



Fachhochschule
Kaiserslautern
University of Applied Sciences

Informatik und
Mikrosystemtechnik
Zweibrücken

Studiengang

Medieninformatik

Bachelorarbeit

Konzeption und Umsetzung einer Isometric Tiled Map Engine für das Cocos2D-x Framework

vorgelegt von

Sascha Heyer

27. September 2012

Betreuung:

Prof. Dr. Michael Bender

Zweitkorrektur:

Dr.-Ing. Hubert Zitt

Zusammenfassung

Das Thema dieser Bachelorarbeit umfasst die Konzeption und Umsetzung einer Isometric Tiled Map Engine für das Cocos2D-x Framework, auf Basis von Cross-Plattform Entwicklung.

Es werden vorab die Grundlagen isometrischer, Tile- basierender Verfahren beschrieben, sowie die Grundlagen zu den unterschiedlichen Ansichten, wie zum Beispiel die isometrische im Gegensatz zu der perspektivischen.

Zusätzlich wird auf die Erstellung isometrischer Grafiken eingegangen, da diese speziell erstellt werden müssen.

Weiterhin werden für die Isometric Tiled Map Engine verschiedene Funktionen, wie zum Beispiel das Bewegen oder das Platzieren von Objekten, implementiert und deren Funktionsweisen erläutert.

Ein Schwerpunkt der Isometric Tiled Map Engine liegt bei der Performance, welche unter anderem durch eine möglichst konstante und hohe Framerate realisiert wird, sodass die Engine ruckelfrei laufen kann. Der andere Schwerpunkt liegt bei den Schnittstellen, da durch diese umfassende Nutzungs- und Konfigurationsmöglichkeiten der Engine bestehen.

Abstract

The issue of this Bachelor thesis involves the conception and implementation of an isometric tiled map engine for Cocos2D-x framework, on the basis of cross platform development.

First of all, the basics of isometric, tile- based procedures will be described, as well as the basics of the different views, like the isometric in contrast to the perspective.

Additionally it is described how isometric graphics are created, because there is a special way to produce them.

Furthermore it will be explained how different functions, like moving or placing objects, are implemented for the engine and how they work.

One key aspect for the isometric tiled map engine is the performance, which is, among other things, realized by a mostly constant frame rate, so that the engine can run smoothly. The other key aspect are the interfaces, because they enable the possibility of extensive using and configuration possibilities of the engine.

Inhaltsverzeichnis

1 Einleitung	1
1.1 Projektumgebung Nurogames GmbH	1
1.2 Motivation	1
1.3 Aufbau dieser Arbeit	1
1.4 Nützliche Informationen	2
2 Einführung	3
2.1 Was ist Cocos2D-x	3
2.2 Schlüsselfunktionen	5
2.3 Entwicklungsumgebung einrichten	6
2.3.1 Tegra Android Development Pack	6
2.3.2 Download Cocos2D-x	6
2.3.3 Cocos2D-x umstrukturieren	6
2.3.4 Cocos2D-x Templates für Visual Studio installieren	7
2.4 Cross-Plattform Entwicklung	7
3 Grundlagen Cocos2D-x	8
3.1 Node	8
3.2 Layer	11
3.3 Szene	13
3.4 Szenen-Transition	14
3.5 Szene Graphen	14
3.6 Director	16
3.7 Sprites	17
3.8 Spritesheet	17
3.9 Actions	19
3.10 Label und Font	20
3.11 Menu	22
3.12 Touch Events	23
3.13 Scheduler	24
3.14 Ankerpunkt (Anchor Point)	26
3.15 Accelerometer	27
3.16 Singleton	28
3.17 Sounds / Audio	30
3.18 Fazit	31

4 Entwicklungsrelevantes in Cocos2D-x	32
4.1 Native Development Kit	32
4.2 JNI	32
4.2.1 Aufruf einer Java Methode von C/C++ aus	33
4.2.2 Aufruf einer nativen Methode von Java aus	35
4.3 Editoren / Tools für Cocos2D-x	35
4.4 OpenGL ES	36
4.5 Präprozessormacros	36
5 Erweiterung der Nuroengine-X	37
5.1 Gestenerkennung (GestureRecognizer)	37
5.1.1 Implementierte Funktionen der Gestenerkennung	37
5.1.2 Swipeerkennung / Swiperichtung	38
5.1.3 Swipewinkel	38
5.1.4 Swipe Distanz	39
5.1.5 Bewegungsdistanz	40
5.1.6 Bewegungsgeschwindigkeit	41
5.1.7 Umwandlung der Touch Koordinaten	42
5.1.8 Nutzung der Gestenklasse	42
5.2 Scrollmenu (gestenbasiert)	43
5.2.1 Scrollen	43
5.2.2 Ausrichtung des Menüs	45
5.2.3 Menügrenzen	45
5.2.4 Unterklasse von Menu	45
5.2.5 Nutzen der ScrollMenu Klasse	45
5.2.6 Performance Analyse	47
5.3 WebView	48
5.3.1 Struktur des Systems	48
5.3.2 Instanzen der WebView Klasse	49
5.3.3 Init	50
5.3.4 Callbacks	50
5.3.5 Callback registrieren	51
5.3.6 Javascript aufrufen	52
5.3.7 WebView Implementierung Android seitig	53
5.3.8 WebView Einstellungen	54
5.3.9 HTTP-Authentifizierung	54
5.3.10 Kommunikation von Java zu C++ sowie JavaScript Schnittstelle	55
5.3.11 JavaScriptInterface Java seitig hinzufügen	57
5.3.12 Nutzen der WebView Klasse	58
5.3.13 Nutzung des JavaScript Interfaces	58

6 Isometric Tiled Map Engine	59
6.1 Grundlagen	59
6.1.1 Tiled Maps Arten	59
6.1.2 Unterscheidung zwischen Isometrie und Perspektive	61
6.1.3 Erzeugung regelmäßiger Pixelmuster	62
6.1.4 Faktor 2:1	62
6.1.5 Isometrische Tile (Grundfläche) erstellen	62
6.1.6 Isometrische Grafiken erstellen	63
6.1.7 Blender Einstellungen zur Erstellung isometrischer Renderings	63
6.2 Analyse	65
6.2.1 Ausschlussanalyse Cocos2D-x Tiled Map	65
6.2.2 Anforderungsanalyse	65
6.3 Entwurf / Konzeption	71
6.3.1 Übersicht über die Tile Klasse	71
6.3.2 Übersicht über die TileObject Klasse	72
6.3.3 Übersicht über die TileObjectPath Klasse	73
6.3.4 Übersicht über die TileMap Klasse	73
6.3.5 Übersicht über die TileController Klasse	74
6.3.6 Übersicht über die TileMath Klasse	75
6.4 Implementierung	75
6.4.1 Tile	75
6.4.2 TileMap	78
6.4.3 TileObject	88
6.4.4 TileObjectPath	93
6.4.5 TileMath	96
6.4.6 TileController	102
6.5 Performanceoptimierung	106
6.5.1 Clipping- und Visibilityoptimierung	107
6.5.2 Iterationsoptimierung	108
6.5.3 Performancevergleich	110
6.5.4 Weitere Optimierungsmöglichkeiten	111
6.6 Nutzung	111
6.6.1 Schnittstellen	111
6.6.2 Tiled Map initialisieren	112
6.6.3 Controller initialisieren	112
6.6.4 Hintergrund hinzufügen	112
6.6.5 Hinzufügen von Objekten	113
6.6.6 Opacity Modus	113
6.6.7 Setzen der Tiled Map Modis	113
6.6.8 Nutzung des Callbacks	114
6.6.9 Beispielhafte Anwendung der Isometric Tiled Map Engine	115

6.7 Ausblick und Einordnung des Cocos2D-x Frameworks	115
6.7.1 Probleme während der Entwicklungsphase	116
6.7.2 Vorteile gegenüber der in Cocos2D-x integrierten TiledMap Lösung	116
7 Fazit	117
Anhang	118
Quellenverzeichnis	120
Abbildungsverzeichnis	121
Tabellenverzeichnis	123
Klassendiagramm	126

1 Einleitung

1.1 Projektumgebung Nurogames GmbH

Nurogames ist eine Spieleentwicklungs firma mit dem Schwerpunkt auf Cross-Plattform Entwicklung im Bereich Mobile Games. Hierbei kommen verschiedene Techniken wie C++, ObjectivC, Java und Cocos2D-x zum Einsatz. Das Entwicklerstudio befindet sich im Herzen von Köln und besteht aus einem jungen und dynamischen Team.

Kurze Entscheidungszyklen und Gestaltungsfreiraum lassen eine sehr agile und kreative Entwicklung zu.

1.2 Motivation

Die Motivation für diese Arbeit besteht darin, einer der komplexesten Bereiche der Spieleentwicklung, die Game Engine Entwicklung, kennen zu lernen.

Ohne Engine lässt sich ein Spiel nur mit erheblichem Aufwand entwickeln. Die Nutzung einer Engine bietet sehr viele Vorteile, welche sich von der Einfachheit bis hin zur Zeitersparnis bei der Entwicklung erstrecken.

Zudem ist die Teamdynamik in keinem anderen IT-Bereich so vielfältig, da in der Spielebranche sehr viele Berufszweige zusammenarbeiten und man somit auch Einblicke in die verschiedenen Bereiche bekommt.

Ein weiterer wichtiger Punkt ist ein Einblick in die Gamesbranche zu bekommen, um die spätere Berufswahl besser treffen zu können.

1.3 Aufbau dieser Arbeit

Kapitel zwei erläutert das nötige Vorwissen zu Cocos2D-x, die Entstehungsgeschichte davon und auch die verschiedenen Funktionalitäten, die es bietet.

In Kapitel drei werden die Grundlagen von Cocos2D-x vermittelt und ein grundlegendes Verständnis dafür geschaffen.

Kapitel vier beschäftigt sich mit entwicklungsrelevanten Themen, die im Laufe dieser Arbeit von Bedeutung sind und erwähnt werden sollten.

Kapitel fünf beschreibt die Projekte, die in der Praxisphase entwickelt wurden. Diese Projekte erweitern die firmeninterne, auf Cocos2D-x basierende, Engine um weitere Funktionen. Im Rahmen der Praxisphase sind hierbei vier Projekte entstanden. Auf drei von diesen wird näher eingegangen und deren Implementierung und Nutzen erklärt.

Die Callback Komponente der WebView wird als Grundlage für die Bachelorarbeit angesehen und gehört somit noch dazu. Weiterhin kann das Kapitel Entwicklungsrelevantes in Cocos2D-x auch als zur Bachelorarbeit zugehörig gesehen werden, da dies hier ebenso relevant ist.

Kapitel sechs umfasst die Bachelorarbeit. Der Themenbereich dieser Arbeit ist eine Isometric Tiled Map Engine. Es werden die Grundlagen erklärt, sowie eine Analyse der Tiled Map Engine erstellt und mögliche Alternativen begutachtet. Weiterhin umfasst dieses Kapitel somit auch die Konzipierung und Umsetzung der Isometric Tiled Map Engine.

Kapitel sieben ist das Fazit dieser Arbeit.

1.4 Nützliche Informationen

Nützliche Informationen, die entweder Performancerelevant oder anderweitig wichtig sind, befinden sich in eingerahmten Boxen.

Dabei ist zwischen einer Hinweisbox und einer Optimierungsbox zu unterscheiden.

Optimierung:

Hilfreiche Optimierungsmöglichkeiten

Hinweis:

Hinweis auf etwas, auf das besonders Wert gelegt werden sollte

2 Einführung

Dieses Kapitel beeinhaltet Informationen rund um Cocos2D-x, Historie, Entwicklung, Aufbau und die ersten Schritte des Frameworks.

2.1 Was ist Cocos2D-x



Abb. 2.1: Cocos2D-x Logo Landscape

Cocos2D-x ist ein Open Source 2D Framework für mobile Spiele und setzt den Fokus auf Cross-Plattform Entwicklung (näheres in Abschnitt Cross-Plattform Entwicklung). Spiele können in C++ oder Lua entwickelt werden.

