass Entwicklungen vererbt werden können, lässt sich das Ungleichgewicht der Kampfkräfte teilweise aufheben.

Bei solch unterschiedlichem Verhalten, wie zwischen Makrophagen und Bakterien, muss beim Balancing gut aufgepasst und vor allem viel getestet werden. Durch das Balancing von Werten, wie Lebenspunkten und Schaden lässt sich erreichen, dass die Spieler am Anfang einen Vorteil herausspielen können, der zum Ende hin kleiner werden, aber nicht auf unfaire Weise verschwinden soll.

3.9. Steuerung einzelner Bakterien

Die Steuerung erfolgt ausschließlich über Fingereingaben. Dabei werden folgende Events unterschieden: Finger-Down, Finger-Move und Finger-Up. Die Steuerung der Bakterien soll intuitiv und komfortabel sein. Die Spieler sollen das Gefühl haben, sie könnten die Bakterien herumschubsen. Dazu ist es wichtig, dass sie sich träge weiterbewegen, nachdem sie angeschubst wurden. Solange die Bakterien vom Finger bewegt werden, darf für sie aber keine Masse und Trägheit simuliert werden damit sie sofort und exakt auf Fingerbewegungen reagieren. Physikengines haben typischerweise Probleme, wenn die Masse auf null oder sehr klein gesetzt wird, was aber für das gewünschte Verhalten zwingend nötig ist. Auch die in Angel integrierte Physikengine Box2D dürfte da keine Ausnahme sein. Außerdem sind ihre meisten Features für Cell Wars nicht nötig, macht die Entwicklung aber komplizierter. Daher ist die beste Lösung, die Simulation von Masse und Trägheit selbst zu entwickeln.

In der aktuellen Implementierung wird beim Finger-Down Event der Finger, genau genommen die Cursor ID, in einer Map mit einem Touchable verknüpft. Bei einem Finger-Move Event wird zu der Cursor ID das Touchable gesucht und entsprechend der Cursorbewegung verschoben. Währenddessen werden die letzten Positionsänderungen mit den dazugehörigen Zeiten gespeichert. Die aktuelle Geschwindigkeit wird gedämpft an die errechnete Geschwindigkeit angepasst. Gedämpft deshalb, weil die Finger Events im TUIO-Thread nicht zu jedem Frame erkannt werden und die Geschwindigkeit nicht zu sehr schwanken darf, da sonst die Lautstärke des Bewegungssounds und die Anzahl der Bewegungspartikel springen würde. Sofort nach dem Finger-Up Event wird die aktuelle Geschwindigkeit als Basis für die träge Bewegung genommen. Die aktuelle Geschwindigkeit wird dann linear und exponentiell verringert. Dadurch bewegt sich eine Bakterie kurz mit der Fingergeschwindigkeit weiter, bremst vor allem exponentiell ab, rollt aber nicht endlos weiter. Mit etwas Übung lässt sich die Bewegung der Bakterien nach dem Anschubsen einschätzen.

Dabei ist ein exaktes Einschätzen ist gar nicht nötig, da die Bakterien über eine eingeschränkte Intelligenz verfügen. Sie bewegen sich auf Makrophagen in ihrer Wahrnehmungsreichweite zu und sie fressen Zellen, wenn sie Heilung oder Ressourcen brauchen.

3.10. Steuerung mehrerer Bakterien

Da bei einem Multitouch-Tisch beliebig viele Eingaben gleichzeitig auftreten können, ist es möglich, dass auch mehrere Finger auf eine einzige Bakterie gedrückt werden. Auch dabei darf sie sich nicht unvorhersehbar verhalten. Gelöst wurde dieses Problem dadurch, dass nur ein Finger, die Cursor ID vom ersten Finger-Down Event auf diesem Touchable akzeptiert wird. Das Touchable wird unberührbar für andere Finger, solange der erste nicht wieder abgehoben wurde. So lässt sich auch erreichen, dass aufeinander liegende Bakterien mit mehreren Fingern gleichzeitig bewegt werden können.