Da es erst seit rund einem Jahr existiert, findet man keine Literatur dazu.

Lizenz

Cocos2D-x steht unter MIT¹ Lizenz. Die MIT Lizenz ist eine sehr offene Lizenz, was heißt, dass mit der Software oder dem Quellcode alles gemacht werden kann. Es muss lediglich der Urheberrechtsvermerk erhalten bleiben und es wird nicht für eventuelle Fehler gehaftet.

Im Sinne der Entwickler von Cocos2D-x wird gewünscht, das Logo als Splashscreen anzuzeigen, dies ist jedoch kein muss.

Community

Die Community ist sehr aktiv und berücksichtigt eingereichte Änderungswünsche oder behebt auch zeitnah gemeldete Bugs. Das zeigt, dass die Verwender von Cocos2D-x sehr stark eingebunden werden.

¹Massachusetts Institute of Technology Licence

Im Forum können Fragen gestellt und Anregungen ausgetauscht werden.

Dokumentation

Zu Cocos2D-x existiert eine ausführliche Dokumentation in Form von Tutorials, Framework-Dokumentation und Beispielanwendungen. Alternativ steht ein Forum zur Verfügung, in dem Fragen gestellt werden können.

Spiele

Die Anzahl angegebener Spiele, welche mit Cocos2D-x entwickelt wurden, liegt laut Projektseite bei 340 (Stand: 01.Juni.2012). Da dies nur die Spiele sind, die auf der Projektseite hinzugefügt wurden, ist die Anzahl wohl noch weitaus größer.

Zukunft

Für die Zukunft wird Cocos2D-x das hauptsächlich benutzte 2D- Framework sein, da es kostenlos und sehr aktiv ist. Kaum ein anderes Framework im Cross-Plattform Bereich bietet eine so große Anzahl an Möglichkeiten.

Cocos2D Varianten

- Cocos2D iPhone
iOS und OS X
- Cocos2D Android
Android, Windows und Ubuntu
- Cocos2D Windows Phone (xna)
Windows Phone, Windows
- Cocos2D-x
iOS, Android, Windows, OS X und Ubuntu
- Cocos2D-x lua
iOS, Android, Windows Phone, Windows, OS X und Ubuntu
- Cocos2D-html5
iOS, Android, Windows Phone, Windows, OS X, Ubuntu und Browser
- Cocos3d

Weitere Plattformen:

- Bada
- Marmalade
- LePhone
- MeeGo
- WoPhone

Framework Aufbau

Der grüne Bereich ist die Android Komponente, kann aber auch eine der möglichen Plattformen sein. Der lila Bereich beschreibt den OpenGL Teil, welcher in die Plattform Komponente integriert ist. Auf den genannten Bereichen baut schließlich noch der blaue Bereich auf, bei welchem es sich um Cocos handelt (Abb. 2.2).

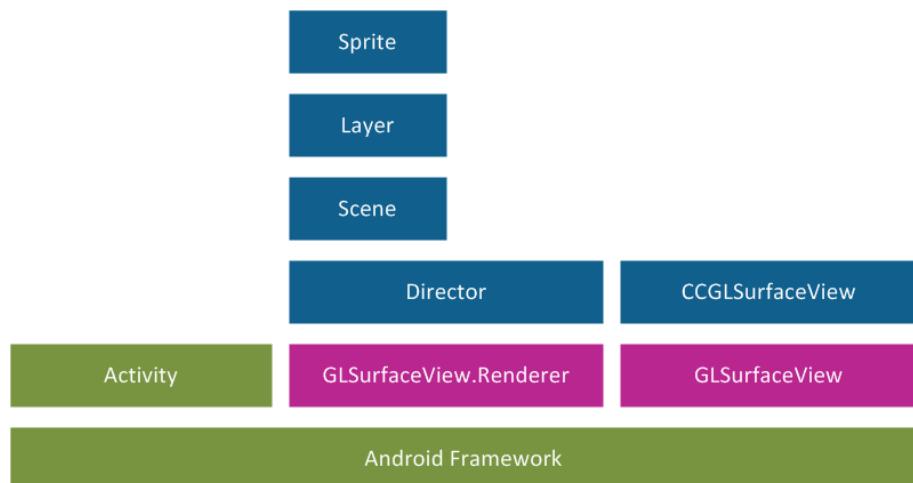


Abb. 2.2: Aufbau des Frameworks beispielhaft mit Android

2.2 Schlüsselfunktionen

Cocos2D-x umfasst eine Gruppe von Funktionen, welche im Folgenden gelistet sind:

- Szenengraph
- Szenentransition
- Sprites
- Animationsverhalten
- GUI Elemente
- Partikelengine
- 2D Physik Engine
- Touch, Maus und Accelerometer
- Sound und Effekt Engine
- Text Rendering
- Texture Atlas
- Geräteorientierung
- Cross-Plattform

2.3 Entwicklungsumgebung einrichten

Bei der Entwicklung mit Cocos2D-x werden verschiedenste Tools, Entwicklungsumgebungen und SKDs benötigt. In diesem Kapitel wird der Nutzen, sowie deren Installation und Konfiguration erklärt.

2.3.1 Tegra Android Development Pack

Nvidia bietet ein Entwicklungspaket an, in welchem die wichtigsten Tools, die im Rahmen der Entwicklung mit Cocos2D-x benötigt werden, enthalten sind. Wenn keine spezielle Version der einzelnen Tools benötigt wird, bietet es sich an, das Nvidia Android Development Pack zu nutzen.

In dieser Version sind folgende Tools enthalten:

- NVIDIA Debug Manager
- JDK 6u24
- CDT 8.0.0
- Android SDK r18
- Cygwin 1.7
- ADT 15.0.0
- Android NDK r7c
- Eclipse 3.7.1
- Apache Ant 1.8.2

Der Vorteil an diesem Paket liegt darin, dass nicht jedes Tool beziehungsweise SDK einzeln heruntergeladen und installiert werden muss. Dies spart Zeit und das aufwendige Einrichten der Tools entfällt. Heruntergeladen werden kann das **Tegra Pack** auf der NVIDIA Developer Seite.

2.3.2 Download Cocos2D-x

Zuerst muss die Cocos2D-x Version, welche verwendet werden soll, heruntergeladen werden. Die Versionen sind auf der Cocos2D-x Seite zu finden. Die hier verwendete Version von Cocos2D-x ist: cocos2d-1.0.1-x-0.12.0. Nachdem die benötigte Version heruntergeladen ist, muss sie an einem geeigneten Platz entpackt werden.

2.3.3 Cocos2D-x umstrukturieren

Für die Nutzung in größeren Projekten macht es Sinn, Cocos2D-x umzustrukturen. So bietet es sich an, die einzelnen Komponenten, die in Cocos2D-x enthalten sind, jeweils in eigenen Libraries auszulagern.

Für kleine Projekte kann die Struktur von Cocos2D-x beibehalten werden.

2.3.4 Cocos2D-x Templates für Visual Studio installieren

Im Cocos2D-x root Verzeichnis befindet sich die "install-templates-msvc.bat". Diese Datei installiert für Visual Studio 2008 und 2010 die Cocos2D-x Templates.

Danach ist es möglich in Visual Studio ein Cocos2D-x Projekt anzulegen.

2.4 Cross-Plattform Entwicklung

Da immer mehr Endgeräte, mit verschiedensten Betriebssystemen auf dem Markt erscheinen, müssen Applikationen auf mehr als nur einer Plattform entwickelt werden. Um das komplette Spektrum an Kunden zu erreichen, muss die Applikation auf möglichst vielen Geräten lauffähig sein.

Um Zeit und auch Kosten zu sparen, macht es Sinn, mit Hilfe von Cross-Plattform Entwicklung für mehrere Geräte, beziehungsweise Systeme, direkt zu entwickeln.

In den letzten Jahren sind einige nützliche Tools und Frameworks, im Bereich Mobil Cross-Plattform Entwicklung, erschienen. Cocos2D-x ist ein solches Framework und bietet Unterstützung für folgende Plattformen an:

- iOS
- Android
- Bada
- Win32
- Linux
- BlackBerry Tablet OS

3 Grundlagen Cocos2D-x

In diesem Kapitel werden die Grundlagen von Cocos2D-x erklärt und anhand von Beispielen die grundlegenden Funktionen veranschaulicht.

Hilfreiche Tipps bezüglich Problemvermeidung und Optimierung sind in Hinweis und Optimierungsboxen beschrieben.

3.1 Node

CCNode ist die Basisklasse aller Klassen, die Elemente beeinhalten, die gezeichnet werden müssen. Das bedeutet, dass alle anderen Klassen, bei denen etwas gezeichnet werden muss, Unterklassen von CCNode sind und somit deren Attribute und Methoden (public, protected) nutzen können.

Aufgrund der großen Anzahl an öffentlichen Methoden, werden im Folgenden nur die wichtigsten gelistet:

- Position (x, y)
- Skalierung
- Rotation
- Ankerpunkt
- Größe
- Sichtbarkeit
- Z-Anordnung
- OpenGL Z-Anordnung
- Tag (Identifikation)
- Kinder
- Schedule
- Konvertierung in verschiedene Koordinaten

Node erzeugen

Ein CCNode Objekt kann wie in (Quellcode 3.1) erzeugt werden.

Quellcode 3.1: Node erzeugen

```
1 CCNode* pkNode= new CCNode();
```

Kinder hinzufügen

Es ist möglich, einem Node, Kinder hinzuzufügen und somit einen Graphen zu erstellen. Wie genau dieser Graph aufgebaut ist, wird in Kapitel 3.5, Szene Graphen erläutert.

Dabei kann ein Kind immer nur zu einem Elternteil gehören. Es ist nicht möglich ein Kind mehreren Eltern hinzuzufügen.

Beim Hinzufügen von Kindern ist zu beachten, dass es drei überladene Methoden von addChild gibt. In Quellcode 3.2, Zeile 1 wird ein Kind ohne zusätzliche Parameter hinzugefügt. In den nächsten Abschnitten folgen die Beispiele mit zwei und drei Parametern.

Quellcode 3.2: Kinder hinzufügen

```
1 pkNode->addChild( firstChildObj );
```

z-Ausrichtung

Die z-Ausrichtung (Quellcode 3.3) beschreibt die Anordnung auf der z-Achse. Nimmt man zwei Blätter Papier und gibt dem ersten Blatt eine z-Ausrichtung von 0 und dem anderen Blatt eine 1 so liegt das Blatt mit der 1, über dem mit der 0.

Quellcode 3.3: CCNode z-Order

```
1 pkNode->addChild( firstChildObj, 0 );
2 pkNode->addChild( secondChildObj, 1 );
```

Tag-ID

Die letzte Methode, mit drei Parametern (Quellcode 3.4 Zeile 1), gibt dem Kind eine ID, mit welcher direkt auf ein Kind zugegriffen werden kann (Quellcode 3.4 Zeile 2).

Quellcode 3.4: CCNode Tag-ID

```
1 pkNode->addChild( thirdChildObj, 1, 42 );
2 CCNode* pkRetrievedNode = pkNode->getChildByTag( 42 );
```

Hinweis:

Es ist möglich, mehreren Kindern die gleiche ID zu geben. In diesem Falle wird beim Zugriff auf ein Kind mit einer ID das erste Kind genommen. Daher sollten die vergebenen IDs immer eindeutig sein.

Kind(er) entfernen

Kinder lassen sich auf zwei Arten entfernen. Die erste Möglichkeit besteht darin, alle Kinder zu entfernen (Quellcode 3.5, Zeile 2). Die zweite Möglichkeit ist das Entfernen von Kindern mit Hilfe eines Zeigers (Quellcode 3.5 Zeile 3).

Quellcode 3.5: CCNode Kind entfernen

```
1 CCNode* pkRetrievedNode = pkNode->getChildByTag( 42 );
2 pkNode->removeAllChildrenWithCleanup( true );
3 pkNode->removeChild( pkRetrievedNode );
```

Position

Nodes können an eine bestimmte Position gesetzt werden. In (Quellcode 3.6, Zeile 1) wird ein Node an die Position (50, 100) gesetzt. Analog kann auch die Position ausgesehen werden (Quellcode 3.6, Zeile 2).