Die einfachste Möglichkeit, mehrere Bakterien gleichzeitig zu bewegen, ist mehrere Finger dafür zu benutzen. Doch spätestens ab 10 Bakterien wird dies impraktikabel. Dafür gibt es in Cell Wars die Möglichkeit der Mehrfachselektion. Um so eine Auswahl zu erzeugen, zieht man einen Kreis um alle Bakterien, die zu der Auswahl gehören sollen. Sobald der Finger gehoben wird, erzeugt das Spiel den Auswahlkreis mit dem Upgrade-, Replicate- und Auswahl-Aufheben-Button. Eine Berührung dieser Buttons sorgt dafür, dass alle ausgewählten Bakterien mit vollen Ressourcen entwickelt oder verdoppelt werden. Dann werden dem Auswahlkreis die eingekreisten Bakterien zugewiesen. Das Spiel erkennt diese Bakterien durch einen Point-In-Polygon-Test. Die ausgewählten Bakterien können dann nicht mehr direkt berührt werden. Stattdessen fängt der Auswahlkreis die Eingaben ab und leitet sie an alle ihm zugeordneten Bakterien weiter.

Abbildung 20: Selektionskreis ziehen

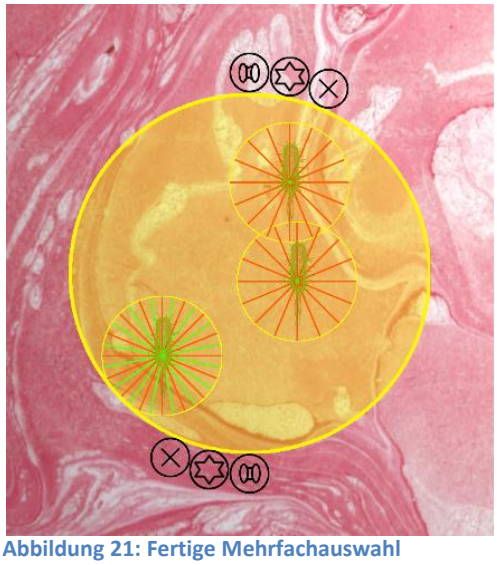

Abbildung 21: Fertige Mehrfachauswahl

Wie bereits beschrieben, wird bei einem Finger-Down Event ein Touchable unter dem Finger gesucht, damit die Cursor ID diesem zugeordnet werden kann. Sollte kein Touchable gefunden werden, wird die ID dem Wert NULL zugeordnet. Wenn beim Finger-Up Event diese NULL zugeordnet wurde, wird dieser Cursor mit seinem Bewegungspfad als potentielle Auswahl behandelt. Wenn Start- und Endpunkt der Bewegung in der Nähe sind, wird der Point-In-Polygon-Test gestartet. Das Polygon ist dabei der Bewegungspfad des Cursor, dargestellt als Liste aus Positionen. Dabei muss die Startposition als Endposition ans Ende der Liste kopiert werden, da sonst ein Lücke im Polygon bleiben würde. Die zu testenden Punkte sind die Positionen aller Bakterien.

Am Anfang dieses Tests steht ein Bounding-Box-Test. Das heißt, für das ganze Polygon werden die maximalen und minimalen X- und Y-Werte gesucht. Nur wenn der zu testende Punkt zwischen diesen Extremwerten liegt, ist der Point-In-Polygon-Test überhaupt nötig. So kann etwas Rechenaufwand

gespart werden, wobei man bedenken muss, dass das ganze Polygon Punkt für Punkt durchgegangen werden muss, um die Extremwerte für den Bounding-Box-Test zu finden. Dieser Test bringt also im Extremfall zusätzlichen Rechenaufwand.

Für den Point-In-Polygon-Test gibt es zwei wichtige Algorithmen: Winding Number und Crossing Number. Sie unterscheiden sich in ihrem Ergebnis dadurch, dass Punkte in Überschneidungen innerhalb des Polygons bei Crossing Number nicht zum Ergebnis gehören. Da die Besonderheit dieses Algorithmus' für Cell Wars keine Nachteile haben dürfte, der Crossing Number Algorithmus dafür einfacher zu verstehen und implementieren ist, wurde er für das Spiel gewählt.

Er basiert darauf, dass vom Testpunkt aus ein Strahl in eine beliebige Richtung (die X-Richtung bietet sich an) ausgesandt wird. Wenn dieser Strahl mit dem Polygon eine ungerade Anzahl an Schnittpunkten hat, liegt der Testpunkt im Polygon.