Quellcode 3.6: CCNode Position

```
1 pkNode->setPosition( ccp( 50, 100 ) );
2 CCPoint kNodePosition = pkNode->getPosition();
```

Ankerpunkt

Es besteht die Möglichkeit, den Ankerpunkt zu ändern (Quellcode 3.7). Auf die genauen Auswirkungen wird in Abschnitt 3.14, Ankerpunkte eingegangen.

Daher wird dies hier nur erwähnt.

Quellcode 3.7: Ankerpunkt

```
1 pkNode->setAnchorPoint( x, y );
```

Sichtbarkeit

Die Sichtbarkeit von Elementen kann eingestellt werden (Quellcode 3.8). Dabei ist besonders der Faktor, in Bezug auf die Zeichenmethode innerhalb von Cocos2D-x, relevant. Objekte, deren Sichtbarkeit auf *false* gesetzt ist, werden nicht gezeichnet.

Quellcode 3.8: Sichtbarkeit

```
1 bool bIsNodeVisible= pkNode->getisVisible();
2 pkNode->setisVisible( false );
```

Skalierung

Die Skalierung eines CCNode kann ebenfalls eingestellt werden (Quellcode 3.9). Es sollte jedoch beachtet werden, dass sich ein Skalierungsfaktor >1 negativ auf die Qualität der, sich innerhalb dieses Nodes befindlichen, Assets auswirkt.

Quellcode 3.9: Skalierung

```
1 pkNode->setScale(0.5);
2 float fNodeScale = pkNode->getScale();
```

Rotation

Nodes können rotiert werden, wobei die Rotation in Degree angegeben wird (Quellcode 3.10).

Quellcode 3.10: Rotation

```
1 pkNode->setRotation( 90)
2 float fNodeRotation = pkNode->getRotation();
```

3.2 Layer

Unter einem Layer versteht man eine Ebene, ähnlich wie in Photoshop, auf welcher verschiedene Elemente abgelegt sind. Es folgt ein Anwendungsbeispiel anhand eines Spiels:

Ein Spiel besteht aus einem Hintergrund und einem Benutzer Interface (Leben, Punkte), sowie einer Spielfigur.

Diese Spielemente können auf verschiedene Layer ausgelagert werden. Sie liegen übereinander und erscheinen somit als ein Ganzes.

Somit ist ein Layer eine logische Abstraktion von Elementen auf dem Bildschirm.

Quellcode 3.11: Layer

```
1 CCLayer* pkBackgroundLayer = new CCLayer();
2 CCLayer* pkForegroundLayer = new CCLayer();
3
4 CCSprite* pkBackgroundSprite = CCSprite::spriteWithFile(
5                                         "background.png");
6 CCSprite* pkForegroundSprite = CCSprite::spriteWithFile(
7                                         "moon.png");
8
9 pkBackgroundSprite->setPosition( CCPoint( kDeviceSize.width/2,
10                                     kDeviceSize.height/2));
```

11

```

12 pkForegroundSprite->setPosition( CCPoint( kDeviceSize.width/2,
13                                     kDeviceSize.height/2));
14
15 pkBackgroundLayer->addChild( pkBackgroundSprite);
16 pkForegroundLayer->addChild( pkForegroundSprite);
17
18 addChild( pkBackgroundLayer, 1);
19 addChild( pkForegroundLayer, 2);

```

Die folgende Grafik (Abb. 3.1) zeigt die Ausgabe des in Quellcode 3.11 implementierten Codes. Der Hintergrund liegt auf einem anderen Layer als der Mond. Man erkennt, dass sich der Mond über dem Hintergrund befindet. Dies resultiert aus der z-Ausrichtung der Layer.

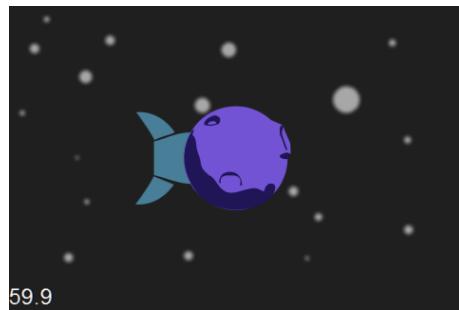


Abb. 3.1: Layer übereinander angeordnet

Im Folgenden sind die Ebenen (Layer) aufgesplittet (Abb. 3.2), um die Anordnung besser darzustellen.

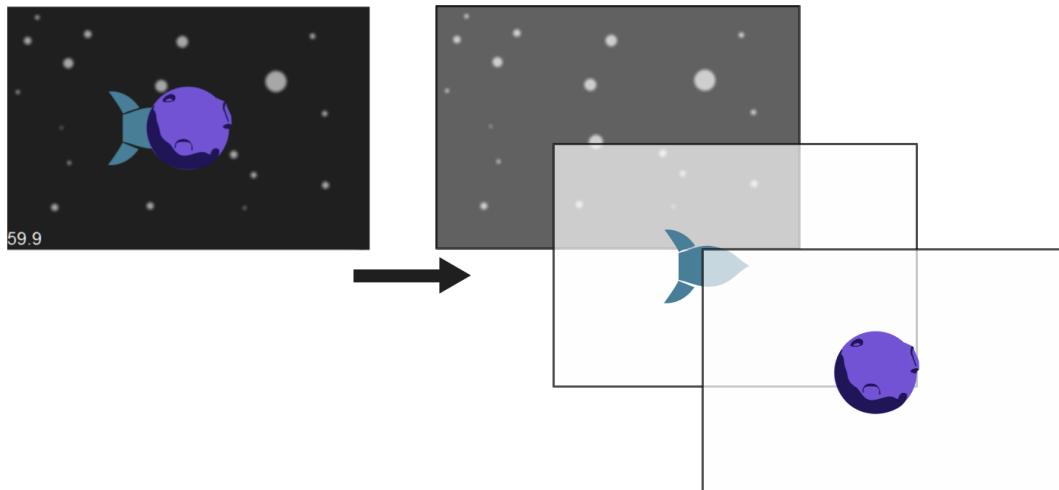


Abb. 3.2: Layer visuell sichtbar gemacht

Besondere Benutzung von Layern:

Auf Layern ist es möglich, auf Touch (Kapitel 3.12), Accelerometer (Kapitel 3.15), Tastatur und Mouse Events zu reagieren.

3.3 Szene

Cocos2D-x ist in Szenen organisiert. Dies kann man sich vorstellen, wie einen Bildschirm. Es gibt in einem Spiel mehrere Szenen, zum Beispiel für Menu, Highscore, GameOver oder Ähnliches.

Eine Szene ist die Umsetzung eines View Controllers und wird mit dem CCDirector (siehe Kapitel 3.6 Director) verwaltet. Es ist möglich, für den Wechsel zwischen Szenen, Transition-Effekte zu nutzen (siehe Kapitel 3.4).

Eine Anwendung kann mehrere Szenen besitzen, jedoch ist maximal eine aktiv.

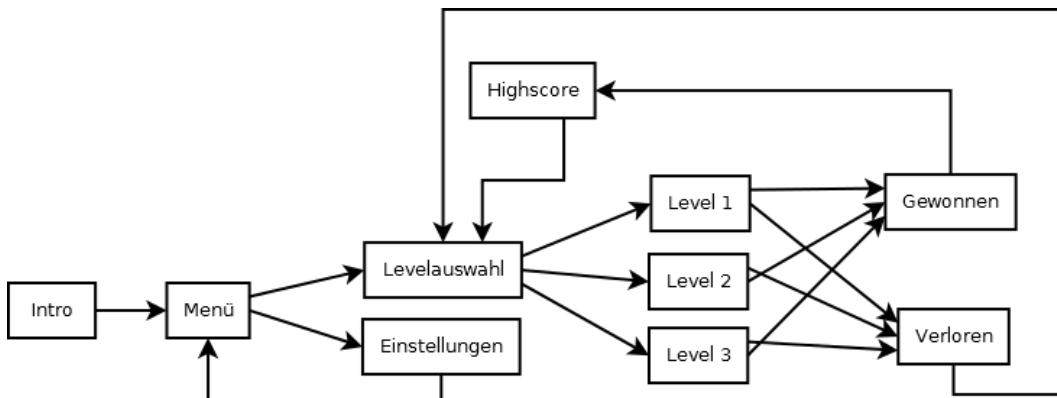


Abb. 3.3: Aufbau Szenestruktur eines Spieles

Push und Pop von Szenen

Szenen können auf den Stack gelegt und abgerufen werden (Quellcode 3.12 Zeile 1 und 2). Dies macht dann Sinn, wenn zum Beispiel die Settings geöffnet werden und nur für eine kurze Dauer eine Szene auf den Stack gelegt wird.

Wenn man jedoch von der Levelauswahl einen Level startet, sollte die Szene mit replaceScene, siehe Quellcode 3.12, Zeile 3, ersetzt werden.

Quellcode 3.12: CCDirector

```

1 CCDirector::sharedDirector() -> pushScene( pScene );
2 CCDirector::sharedDirector() -> popScene ( pScene );
3 CCDirector::sharedDirector() -> replaceScene( pScene );
  
```

Optimierung:

Push und Pop von Szenen sollte vermieden werden, da es sehr teuer ist ganze Szenen auf dem Stack abzulegen. Nur in geeigneten Fällen benutzen!

3.4 Szenen-Transition

Unter einer Szenen-Transition ist der Wechsel zwischen zwei Szenen zu verstehen.

TransitionsScene

TransitionFade, TransitionFlipAngular, TransitionShrinkGrow, TransitionMoveInB, TransitionMoveInT, TransitionMoveInL, TransitionMoveInR, TransitionFadeTR, TransitionFadeUp, TransitionFlipX, TransitionFlipY, TransitionPageTurn, TransitionCrossFade

In Quellcode 3.13 ist die Implementierung eines Szenenwechsels zu sehen.

Quellcode 3.13: Szenenwechsel

```
1 CCDirector* pkDirector = CCDirector::sharedDirector();
2 pkDirector->replaceScene(
3     CCTransitionFlipX::transitionWithDuration( 0.7f, otherScene));
```

Hinweis:

Transitions sollten wohl gewählt genutzt werden. Zu viele verschiedene Transitions lassen die Anwendung schnell "billig" aussehen.

3.5 Szene Graphen

Die Grundbegriffe Node, Szene und Layer wurden in den vorherigen Abschnitten bereits erklärt. Im Folgenden wird näher auf den Szenen Graphen von Cocos2D-x eingegangen. In Kapitel 3.1 wurde erklärt, dass alles in Cocos aus CCNodes besteht, womit es also möglich ist, einen Szenen Graphen (Abb. 3.4) "fast" beliebig aufzubauen.

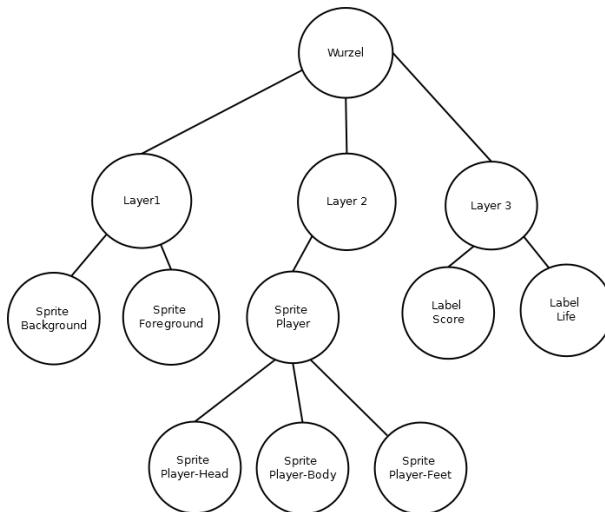


Abb. 3.4: Szenengraph

Optimierung:

Um Objekte zu gruppieren, sollte wenn möglich ein CCNode und kein CCLayer genutzt werden. Der Overhead ist bei CCLayer höher als bei CCNode, da CCLayer auch auf Touch und anderes reagieren muss. Es macht also Sinn, CCNode zum Gruppieren von Objekten zu nutzen, wenn kein Touch oder Ähnliches benötigt wird.

Es muss jedoch beachtet werden, dass die Struktur, beziehungsweise der Aufbau, auch sinnvoll ist. In Cocos ist nicht sofort ersichtlich, dass bei jedem Hinzufügen von zum Beispiel Items zu einem Menu, addChild(...) aufgerufen wird. So können bei CCMenus die Kinder mit der Methode

Quellcode 3.14: Layer als Kind von Scene setzen

```
1 static CCMenus* menuWithItems (CCMenuItem *item, ...)
```

hinzugefügt werden. Hier ist also nicht direkt ersichtlich, dass ein addChild aufgerufen wird. Bei genauerer Betrachtung des CCMenus Codes (Quellcode 3.15 und 3.16) ist zu erkennen, dass indirekt wieder addChild aufgerufen wird. Also kann man daraus schließen, dass der Aufruf von addChild bei einem Menü auch direkt möglich ist.

Die Methode menuWithItems() ruft die Methode initWithItems() auf, welche das eigentliche Hinzufügen der Objekte in den Graphen übernimmt.

Quellcode 3.15: Menü erzeugen Methode

```
2 CCMenus * CCMenus::menuWithItems(CCMenuItem* item, ...)  
3 {  
4     va_list args;  
5     va_start(args, item);  
6     CCMenus *pRet = new CCMenus();  
7     if (pRet && pRet->initWithItems(item, args)) // Aufruf der  
8         initWithItems Methode  
9     {  
10 ...
```

Quellcode 3.16: Menü mit Items initialisieren Methode

```
10 bool CCMenus::initWithItems(CCMenuItem* item, va_list args)  
11 {  
12     if (init())  
13     {  
14         int z=0;  
15         if (item)  
16         {  
17             this->addChild(item, z);  
18             CCMenuItem *i = va_arg(args, CCMenuItem*);  
19             while (i)  
20             {
```

```

21     z++;
22     this->addChild(i, z); // Aufruf von addChild
23 ...

```

Hier dienen die Methoden jedoch zur vereinfachten Nutzung der CCMenus Klasse. Analog ist dies auch in den anderen Klassen realisiert. Es wird gezeigt, dass bei jedem Hinzufügen von Objekten, zu einem anderen Objekt, immer die addChild Methode aufgerufen wird. So wird der Graph vollständig erzeugt.

3.6 Director

Für die Verwaltung des Fensters und der Szenen ist der CCDirector zuständig. Ein Spiel enthält zum Beispiel mehrere Szenen wie Levelauswahl, Gameover und weitere. Zwischen diesen Szenen kann man via Director wechseln.

Der Director ist als Singleton implementiert. Vorteile und Nutzung dieses Musters wird in Kapitel 3.16 Singleton näher erläutert.

Weiterhin ist der CCDirector für folgende Funktionen zuständig:

- Bietet Zugriff auf die aktuell laufende Szene
- Start, replace, push und pop von Szenen
- Bietet Zugriff auf Cocos2D-x configuration details
- OpenGL view und window
- Pausieren, Fortsetzen und Beenden eines Spiels
- Koordinaten konvertieren

Da der Director ein Singleton ist, kann auf diesen direkt klassenübergreifend zugegriffen werden, siehe (Quellcode 3.17). In Quellcode 3.17, Zeile 1 wird beispielhaft die Displaygröße ausgelesen und gespeichert. Da der Zugriff auf die anderen Methoden analog ist, wird in Quellcode 3.17, Zeile 2 ein Platzhalter verwendet, welcher je nach Anforderung ersetzt werden kann. Eine komplette Liste aller Methoden kann der Cocos2D-x Dokumentation entnommen werden.

Quellcode 3.17: Shared Director

```

1 CCSIZE kDeviceSize = CCDirector::sharedDirector()->getWinSize();
2 CCDirector::sharedDirector()-><METHOD>

```

3.7 Sprites

Ein Sprite ist ein aktives Objekt in Cocos2D-x, welches zweidimensional ist, ein Bild beeinhaltet und animiert werden kann. Alle Grafiken, die für ein Spiel relevant sind, können als ein Sprite verstanden werden. Jedoch kann auch alles andere, was in einem Spiel sichtbar und kein GUI Element ist, als ein Sprite angesehen werden.

Wie in Quellcode 3.18, Zeile 2 ersichtlich ist, kann ein Sprite mit dem Dateinamen erzeugt werden. Mit setPosition (Quellcode 3.18 Zeile 3) wird das Sprite mittig auf das Display positioniert.

Quellcode 3.18: Sprites

```
1 CCSIZE* pkDeviceSize = CCDirector::sharedDirector()->getWinSize();
2 CCSprite* pkSprite = CCSprite::spriteWithFile( "sprite.png");
3 pkSprite->setPosition( ccp( pkDeviceSize.width/2,
4                               pkDeviceSize.height/2) );
```

Optimierung:

In den meisten Spielen kommen, aufgrund des Performancegewinns, Spritesheets zum Einsatz. Der Vorteil der Spritesheets wird im folgenden Kapitel beschrieben.

3.8 Spritesheet

Cocos2D-x unterstützt Spritesheets, worunter man das Kombinieren von Einzelsprites zu einem großen Image versteht.

Sprite Sheets werden aus verschiedenen Gründen genutzt:

- Animation von 2D Images
- Reduzierung der Zugriffe auf Images (siehe Kapitel Performanceoptimierung)
- Bündelung des Images in ein File
- Konfigurationsmöglichkeiten auf Seiten des SpriteTools

Spritesheets können komfortabel mit dem Tool TexturePacker erstellt werden. Die Nutzung gestaltet sich einfach.

Die Vorteile dieses Tools liegen in den Optimierungsmöglichkeiten des Spritesheets.

Ist das Spritesheet mit diesem Tool erstellt worden, findet man zwei Files, ein Image und eine P-List Datei. Mittels des Singleton CCSpriteFrameCache werden alle Sprites im Cache gespeichert (Quellcode 3.19 Zeile 1) und können mit dem Namen abgerufen werden (Quellcode 3.19 Zeile 3 und 4).

Quellcode 3.19: Spritesheets nutzen

```

1 CCSpriteFrameCache::sharedSpriteFrameCache() -> addSpriteFramesWithFile( "Images/spritesheet.plist");
2
3 CCSprite* pkBackground = CCSprite::spriteWithSpriteFrame(
    CCSpriteFrameCache::sharedSpriteFrameCache() -> spriteFrameByName( "background.png"));

```

Optimierung:

Spritesheets reduzieren die OpenGL ES Texture-Bind Aufrufe auf einen Aufruf. Zudem sinkt der Speicherverbrauch der Sprites, denn ein 138x138 Sprite / Image kann nicht exakt in dieser Größe im Speicher abgelegt werden. Wie bekannt, arbeitet der Arbeitsspeicher mit 2^n , dass heißt 0, 1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024 und so weiter. Somit sollten die Texturen und die Sprites, auch in diesem Format gespeichert werden (2^n). Diese Methodik ist auch bekannt als "The Power of Two".

P-List Datei

In dieser Datei (Quellcode 3.20) sind die Einzelgrafiken des Spritesheets (Abb. 3.5) definiert, und es wird der Dateiname, die Position im Spritesheet, der Offset und die Größe gespeichert.

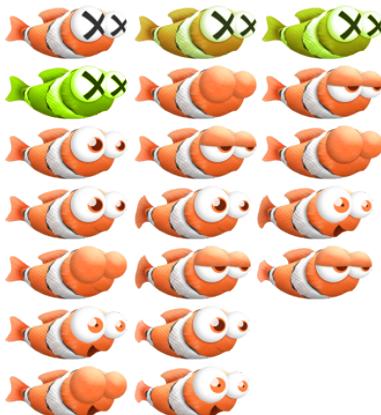


Abb. 3.5: Spritesheet - (Grafikquelle: Nurogames)

Mit dieser P-List Datei können, mittels der Koordinaten, aus dem Spritesheet-Image (spritesheet.png) die jeweiligen Einzelsprites geholt werden. Mit Hilfe des Namens kann, wie in (Quellcode 3.19 Zeile 3), das Sprite abgerufen werden.

Quellcode 3.20: Spritesheet.plist

```

1 <key>dessert_world_p_3.png</key>
2   <dict>
3     <key>frame</key>
4       <string>{{571,242},{104,39}}</string>
5     <key>offset</key>
6       <string>{0,0}</string>

```

```
7      <key>rotated</key>
8      <false/>
9  </dict>
```

3.9 Actions

Cocos2D-x bietet eine Auswahl von Actions an, um Elemente zu animieren (Position, Rotation, Skalierung, Sichtbarkeit und weitere). Da diese in CCNode implementiert sind, ist es möglich jedes Cocos2D-x Element mit Actions zu versehen. Diese können mit dem CCActionManager pausiert und fortgesetzt werden.

Folgende Unterklassen von CCAction können direkt genutzt werden:

- **CCFiniteTimeAction**

Action, die einen bestimmten Zeitraum andauern soll, und welche die Basisklasse der vorhandenen Actions ist

- **CCIInstantAction**

Action ohne Dauer ausführen

- **CCFollow**

Action, die einem Node folgt (alternative zu CCCamera)

- **CCSpeed**

Ändert die Geschwindigkeit einer Action

- **CCRepeatForever**

Wiederholt eine Action dauerhaft

- **CCSequence**

Definiert eine Reihenfolge von Actions die ausgeführt werden

- **CCReverseTime**

Action in umgekehrter Reihenfolge ausführen

- **CCTintTo / CCTintBy**

Sprites färben

Dabei muss zwischen Basisklassen unterschieden werden, da es für verschiedenen Actions unterschiedliche Basisklassen gibt.

Der folgenden Tabelle kann die jeweilige Basisklasse entnommen werden:

ActionEase	EaseBackIn, EaseBackInOut, EaseBackOut, EaseBounce, EaseElastic, EaseExponentialIn, EaseExponentialOut, EaseRateAction, EaseSineIn, CCEaseSineInOut, EaseSineOut, EaseExponentialOut
GridAction	FlipX3D, Lens3D, Liquid, Ripple3D, Shaky3D, Twirl, Waves, Waves3D, FadeOutTRTiles, JumpTiles3D, ShakyTiles3D, ShatteredTiles3D, ShuffleTiles, SplitCols, SplitRows, TurnOffTiles, WavesTiles3D
ActionInterval	Blink, FadeIn, FadeOut, FadeTo, JumpBy, MoveTo, RotateBy, ScaleTo, SkewTo, Spawn
ActionInstant	FlipX, FlipY, Hide, Place, ReuseGrid, Show, StopGrid, ToggleVisibility

Hinweis:

GridAction und ActionEase ist wieder Unterklasse von ActionInterval, was bedeutet, dass sie eine Dauer für die Ausführung der Action besitzen.

Es gibt also eine sehr große Auswahl an möglichen Actions, die auf Objekte ausgeführt werden können.

Exemplarisch ist im Folgenden eine Action der Klasse ActionInstant deklariert. Als Objekt, welches die Action ausführen soll, wird ein Sprite genutzt (*es können jedoch alle, von CCNode abgeleiteten Klassen animiert werden.*) (Quellcode 3.21 Zeile 1). Dieses Sprite wird mittig auf das Display positioniert (Quellcode 3.21 Zeile 2) und dem aktuellen Layer hinzugefügt.

In (Quellcode 3.21 Zeile 4) wird die Action CCMoveTo auf das Sprite ausgeführt. Wobei *actionWithDuration*, der erste Parameter, die Dauer beschreibt und der zweite Parameter die Position, an die sich das Sprite bewegen soll.