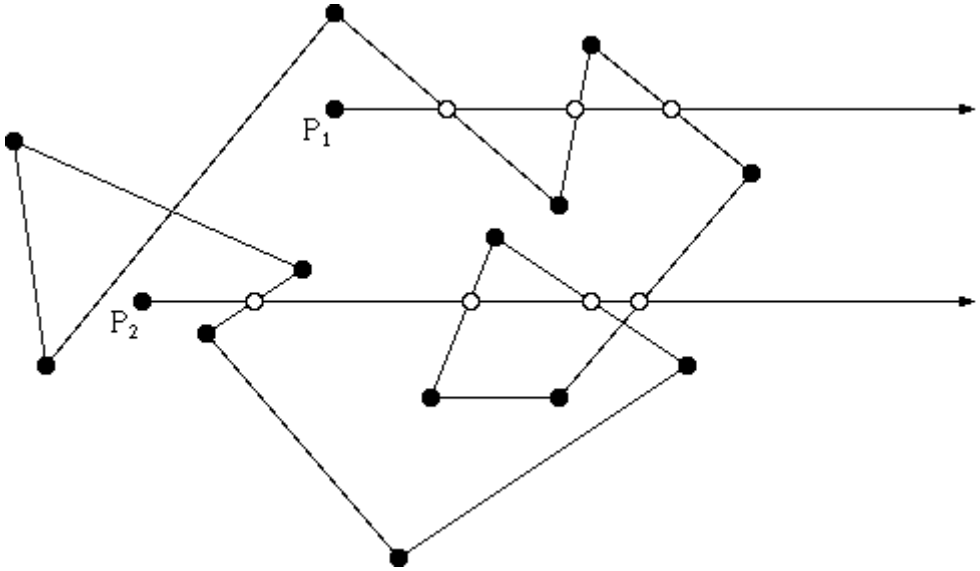

Abbildung 22: Point-In-Polygon-Test[7]

Der Strahl von P1 hat drei Schnittpunkte mit dem Polygon. Das heißt, P1 liegt im Inneren des Polygons. Der Strahl von P2 hat dagegen vier Schnittpunkte, also liegt P2 außerhalb.

Durch die Ergebnisse können alle selektierten Bakterien herausgefunden und einem Auswahlkreis zugeordnet werden

[7] Quelle: http://idav.ucdavis.edu/~okreylos/TAship/Spring2000/PointInPolygon.html(15.06.2013)

3.11. Testen

Gelegentlich muss geprüft werden, ob die implementierten Features auch so funktionieren, wie geplant ist. Die meisten Funktionalitäten lassen sich bereits auf dem Entwicklungssystem testen. Dafür gibt es verschiedene Simulatoren. Leider funktioniert davon einzig der Java TUIO-Simulator. Die meisten anderen starten erst gar nicht oder reagieren auf keine Eingaben. Auf dem Fenster des funktionierenden Simulators können mit der Maus und Tastenkombinationen alle wichtigen TUIO-Events erzeugt werden. Dabei wird die Maus über dem Simulator-Fenster bewegt. Das Spiel läuft allerdings in einem ganz anderen Fenster. Dadurch entsteht ein ungünstiger Abstand zwischen Eingabeposition und Reaktion des Spiels. Dennoch ist der Simulator eine große Erleichterung, da die Entwicklung somit fast unabhängig von den physikalischen Problemen des Trackings ist.

Dennoch muss unbedingt auch auf der Zielplattform getestet werden, da manche Probleme nur dort auftreten. So gab es bei Cell Wars Fehler, wenn ein Cursor innerhalb von einem Frame mehrfach auf- und abgesetzt wird. Nebenbei unterscheiden sich die Aktualisierungsraten von tatsächlicher Trackingsoftware und Simulator.

Manchmal hilft die Zielplattform sogar beim Debuggen. Beispielsweise sind auf dem Entwicklungssystem unregelmäßig unerklärliche Fehler aufgetreten. Auf dem richtigen Multitouch-Tisch, dessen Rechner deutlich schlechter und auch noch von der Trackingsoftware ziemlich ausgelastet ist, traten diese Fehler sehr viel häufiger auf. Die einzige logische Erklärung für solche Phänomene ist ein Fehler beim Multithreading. Und so war es auch bei Cell Wars.

Beim Testen durch andere Personen lassen sich auch unerwartete psychologische Effekte feststellen. Z.B. haben viele Spieler die Angewohnheit, bei einer Mehrfachauswahl den Kreis nicht nur einmal um die Bakterien zu ziehen, sondern sie ziehen in bis zu eineinhalb mal herum. Dies führt dann dazu, dass die Start- und Endpunkte des Polygons zu weit voneinander entfernt sind und die Auswahl dadurch fehlschlägt. Die Ursache dafür ist vermutlich, dass die Spieler dem Spiel die Auswahlerkennung nicht zutrauen und deshalb getreu dem Motto "Besser zu viel, als zu wenig" die Auswahllinie länger ziehen. Durch mehr Erfahrung mit dem Spiel dürfte dieses Verhalten langsam abgebaut werden.