Quellcode 3.21: Action

```

1 CCSprite* pkSprite = CCSprite::spriteWithFile( "logo.png" );
2 pkSprite->setPosition( CCPoint( kDeviceSize.width/2,
3                                     kDeviceSize.height/2 ) );
4 addChild( pkSprite, 10 );
5 pkSprite->runAction( CCMoveTo::actionWithDuration( 5, CCPoint( 0, 0 ) ) );

```

3.10 Label und Font

Labels werden benötigt, um Text auf dem Display anzuzeigen. Cocos2D-x unterstützt dabei zwei Arten:

- **True Type Fonts** (TTF) mit der Klasse CCLabel
- **Texture Atlas Label** mit den Klassen CCLabelAtlas und CCBitmapFontAtlas

Die Unterschiede bei diesen zwei Arten liegen bei der Performance, der Erstellung und den Konfigurationsmöglichkeiten.

True Type Fonts

True Type Fonts sind einfach anzuwenden, weisen aber bezüglich der Performance erhebliche Nachteile auf. Die Erstellung und Erneuerung eines TTFLabel ist sehr teuer.

Ein Label (TTF) kann folgendermaßen hinzugefügt werden:

Quellcode 3.22: CCLabelTTF

```
1 CCLabelTTF* pkLabel = CCLabelTTF::labelWithString( "LabelText" ,
2                                         "arial" ,
3                                         16) ;
4 addChild( pkLabel );
```

Texture Atlas Label

Das Erstellen geht schneller vorstatten, als bei den True Type Fonts, da nicht jedesmal eine neue Textur erzeugt werden muss. Schriften können konfiguriert werden (Schatten, Verläufe und weitere). Der Nachteil liegt jedoch darin, dass zum Erstellen von Texture Atlas Labels Font Editoren benötigt werden.

Wie oben erklärt, muss bei Textured Atlas Labels zwischen CCLabelAtlas und CCBitmapFontAtlas unterschieden werden. Der Hauptunterschied liegt in dem Format, in welchem die Schrift gespeichert ist.

CCBitmapFontAtlas

Hier erfolgt die Speicherung der Schrift in einem FontFile **.fnt*.

Quellcode 3.23: CCBitmapFontAtlas

```
1 CCLabelBMFont* pkLabel = CCLabelBMFont::labelWithString("Hello World"
2                                         fntFile:@#bitmapFontTest.fnt") ;
2 addChild( pkLabel );
```

CCLabelAtlas

Hier erfolgt die Speicherung der Schrift in einem ImageFile **.png*.

Quellcode 3.24: CCLabelAtlas

```
1 CCLabelAtlas* pkLabel = CCLabelAtlas::labelAtlasWithString("Hello World",
2                                         "arial-charmap.png" , 48 , 64 , " ' ' ");
2 addChild( pkLabel );
```

Optimierung:

Ein Label -*CCLabelTTF* ist sehr langsam, denn bei jeder Textänderung, muss die Textur neu erstellt werden. Ein Negativbeispiel für die falsche Nutzung von *CCLabelTTF* ist ein Gamescore im Spiel. Dieser verändert kontinuierlich seinen Wert, was heißt, dass in sehr kurzer Zeit sehr oft die Textur neu erstellt wird. Dies führt zu merklichen Performanceeinbrüchen. Eine bessere Lösung ist *CCLabelBMFont* oder *CCLabelAtlas*. Hier ist die Textur in Form einer Bitmap beziehungsweise *PNG* bereits in die Applikation mit eingebunden und muss nicht jedesmal neu erstellt werden.

3.11 Menu

Cocos2D-x besitzt ein ausführliches Menü System. Ein Menü Item lässt sich mit verschiedenen Möglichkeiten erzeugen:

- Font
- Label
- AtlasFont
- Image
- Sprite

CCMenu

CCMenu ist die Oberklasse aller *CCMenu* Elemente und beeinhaltet die verschiedenen Item Elemente, welche mit *addChild* hinzugefügt werden können. Einem *CCMenu* kann ein *CCMenuItem* hinzugefügt werden (Quellcode 3.25).

CCMenuItem

Ein Menu Item kann auf verschiedene Möglichkeiten erzeugt werden.

CCMenuItem	<i>CCMenuItemSprite</i> - mit einem <i>Sprite</i>
	<i>CCMenuItemLabel</i> - mit einem <i>Label</i>
	<i>CCMenuItemImage</i> - mit einem <i>Image</i>
	<i>CCMenuItemAtlasFont</i> - mit Hilfe eines <i>Atlas Font</i>
	<i>CCMenuItemFont</i> - mit Hilfe eines <i>Font</i>

CCMenuItemImage

Hier wird ein MenuItem mit Image erzeugt. Es können drei verschiedene Images übergeben werden.

- unselected image
Nichts ist selektiert
- selected image
Selektiert
- disabled image
Deaktiviert

Quellcode 3.25: Menüs

```

1 CCSIZE kDeviceSize = CCDirector::sharedDirector()->getWinSize();
2
3 CCMenu* pkMenu = new CCMenu();
4 pkMenu->setPosition( CCPoint( kDeviceSize.width/2,
```

```

5                         kDeviceSize.height / 2) );
6
7 CCLabelTTF* pkLabel = CCLabelTTF::labelWithString( "Spiel starten",
8                                         "arial",
9                                         16);
10
11 CCMenuItemLabel* pkLabelItem = CCMenuItemLabel::itemWithLabel( pkLabel );
12
13 pkMenu->addChild( pkLabelItem, 10 );
14
15 addChild(pkMenu);

```

3.12 Touch Events

Mit einem Layer ist es möglich, auf Touchevents zuzugreifen. Es muss jedoch zwischen zwei Verfahren unterschieden werden.

- **Touch Target**

Verfahren für Multitouch, bei welchem es möglich ist, mehrere Toucheingaben zu verarbeiten

- **Touch Delegate**

Verfahren für Singletouch, bei welchem nur auf ein Touchereignis reagiert werden kann.

Hinweis:

Bei dem Touch Delegate Verfahren wird der aktuelle und der vorherige Touch-Punkt gespeichert.

Aktivieren der verschiedenen Verfahren

Die Aktivierung dieser beiden Verfahren unterscheidet sich. Für Touch Target (Quellcode 3.26) muss lediglich Touch aktiviert werden:

Quellcode 3.26: Touch aktivieren Touch Target

```

1 setIsTouchEnabled( true );
2 CTouchDispatcher::sharedDispatcher()->addTargetedDelegate( this , 0 , true );

```

Bei Touch Delegate (Quellcode 3.27) muss ein TouchDispatcher registriert werden:

Quellcode 3.27: Touch aktivieren Touch Dispatcher

```

1 setIsTouchEnabled( true );
2 CCTouchDispatcher::sharedDispatcher()->addStandardDelegate( this , 0 );

```

Die Methoden für die Touch-Events unterscheiden sich bei diesen beiden Verfahren auch durch die Methodennamen und die Parameter. Bei Multitouch muss eine Menge an Touchpunkten gespeichert werden.

Dabei können verschiedene Methoden, je nach Nutzung, überschrieben werden. ccTouchBegan wird aufgerufen, sobald sich der Finger auf dem Display befindet und ccTouchEnded wenn der Finger nicht mehr das Display berührt. Für jede Bewegung des Fingers ist ccTouchMove zuständig. In Fällen, bei denen der Touch unterbrochen wird, zum Beispiel bei einem eingehenden Anruf, wird ccTouchCancelled aktiv.

Touch Delegate

- Began - virtual bool ccTouchBegan(CCTouch* touch, CCEvent* event);
- Moved -virtual void ccTouchMoved(CCTouch* touch, CCEvent* event);
- Ended - virtual void ccTouchEnded(CCTouch* touch, CCEvent* event);
- Cancelled - virtual void ccTouchCancelled(CCTouch* touch, CCEvent* event);

Touch Target

- Began - virtual void ccTouchesBegan(CCSet* touches, CCEvent* event);
- Moved -virtual void ccTouchesMoved(CCSet* touches, CCEvent* event);
- Ended - virtual void ccTouchesEnded(CCSet* touches, CCEvent* event);
- Cancelled - virtual void ccTouchesCancelled(CCSet* touches, CCEvent* event);

3.13 Scheduler

Der Schedule dient zum wiederholten Ausführen spezieller Selector Methoden, welche als Parameter ein ccTime besitzen müssen.

Es muss zwischen zwei Typen von Selector unterschieden werden. Unter einem Selector, versteht man die wiederholt aufzurufende Methode.

- update selector
- custom selector

Update Selector

Der Update Selector (Quellcode 3.28) wird bei jedem Frame aufgerufen und es kann die Priorität (falls mehrere vorhanden) geändert werden. Die aufzurufende Methode ist in diesem Fall fest definiert und darf nicht geändert werden.

Quellcode 3.28: Update Selector

```
1 void GameScene::scheduleLoop()
2 {
```

```

3     scheduleUpdate();
4 }
5
6 void GameScene::update( ccTime delta)
7 {
8     //Code der wiederholt ausgefuehrt werden muss, zum Beispiel
9     //Kollisionserkennung
9 }
```

Custom Selector

Der Custom Selector wird standardmäßig jedes Frame aufgerufen, wobei aber auch ein eigenes Intervall angegeben werden kann. Beim Starten des Schedulers gibt es überladene Methoden. In Quellcode 3.29 Zeile 5 wird der Schedule Selector ohne Parameter hinzugefügt.

In Quellcode 3.29 Zeile 6 gibt es einen zusätzlichen float Parameter, der das Intervall angibt. Angenommen man hat ein Intervall von 0.1f, so bedeutet dies, dass jede 1/10 Sekunde die Schedule Methode aufgerufen wird.

Der letzte Parameter (Quellcode 3.29 Zeile 7) gibt an, ob die Schedule Methode weiterhin aufgerufen werden soll, oder ob sie pausiert ist.

Hinweis:

Wenn möglich, sollte der Custom Selector vermieden werden, da der Update Selector schneller ist und weniger Speicher verbraucht.

Quellcode 3.29: Custom Selector

```

1 void GameScene::scheduleLoop()
2 {
3     bool bIsPaused = true;
4
5     //schedule( scheduleselector( GameScene::mainLoop));
6     //schedule( scheduleselector( GameScene::mainLoop, 0.5f));
7     schedule( scheduleselector( GameScene::mainLoop, 0.5f, bIsPaused));
8 }
9
10 void GameScene::mainLoop( ccTime delta)
11 {
12     //Code der wiederholt ausgefuehrt werden muss, zum Beispiel
13     //Kollisionserkennung
13 }
```

3.14 Ankerpunkt (Anchor Point)

Jedes Element in Cocos2D-x besitzt einen Ankerpunkt, siehe (Kapitel 3.1 Node). Wenn ein Cocos Objekt (Layer, Menu, Sprite, oder ähnliches) positioniert werden soll, ist der Ankerpunkt und die Position wichtig. Da beide Punkte in direktem Zusammenhang stehen und die Ausrichtung und den Rotationsspunkt bestimmen, ist es wichtig, Ankerpunkte zu verstehen.

Ankerpunkte können folgende x und y Werte besitzen:

$$0 \leq x \leq 1 \text{ mit } x \in \mathbb{R} \text{ und } 0 \leq y \leq 1 \text{ mit } y \in \mathbb{R}$$

Verschiedene Ankerpunkte mit ihren Positionen sind in Abbildung 3.6 zu sehen.

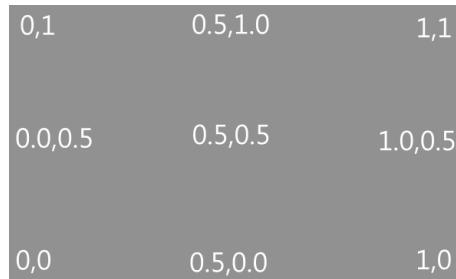


Abb. 3.6: Position der Ankerpunkte

Der Quellcode 3.30 zeigt ein Sprite, welches den Ankerpunkt bei (0,0) und die Position bei (0,0) hat.

Quellcode 3.30: Sprite Position und Ankerpunkt bei 0

```

1 CCSprite *pkSprite = CCSprite::spriteWithFile( "spriteImage.png" );
2 pkSprite->setPosition( CCPoint( 0, 0 ) );
3 pkSprite->setAnchorPoint( CCPoint( 0, 0 ) );
4 addChild( pkSprite );

```

Rotation in Verbindung mit Ankerpunkten

Die Rotation um einen Ankerpunkt, lässt sich mit einem Blatt gut veranschaulichen. Man legt den Finger einmal mittig auf das Blatt und dreht es. Danach legt man den Finger an ein Eck und dreht das Blatt. So können die Unterschiede der Rotation in Verbindung mit dem Ankerpunkt erkannt werden (der Finger ist hierbei der Ankerpunkt). Die folgende Abbildung (Abb. 3.7) zeigt beispielhaft die Rotation an verschiedenen Ankerpunkten.

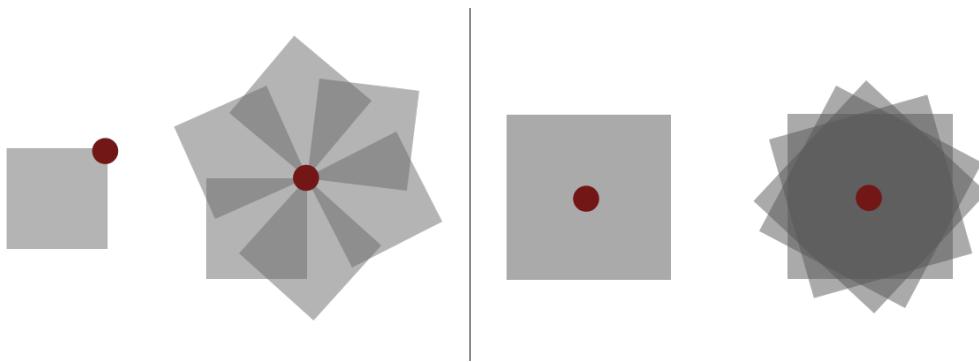


Abb. 3.7: Rotation um Ankerpunkte (links bei 1,1) und (rechts bei 0,5,0,5)

Position in Verbindung mit Ankerpunkten

Nicht nur bei der Rotation, sondern auch bei der Positionierung von Objekten sind Ankerpunkte relevant. Da die Positionierung wie in Quellcode 3.31

Quellcode 3.31: Sprite Position und Ankerpunkt bei 0

```
1 pkSprite->setPosition( CCPoint( 0 , 0 ));
```

immer relativ zu einem Ankerpunkt ist, muss beachtet werden an welcher Stelle sich der Ankerpunkt befindet. Die folgende Abbildung (Abb. 3.8) zeigt ein Beispiel für die Positionierung, in Bezug auf Ankerpunkte.

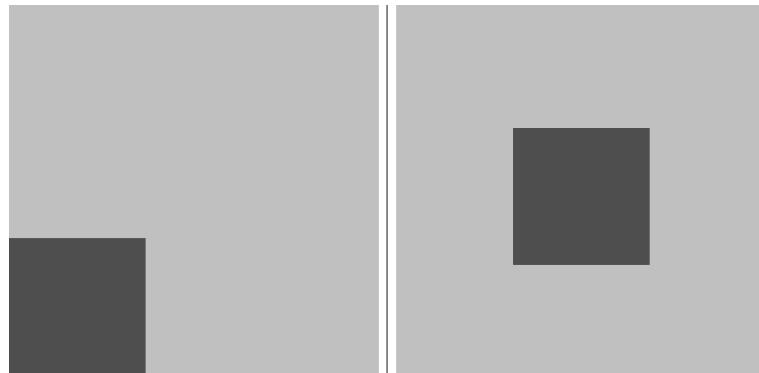


Abb. 3.8: Positionierung in Bezug auf Ankerpunkte des Layers (links bei 0,0) und (rechts bei 0,5,0,5)

3.15 Accelerometer

Seit Einführung der Smartphones besitzen viele mobile Geräte einen Accelerometer (Beschleunigungssensor). Dieser wird zum Beispiel genutzt, wenn das Gerät zwischen horizontaler und vertikaler Ausrichtung gedreht wird. Der Beschleunigungssensor erkennt diese Bewegung und passt die Darstellung an.

Auch bei Spielen kann dies für eine Gameplay Steuerung genutzt werden. So kann zum Beispiel ein Lenkrad imitiert werden, sobald das Gerät nach links oder rechts geneigt wird.

Um den Accelerometer zu nutzen, muss dieser aktiviert werden (Quellcode 3.32 Zeile 1).

Danach muss die Accelerometer Methode überschrieben werden. (Quellcode 3.32 Zeile 3).

Somit besteht die Möglichkeit, für alle drei Achsen (x, y, z) die Beschleunigungswerte auszulesen (Quellcode 3.32 Zeile 6 - 8).

So kann zum Beispiel ein Sprite (zum Beispiel ein Raumschiff) bewegt werden.

Quellcode 3.32: Accelerometer

```

1 setIsAccelerometerEnabled( true );
2
3 void TestScene::didAccelerate( CCAcceleration* pkAccelerationValue )
4 {
5
6     float fXAccelerationValue = pkAccelerationValue.x;
7     float fYAccelerationValue = pkAccelerationValue.y;
8     float fZAccelerationValue = pkAccelerationValue.z;
9
10    if ( fYAccelerationValue > 0.25 )
11    {
12        //Bewege Sprite nach rechts
13    }
14
15    if ( fYAccelerationValue < -0.25 )
16    {
17        //Bewege Sprite nach links
18    }
19 }
```

3.16 Singleton

Cocos nutzt in verschiedenen Bereichen das Singleton, welches zu der Kategorie der Creational Patterns¹ gehört. Daher ist es wichtig, etws näher auf Singletons einzugehen. Durch Singletons besteht die Möglichkeit nur eine Instanz eines Objektes zu erzeugen und dies auch sicherzustellen. Wenn das Objekt bereits existiert, wird dies zurückgegeben, ansonsten neu erzeugt. Somit ist das Singleton von jeder Klasse aus erreichbar, ohne dass eine neue Instanz erzeugt werden muss. Man kann sich dies wie eine globale Klasse vorstellen.

Cocos nutzt folgende Singletons:

- CCActionManager
- CCDirector
- CCSpriteFrameCache

¹Entwurfsmuster zur Erzeugung von Objekten

- CCTextureCache
- CCTouchDispatcher
- CCAudioManager
- SimpleAudioEngine

Singletons bieten auch bei eigener Implementierung Vorteile. Ein Anwendungsbeispiel wäre zum Beispiel ein GameManager, welcher klassenübergreifend spielspezifische Funktionen, wie Sound oder Ähnliches, verwaltet.

Im Folgenden wird ein Beispiel für die Implementierung eines Singleton in C++ gezeigt (Quellcode 3.33 und 3.34).

Quellcode 3.33: singleton.h

```

1 #ifndef _SINGLETON_H_
2 #define _SINGLETON_H_
3
4 class Singleton{
5     public:
6
7         static Singleton* getInstance();
8
9         ~Singleton()
10    {
11         instanceFlag = false;
12    }
13
14    private:
15        static bool instanceFlag;
16        static Singleton *single;
17        Singleton()
18    {
19         // private constructor
20    }
21 };
22 #endif

```

Quellcode 3.34: singleton.cpp

```

1 #include "Singleton.h"
2
3 bool Singleton::instanceFlag = false;
4 Singleton* Singleton::single = NULL;
5
6 Singleton* Singleton::getInstance()
7 {
8
9     if (! instanceFlag)

```

```

10     {
11         single = new Singleton();
12         instanceFlag = true;
13
14     return single;
15 }
16 else
17 {
18     return single;
19 }
20 }
```

3.17 Sounds / Audio

Um Sounds in Cocos2D-x zu nutzen, wird die CocosDehension Library benötigt. CocosDehension besteht aus drei Hauptunterklassen:

- **SimpleAudioEngine** Sehr eingeschränkte Funktionalität, für die meisten Mobile Games jedoch ausreichend
- **CDAudioManager** Mehrfachwiedergabe von Musik
- **CDSoundEngine** Das größte Soundpaket

Auf den verschiedenen Plattformen werden unterschiedliche Dateiformate benötigt.

- **Android** .ogg, .wav
- **iPhone** .mp3, .cav
- **Win32**.mid, .wav

Man kann eine grobe Unterscheidung zwischen Hintergrundmusik und Effekten vornehmen (Quellcode 3.35).

Quellcode 3.35: Nutzung der SimpleAudioEngine in Cocos2D-x

```

1 CocosDehension::SimpleAudioEngine::SharedEngine()->playBackgroundMusic( "musik.wav", true);
2 CocosDehension::SimpleAudioEngine::SharedEngine()->pauseBackgroundMusic();
3 CocosDehension::SimpleAudioEngine::SharedEngine()->resumeBackgroundMusic()
;
4 CocosDehension::SimpleAudioEngine::SharedEngine()->playEffekt( "effekt.wav" );
```

3.18 Fazit

In diesem Kapitel sind die Grundlagen von Cocos2D-x erklärt. Mit diesen sollte es möglich sein ein Spiel zu implementieren . Es gibt noch weitere Cocos Klassen, welche einem die Arbeit erleichtern. Diese können jedoch nicht mehr als Grundlagen angesehen werden. Für genauere Informationen empfiehlt sich die Cocos2D-x Seite. Dort findet man eine Dokumentation, ein Forum und eine große Menge an Anleitungen.

Im Grunde wurde fast das komplette Spektrum von Cocos2D-x durch diese Grundlagen abgedeckt. Wenige sehr spezielle, welche hier nicht beschrieben sind, sind:

- Keyboard
- Draw Primitive
- Scrolling backgrounds (parallax)
- Partikelsystem

Sowie die Third-Party Erweiterungen für Cocos2D-x:

- Chipmunk
- Box2D

4 Entwicklungsrelevantes in Cocos2D-x

Neben den Grundlagen, welche in Kapitel 3 erklärt sind, gibt es noch weitere, relevante Bereiche. Diese Bereiche sind sehr weit gefächert. Von Libraries über native Anwendungen bis hin zu OpenGL und Makefiles. Die wichtigsten sind in diesem Kapitel angesprochen.

4.1 Native Development Kit

Das Android Native Development Kit, kurz NDK, ermöglicht es, nativen Code wie C und C++ in Java zu nutzen.

Wenn man nativen Code zum Programmieren nutzt, wird dieser in das .apk gepackt und läuft dann auf der virtuellen Maschine auf dem Gerät.

Um eine native Anwendung unter Android lauffähig zu machen, wird das NDK und eine Unix Umgebung benötigt, welche unter Windows zum Beispiel Cygwin ist. Dieser native Programmcode muss im Eclipse / Android Projekt als Java Build Path - Source hinzugefügt werden. Mittels Java Native Interface besteht die Möglichkeit zwischen den beiden Sprachen C++ und Java zu kommunizieren.

Im Falle von Cocos2D-x ist dies erforderlich, weil das Cocos2D-x Framework in C++ geschrieben ist und auch genutzt wird. Andere Anwendungsfälle sind bereits vorhanden Spiele, die mit C++ programmiert wurden. Diese können ohne Übersetzen in Java für Android lauffähig gemacht werden.

In einigen Fällen bietet das NDK auch Performancevorteile, welche den Nutzen des NDK sinnvoll machen.

4.2 JNI

Java Native Interface ist eine Schnittstelle (Abb. 4.1), mit der es möglich ist von C / C++ Java Methoden aufzurufen und umgekehrt. Da JNI sehr umfangreich ist, werden im Folgenden die Grundlagen, die in Verbindung mit Cocos2D-x benötigt werden, beschrieben.

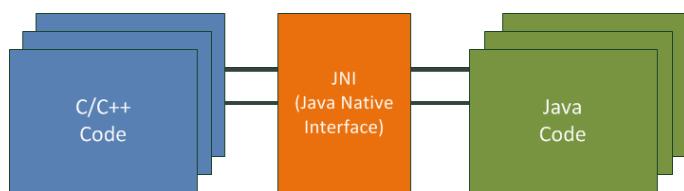


Abb. 4.1: JNI als Bindeglied zwischen Java und C/C++

Es wird zwischen folgenden beiden Aufrufen unterschieden:

- Aufruf einer Java Methode von C / C++ aus
- Aufruf einer C/C++ Methode von Java aus

Ein einfaches Beispiel, welches den Nutzen des NDK und JNI erklärt:

Cocos2D-x besitzt keinen Browser. Möchte man jedoch auf Basis von Cocos2D-x ein Browserspiel implementieren, ist es notwendig, dass Cocos2D-x auf dem jeweiligen Android Gerät den Browser implementiert und darauf auch Zugriff anbietet. Dieser Zugriff von der C++ Seite hin zu Java (Browser) erfolgt mittels JNI. Dies stellt nur ein Beispiel für die Nutzung des JNI dar.