Es hat sich herausgestellt, dass die Spieler in der Hitze des Gefechts, aus Versehen auf den Auswahl-Aufheben-Button von Auswahlkreisen drücken, statt auf den Entwickeln-Button. Dadurch wird die Auswahl aufgehoben und die ganzen Bakterien sind praktisch unkontrollierbar. Gerade am Ende des Spiels kann dies zum Verlieren führen. Gelöst wurde das Problem, indem der Auswahl-Aufheben Button zwei Buttonbreiten von den anderen Buttons abgesetzt wird.

Um das Balancing zu testen, muss man gute mit schlechten Spielern vergleichen. An den Ergebnissen kann man erkennen, ob das Spiel den richtigen Schwierigkeitsgrad hat. Also ob Werte, wie Lebenspunkte, Schaden und Häufigkeit der Objekte richtig eingestellt sind. Im Idealfall haben Einsteiger Zeit, das Spiel zu erlernen, aber auch Profis werden herausgefordert. Bei Cell Wars waren passende Werte nach einigen Versuchen gefunden.

Ein wichtiger Teil des Testens fehlt bei diesem Spiel noch: Das Profiling. Nur mit Profilern können Codestellen entdeckt werden, die optimiert zu deutlich besserer Performance führen.

Nur durch ausgiebiges Testen war es möglich, dass Cell Wars so fehlerfrei geworden ist.

3.12. Installation und Betrieb

Auf der Maschine mit der Trackinghardware muss ein TUIO-Server laufen. Für Cell Wars ist CCV dafür das beste Trackingprogramm.

Für Cell Wars muss auf einem Windows-Rechner das Microsoft Visual C 2010 Redistribution Pack installiert sein. Dieses findet man im Unterordner "MSVC 2010 Redist". Es muss die zum System passende Version installiert werden, also für ein 64-Bit Windows die 64-Bit Version.

Cell Wars selbst muss nicht installiert werde. Das Kopieren auf die interne Festplatte des Rechners sorgt für einen schnelleren Start.

Im Unterordner "bits\Config" liegt die Datei "defaults.lua". Diese Datei kann mit einem beliebigen Texteditor geöffnet und bearbeitet werden. Darin gibt es Einstellmöglichkeiten für Auflösung, Fullscreen, Antialiasing und Weiteres.

4. Fazit und Ausblick

Die Entwicklung von Cell Wars hat einiges an Zeit gekostet. Auch sehr viel Freizeit ist in das Spiel geflossen. Die Meisten die es gesehen und gespielt haben, gaben positive Kommentare ab, woraus man schließen kann, dass sich der Aufwand gelohnt hat. Herausgekommen ist ein kleines Spiel, das trotz seines begrenzten Umfangs eine Menge Hirnschmalz enthält. Auch dadurch, dass man in Cell Wars mit den Bakterien die böse Seite spielt, ist es etwas Besonderes geworden.

Die physikalischen Grundlagen und deren praktische Nutzung sind interessant. Das Infrarotlicht, dass durch die von Fingern unterbrochene Totalreflexion in der Tischplatte in eine Kamera reflektiert wird, ist eine schöne Anwendung für Licht. Es ist auch immer wieder beeindruckend, wie gut die Trackingsoftware mit den Kamerabildern umgehen kann. Dabei ist es sehr beeindruckend, wie vom Erbauer des Tischs aus einfachen Mitteln, wie einer normalen Webcam und einem entwickelten Fotofilm ohne Fotos, eine Kamera mit Infrarotfilter gemacht wurde.

Nicht nur die Arbeit mit der Physik war angenehm, sondern auch die Programmierung mit zwei Quelloffenen Programmen, Angel2D und TUIO-Client.. Selbst das Verschmelzen dieser Teile ging erstaunlich zügig voran. Es gab insgesamt nur wenige unangenehme Stolpersteine. Die größte Herausforderung war, mit funktionierendem Multithreading die Fingereingaben an der richtigen Stelle auf die richtige Weise zu verarbeiten. Und das am besten auch noch in der richtigen Reihenfolge.