4.2.1 Aufruf einer Java Methode von C/C++ aus

Bei einem Aufruf einer Java Methode von C++ aus, kommt eine JNI Hilfsklasse zum Einsatz. Mit dieser ist es möglich Java Funktionen aufzurufen.

Folgende Informationen werden benötigt:

- Methodename
- Klassenname und Package, in dem die Klasse liegt
- Rückgabetyp
- Parametertyp(en)

Für Typen (*Feldbezeichner / Field Descriptor*) bietet JNI eigene "Codierungen" an. Tabelle 4.1 zeigt die, in Quellcode 4.1 Zeile 21, eingesetzten verschiedenen JNI-Typen.

Feldbezeichner	Java Typ
Z	boolean
B	byte
C	char
S	short
I	int
J	long
F	float
D	double

Table 4.1: Feldbezeichner

Feldbezeichner von Referenztypen, wie zum Beispiel der Klasse String, beginnen mit einem *L* gefolgt von dem Klassenbezeichner/Klassenname.

Beispiel:

Die Klasse String liegt in dem Package Java.lang.String, der Feldbezeichner für die String Klasse wäre dann:

Ljava/lang/String

Der vierte Parameter in Quellcode 4.1, Zeile 21 beschreibt, sowohl den Methodentyp, als auch die Parameter. Parameter werden in "()" eingeschlossen und mit einem ";" , falls die Methode mehrere Parameter

besitzt, getrennt.

”(Parameter1;Parameter2;)Rückgabetyp”

In Quellcode 4.1, Zeile 32 wird diese Methode, die mit Hilfe der JNI Hilfsklasse ermittelt wurde, aufgerufen und erhält den String Parameter. Dieser muss vor dem Aufruf in einen jstring umgewandelt werden, denn Java kann den C++ String nicht interpretieren.

Quellcode 4.1: Java ruft C++ auf

```

1 JavaClass::JavaClass( const String& rkPackageName,
2                               const String& rkClassName)
3 :
4     m_kFullClassName()
5 {
6     m_kFullClassName = rkPackageName + "/" + rkClassName;
7 }
8
9 JavaClass::~JavaClass()
10 {
11 }
12
13 void JavaClass::callMethod( const String& rkMethodName,
14                               const String& rkParameter)
15 {
16     JniMethodInfo kJniMethodInfo;
17
18     if ( !JniHelper::getStaticMethodInfo( kJniMethodInfo,
19                                         m_kFullClassName,
20                                         rkMethodName,
21                                         "(Ljava/lang/String;)V"))
22     {
23         msg::error( "JavaClass::callMethod",
24                     "Failed to get method '" + rkMethodName +
25                     "' on class '" + m_kFullClassName + "'!");
26
27         return;
28     }
29
30     jstring kParameter = kJniMethodInfo.env->NewStringUTF( rkParameter);
31     kJniMethodInfo.env->CallStaticVoidMethod( kJniMethodInfo.classID,
32                                               kJniMethodInfo.methodID,
33                                               kParameter);
34
35     kJniMethodInfo.env->DeleteLocalRef( kParameter);
36     kJniMethodInfo.env->DeleteLocalRef( kJniMethodInfo.classID);
37 }
```

4.2.2 Aufruf einer nativen Methode von Java aus

Um eine C/C++ Funktion von Java aus aufzurufen, muss diese zuerst in Java deklariert werden. Zudem muss die Library, in der sich die native Funktion befindet, geladen werden. Dies wird mittels dieser beiden Codezeilen realisiert.

Quellcode 4.2: Native Funktion in Java

```

1 public static void Load()
2 {
3     System.loadLibrary("library");
4 }
5
6 public static native void nativeMethod( String sParameter);

```

In Quellcode 4.2, Zeile 3 ist zu sehen, dass mittels loadLibrary die native Library geladen wird. Dies ermöglicht es Java diese zu nutzen. In Quellcode 4.2, Zeile 6 wird die native Funktion deklariert (native durch das Schlüsselwort native). Somit weiß Java, dass sich diese native Methode in der geladenen Library befindet.

Auf der C++ Seite muss die native Funktion, die von Java aus aufgerufen wird, existieren. Dabei folgt die Methodendeklaration einem bestimmten Schema, wie in Quellcode 4.3, Zeile 3 zu erkennen ist.

Quellcode 4.3: Native Funktion in C++

```

1 extern "C"
2 {
3 JNIEXPORT void Java_Packaged_Classname_Methodname(
4
5
6
7
8     const char *spParameter;
9     spParameter = env->GetStringUTFChars ( kParameter, false );
10    // spParameter nutzen
11    env->ReleaseStringUTFChars( kParameter, spParameter );
12 }
13 }
```

4.3 Editoren / Tools für Cocos2D-x

In der Spieleentwicklung werden, abgesehen von Sourcecode, natürlich auch Grafiken, Partikeleffekt, Physik, Fonts, Sounds und weitere Elemente benötigt. Für jedes dieser Elemente gibt es nützliche Editoren.

- Action Editor
- Map Editor
- Physic Object Editor
- Font Editor
- Spritesheet Editor
- Particle Editor
- Textur Atlas Editor

4.4 OpenGL ES

Open Graphics Library for Embedded Systems ist eine vereinfachte Version von OpenGL und speziell für eingebettete Systeme optimiert. Durch das Weglassen redundanter Operationen, ist OpenGL ES speziell für Geräte mit weniger Rechenleistung optimiert.

Es kommt in den verschiedensten Bereichen, von der Playstation 3 über den PDA, bis hin zu Smartphones, vor. Auch Cocos2D-x nutzt OpenGL ES.

4.5 Präprozessormacros

Präprozessormacros sollten vermieden werden, da sie schwer zu debuggen sind. Speziell Cocos2D-x Macros sollten durch ein C++ Äquivalent ersetzt werden. So kann ein CC_FOR_EACH in den meisten Fällen durch einen Iterator oder eine For-Schleife ersetzt werden.

5 Erweiterung der Nuroengine-X

Die, in diesem Kapitel implementierten, Funktionen sind nicht Bestandteil von Cocos2D-x, sondern erweitern Cocos2D-x und die firmeninterne Nuroengine-X, welche auf Cocos2D-x basiert. Während der Entwicklung von Spielen treten des öfteren Punkte auf, welche das Framework nicht unterstützt. Solche Punkte müssen eigens implementiert werden.

Für die Engine werden folgende Funktionen implementiert:

- Gestenerkennung (InputHandlingLib)
- ScrollMenu (GuiLib)
- Webview (WebViewLib)
- WebRequest (WebRequestLib), worauf nicht näher eingegangen wird

Alle implementierten Libraries sind Cross-Plattformfähig und basieren auf Cocos2D-x.

5.1 Gestenerkennung (GestureRecognizer)

Die Library wurde speziell für die Gestenerkennung implementiert. Sie dient als Grundlage für das ScrollMenu und kann auch für eine Gameplaysteuerung genutzt werden.

Die Gestenerkennung erfolgt in Cocos2D-x nur rudimentär (siehe Kapitel 3.1.9 Touch Events), das heißt, dass vier verschiedene Zustände vorhanden sind, welche erkannt werden können:

- | | |
|--------------|------------------|
| • TouchBegan | • TouchEnded |
| • TouchMoved | • TouchCancelled |

Jedoch kann nicht erkannt werden, ob eine spezielle Geste ausgeführt wird. Um diese erweiterte Gestenerkennung an Cocos2D-x zu koppeln, ist diese Library notwendig.

5.1.1 Implementierte Funktionen der Gestenerkennung

- Swiperichtung (links, rechts, hoch, runter)
- Swipewinkel
- Swipedistanz auf x-Achse
- Swipedistanz auf y-Achse

- Swipedistanz achsenunabhängig
- Bewegungsdistanz
- Bewegungsgeschwindigkeit

5.1.2 Swipeerkennung / Swiperichtung

Grundlage für die Erkennung von Gesten sind Touchpunkte, die der Benutzer mit seinem Finger angegeben hat. Die Gestenerkennung basiert auf zwei Touchpunkten [Touch1 (x, y)] und [Touch2 (x, y)]. Anhand dieser zwei Touchpunkte kann die Bewegungsrichtung berechnet werden.

Es folgt ein Berechnungsbeispiel anhand zweier Touchpunkte t1(10,10) und t2(50, 10). Folgende Formel bestimmt die Bewegungsrichtung:

$$t1(x) - t2(x) = \text{Bewegungswert}X.$$

$$t1(y) - t2(y) = \text{Bewegungswert}Y.$$

Wenn der Bewegungswert < 0 ist, war der Swipe nach links auf der X-Achse, beziehungsweise nach unten auf der Y-Achse. Wenn er größer war, dann nach rechts auf der X-Achse, beziehungsweise nach oben auf der Y-Achse.

Quellcode 5.1: Gesten nach Links prüfen

```
1 const bool GestureRecognizer::isGestureSwipeLeft() const
2 {
3     return m_kStartTouchPoint.x > m_kEndTouchPoint.x;
4 }
```

Da in diesem Beispiel der Bewegungswert -40 ist, also < 0 , ist die Bewegungsrichtung nach links zu sehen, unter (Quellcode 5.1). Analog kann so jede Richtung bestimmt werden:

$$f(\text{Bewegungswert}X) = \begin{cases} \text{links} & \text{für } \text{Bewegungswert}X < 0 \\ \text{rechts} & \text{für } \text{Bewegungswert}X > 0 \end{cases}$$

$$f(\text{Bewegungswert}Y) = \begin{cases} \text{runter} & \text{für } \text{Bewegungswert}Y < 0 \\ \text{hoch} & \text{für } \text{Bewegungswert}Y > 0 \end{cases}$$

5.1.3 Swipewinkel

Die Berechnung des Winkels (Quellcode 5.2) erfolgt mittels der Formel:

$$\text{angleRadians} = \arctan 2(x1 - x2, y1 - y2)$$

Quellcode 5.2: Winkel des Swipe berechnen

```
1 const float GestureRecognizer::swipeAngle() const
```

```

2 {
3     float fAngleRadians = atan2( m_kStartTouchPoint.x -
4                                     m_kEndTouchPoint.x,
5                                     m_kStartTouchPoint.y -
6                                     m_kEndTouchPoint.y);
7
8     return fAngleRadians * (180.0f / 3.14159265f);
9 }
```

5.1.4 Swipe Distanz

Die Distanz eines Swipes ist bei Spielen ein wichtiger Faktor, wenn die Gestenerkennung zum Beispiel als Gameplay Steuerung eingesetzt wird. So kann unter Umständen die Swipe-Distanz als Aussgangspunkt für die Wurfweite oder ähnliches genutzt werden.

Es werden folgende Methoden implementiert:

- Distanz auf der X-Achse (Quellcode 5.3)
- Distanz auf der Y-Achse (Quellcode 5.4)
- Distanz unabhängig von den Achsen (Quellcode 5.5)

Die Ergebnisse der Berechnungen sind jeweils als Pixel (float) zu verstehen. Da eine Distanz nicht negativ sein kann, wurde die Methode so realisiert, dass jeweils der absolute Wert berechnet wird.