Der künstlerische Teil der Arbeit, z.B. die Erstellung der 3D-Modelle, war eine willkommene Abwechslung. Vor allem die interdisziplinäre Arbeit mit Patrick Tejada hat viel Spaß gemacht. Es gab einen ständigen Kontakt mit Austausch und Diskussion von Ideen, Stil, Musik und Sounds. Auch die Kollegen bei Chasing Carrots hatten gute Ideen und Vorschläge. Beispielsweise hat Patrick Wachowiak gezeigt, wie man in 3D Studio Max am besten Härchen auf eine Bakterie bekommt. Auch Vorschläge zum Stil und zur Engine kamen von den Kollegen. Insgesamt lief das ganze Projekt in einem sehr angenehmen und produktiven Umfeld ab.

Dennoch gibt es einige Teile von Cell Wars, die verbessert werden können. Bei Gelegenheit wird dies auch geschehen. Zu den Verbesserungsmöglichkeiten gehört die Steuerung, z.B. dass alle selektierten Bakterien zusammengefasst werden, auch die, die bereits in einer Auswahl sind. Die Zoom-Geste zum Versammeln der Bakterien wäre auch hilfreich. Aber vor allem die Performance dürfte sich durch Profiling noch deutlich steigern lassen. In Zukunft wird es noch eine Version für Maussteuerung geben. Außerdem wird eine Portierung zu Android-Tablets überlegt.

Glossar

Casual Games: Casual Games sind kleine, heute meist browserbasierte Spiele, die nur wenig Zeit der Spieler erfordern.

Gameplay: Gameplay, auch Spielmechanik genannt, ist der Teil, der das eigentliche Spiel ausmacht. Bei Cell Wars das Bewegen und Angreifen der Bakterien.

Game Engine: Game Engine, auch Spiele-Engine genannt, ist ein Framework, das die Entwicklung von Spielen unterstützt. Gute Engines enthalten alle für die Entwicklung nötigen oder hilfreichen Teile, wie z.B. einen Leveleditor und Basisfunktionen.

Mutex: Mutexe sind Datenstrukturen, die dafür sorgen, dass Ressourcen, z.B. Variablen konsistent bleiben, auch wenn mehrere Threads gleichzeitig versuchen darauf zuzugreifen. Ein Thread kann erst darauf zugreifen, wenn die Ressource freigegeben ist.

UDP: User Datagram Protocol ist ein verbindungsloses Protokoll zur Übermittlung von beliebigen Daten. Es verhindert nicht, dass Nachrichten unbemerkt verloren gehen können.

Quellenverzeichnis

[Ang13]: *Angel2D Startseite*. (2013). Abgerufen am 18. Juli 2013 von Angel2D: http://angel2d.com/

[Bur09]: Burger, J. (15. Oktober 2009). *What is so bad about Singletons?* Abgerufen am 14. Juli 2013 von Stackoverflow: http://stackoverflow.com/questions/137975/what-is-so-bad-about-singletons

[Ind13]: *indielib Startseite*. (2013). Abgerufen am 18. Juli 2013 von indielib: http://www.indielib.com/

[Kal09]: Kaltenbrunner, M. (28. August 2009). *TUI 1.1 Protocol Specification*. Abgerufen am 03. Juli 2013 von TUIO.org: http://www.tuio.org/?specification

[Kim11]: Kimmel, D., Mezger, D., & Mezger, S. (25. Juli 2011). Multitouch-Anwendung. Esslingen, Baden-Württemberg, Deutschland: Hochschule Esslingen.

[Lie13]: Liesegang, S. (2013). *Angel2D Dokumentation*. Abgerufen am 04. Juli 2013 von Angel2D: http://docs.angel2d.com/annotated.html

[Löv13]: *Löve Startseite*. (2013). Abgerufen am 18. Juli 2013 von Löve: https://love2d.org/

[Mal11]: Malek, D. (24. Februar 2011). Softwareentwicklung für einen Multitouch-Tisch. Esslingen, Baden-Württemberg, Deutschland: Hochschule Esslingen.

[Row08]: Rowe, S. (28. April 2008). *Prefer Composition Over Inheritance*. Abgerufen am 16. Juli 2013 von MSDN: http://blogs.msdn.com/b/steverowe/archive/2008/04/28/prefer-composition-over-inheritance.aspx

[Sai10]: Sailer, M. (08. Januar 2010). Bau eines Multitouchprototypen. Esslingen, Baden-Württemberg, Deutschland: Hochschule Esslingen.