Quellcode 5.3: Swipe-Distanz auf der X-Achse

```

1 const float GestureRecognizer::swipeDistanceX() const
2 {
3     return ( abs ( m_kEndTouchPoint.x - m_kStartTouchPoint.x ) );
4 }
```

Quellcode 5.4: Swipe-Distanz auf der Y-Achse

```

1 const float GestureRecognizer::swipeDistanceY() const
2 {
3     return ( abs ( m_kEndTouchPoint.y - m_kStartTouchPoint.y ) );
4 }
```

Die Berechnung der Distanz, unabhängig von der Achse, wird mittels dieser Formel realisiert:

$$\overline{AB} = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

mit:

$x_2 = m_kEndTouchPoint.x$

$x_1 = m_kStartTouchPoint.x$

```
y2 = m_kEndTouchPoint.y
y1 = m_kStartTouchPoint.y
```

Hier werden erneut zwei Touchpunkte vorausgesetzt, anhand welcher die Berechnung der Distanz stattfindet.

Quellcode 5.5: Swipe Distanz unabhängig von Achsen

```
1 const float GestureRecognizer::distanceInSwipeDirection() const
2 {
3     return sqrt(float(pow( m_kEndTouchPoint.x - m_kStartTouchPoint.x, 2) +
4                     pow( m_kEndTouchPoint.y - m_kStartTouchPoint.y, 2)))
5 }
```

5.1.5 Bewegungsdistanz

Im Gegensatz zu der Swipedistanz, die nur die direkte Strecke zwischen Touch-Start und Touch-Ende berechnet, berechnet diese Methode (Quellcode 5.6) die komplette Strecke (siehe Abb. 5.1).

Die Berechnung erfolgt dabei durch jeweils zwei Punkte, in diesem Fall dem letzten und dem aktuellen. Aus diesen zwei Werten wird die Distanz berechnet und zu der Gestamtlänge addiert. Danach wird der aktuelle Touchpunkt als vorheriger gesetzt.

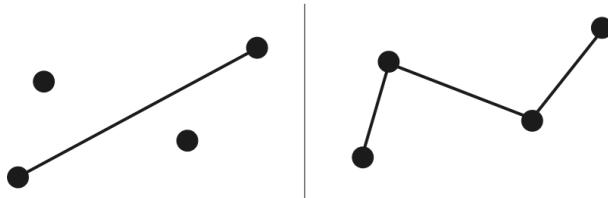


Abb. 5.1: Vergleich Swipedistanz (links) - Bewegungsdistanz (rechts)

Quellcode 5.6: Bewegungsdistanz

```
24 const float GestureRecognizer::distanceMultiple() const
25 {
26     return m_fDistanceMultiple;
27 }
28
29 void GestureRecognizer::addMultipleDistance()
30 {
31     m_fCurrentDistance = sqrt( float( pow( m_kMoveTouchPoint.x -
32                                         m_kPreviousTouchPoint.x, 2) +
33                                         pow( m_kMoveTouchPoint.y -
34                                         m_kPreviousTouchPoint.y, 2)))
35 ;
36     m_fDistanceMultiple += m_fCurrentDistance;
37     m_kPreviousTouchPoint = m_kMoveTouchPoint;
```

38 }

5.1.6 Bewegungsgeschwindigkeit

Für die Berechnung der Geschwindigkeit (Quellcode 5.7) wird ebenfalls die Bewegungsdistanz genutzt. Hierbei wird, solange noch keine Sekunde vergangen ist, ein Distanzabschnitt, der während der Bewegungsdistanz gespeichert wird, auf die Geschwindigkeit addiert.

Quellcode 5.7: Distanzabschnitt hinzufügen

```

1 void GestureRecognizer::addMultipleDistance()
2 {
3     m_fCurrentDistance = sqrt( float( pow( m_kMoveTouchPoint.x -
4                                         m_kPreviousTouchPoint.x, 2) +
5                                         pow( m_kMoveTouchPoint.y -
6                                         m_kPreviousTouchPoint.y, 2) ) );
7
8     m_fDistanceMultiple += m_fCurrentDistance;
9     m_kPreviousTouchPoint = m_kMoveTouchPoint;
10 }
```

Die Geschwindigkeit (Quellcode 5.8) ergibt sich somit aus der Strecke, die in einer Sekunde zurück gelegt wird. Ist eine Sekunde vergangen, wird die Geschwindigkeit wieder auf null zurückgesetzt und der Timer beginnt wieder von vorne.

Quellcode 5.8: Geschwindigkeit ermitteln

```

1 void GestureRecognizer::update()
2 {
3     if ( !m_kTime.isPlaying() )
4     {
5         m_kTime.start();
6     }
7
8     if ( m_kTime.elapsed().seconds() <= 1 )
9     {
10         float fTempDistance= m_fCurrentDistance;
11         m_fSpeedTemp += fTempDistance;
12     }
13
14     if ( m_kTime.elapsed().seconds() >= 1 )
15     {
16         m_fSpeed = m_fSpeedTemp;
17         m_kTime.reset();
18         m_kTime.start();
19         m_fSpeedTemp = 0;
20         m_fCurrentDistance = 0;
```

```

21     }
22 }
```

So kann, zu jedem gewünschten Zeitpunkt, die Geschwindigkeit ausgelesen werden.

Quellcode 5.9: Geschwindigkeit auslesen

```

1 const float GestureRecognizer::multipleSwipeSpeed()
2 {
3     return m_fSpeed;
4 }
```

5.1.7 Umwandlung der Touch Koordinaten

In China ist es üblich, dass der Koordinatenursprung links oben liegt. Dies ist zum Beispiel auch bei der Wii der Fall. Da dies die Berechnung, in Verbindung mit Touch Koordinaten komplizierter macht, bietet Cocos2D-x eine Methode (Quellcode 5.10) an, um den Koordinatenursprung so umzuwandeln, dass der Ursprung links unten liegt.

Quellcode 5.10: Koordinaten umwandeln

```

1     m_kStartTouchPoint = pkTouch->locationInView( pkTouch->view());
2     m_kStartTouchPoint = CCDirector::sharedDirector()->convertToGL(
3                                     m_kStartTouchPoint)
4                                     ;
```

Nach dem Aufruf der convertToGL Methode, sind die Positionen an den Koordinatenursprung links unten angepasst.

5.1.8 Nutzung der Gesteinklasse

Die Klasse GestureRecognizer bietet Methoden (Quellcode 5.11) für den Zugriff auf die wichtigsten Funktionen an. Als Eingabe erwartet die Klasse zwei Touchpunkte. Einen [start] und [end] Punkt. Für die Bewegungsgeschwindigkeit und die Bewegungsdistanz wird der Touch [move] Punkt benötigt. Mit Hilfe dieser Punkte kann die Klasse, wie oben beschrieben, alle nötigen Berechnungen durchführen.

Quellcode 5.11: Nutzung der Gesteinklasse

```

1 #include "GestureRecognizer.h"
2 using namespace InputHandling;
3
4 m_pkGesture = new GestureRecognizer( );
5
6 // Aufruf in der ccTouchBegan Methode
7 m_pkGesture->startTouchPoint( pTouch);
8
9 // Aufruf in der ccTouchUpInside Methode
```

```

10 m_pkGesture->endTouchPoint( pTouch );
11 float fAngle = m_pkGesture->swipeAngle();
12 float fSwipeDistanceX = m_pkGesture->swipeDistanceX();
13 float fSwipeDistanceY = m_pkGesture->swipeDistanceY();
14 float fSwipeDistance = m_pkGesture->swipeDistance();
15 int nSwipeDirection = m_pkGesture->swipeDirection( true );
16
17 // Aufruf in der ccTouchMoved Methode
18 m_pkGesture->moveTouchPoint( pTouch );
19 m_pkGesture->addMultipleDistance();
20 float fDistanceMultiple = m_pkGesture->distanceMultiple();
21 float fSpeed = m_pkGesture->multipleSwipeSpeed();

```

5.2 Scrollmenu (gestenbasiert)

Das Scrollmenu basiert auf Gesten, welche hier speziell die Wischgeste (engl: swipe) für mobile Endgeräte betrifft. Grund für die Implementierung dieser Library war der Bedarf an einem scrollbaren Menu für ein Shop (In-Game-Items) und die Levelauswahl eines Spiels.

Zudem wurde ein Beschleunigungseffekt in das Shopsystem integriert, welches das Menu je nach Gestengeschwindigkeit beschleunigt und nachscrollen lässt.

Die ScrollMenu Library nutzt die, im Kapitel vorher beschriebene, GestureRecognizer Library. Auf Abbildung 5.2 ist die Nutzung des Scroll Menüs dargestellt.



Abb. 5.2: Anwendung des ScrollMenüs in einem Spiel - (Quelle: Nurogames)

5.2.1 Scrollen

In der TouchMoved Methode wird die Differenz zwischen StartTouch und MoveTouch berechnet. Diese Differenz ist Grundlage für die Bewegung (das Scrollen) durch das Menü. Wird die Bewegung von rechts

nach links ausgeführt, ist dieser Wert < 0 und das Menü wird um den Wert nach links bewegt (Abb. 5.3).

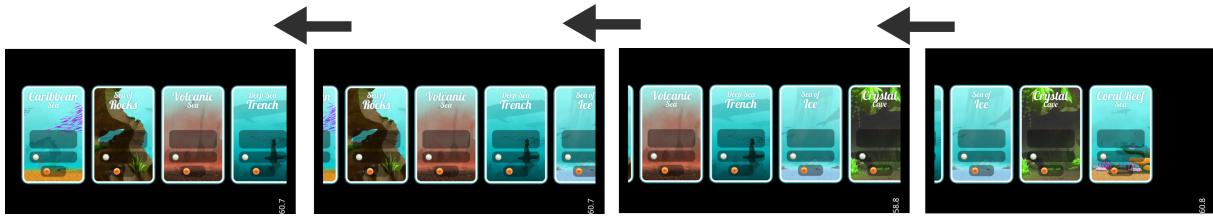


Abb. 5.3: Darstellung beim Scrollen des Menüs nach links

Wie bei Smartphones üblich, wird ein **Nachscrolleffekt** implementiert. Unter einem Nachscrolleffekt versteht man die Verlangsamung der Bewegung des Menüs nachdem der Finger beim Scrollen vom Display genommen wurde.

Für die Reduktion der Menüscrollgeschwindigkeit ist der Velocitywert und ein Timer verantwortlich. Der Timer multipliziert bei jedem Tick (hier bei jedem Frame) die Geschwindigkeit mit dem Verlangsamungsfaktor (Quellcode 5.12 Zeile 20).

Wenn der Velocitywert (`kDifferencePoint`) < 1 ist (Quellcode 5.12, Zeile 3), wird er auf 0 reduziert und das Menu bleibt stehen.

Der Nachscrolleffekt kann jedoch beim Touch auf das Display beendet werden, um somit eine Auswahl möglich zu machen. Damit das Menu nicht mit zu hoher Geschwindigkeit nachscrollt, wird ein `fMaxSpeed` festgelegt. Dieser Wert wird bei Bedarf gesetzt, falls die Geschwindigkeit die Maximalgeschwindigkeit überschreitet.

Quellcode 5.12: Nachscrolleffekt

```

1 void ScrollMenu::accelerateItems( ccTime kDeltaTime)
2 {
3     if ( fabs( m_kDifferencePoint.x ) < 1 )
4     {
5         m_kDifferencePoint.x = 0;
6     }
7
8     if ( fabs( m_kDifferencePoint.y ) < 1 )
9     {
10        m_kDifferencePoint.y = 0;
11    }
12
13    m_kDifferencePoint.x = clamp( m_kDifferencePoint.x,
14                                    -m_fMaxSpeed,
15                                    m_fMaxSpeed );
16
17    m_kDifferencePoint.y = clamp( m_kDifferencePoint.y,
18                                    -m_fMaxSpeed,
19                                    m_fMaxSpeed );
20
21    m_kDifferencePoint.x *= m_fSlowingDownValue;

```

```

22
23     m_kDifferencePoint.y *= m_fSlowingDownValue;
24
25     moveItemsWithDirection( m_kDifferencePoint );
26 }
```

5.2.2 Ausrichtung des Menüs

Die Ausrichtung des Menüs erfolgt statisch, was heißt, dass sie sich nicht bei einer Rotationsveränderung des Gerätes ändert. Eine Implementierung der Autorotation wäre mit OpenGL möglich gewesen, ist in dieser Klasse jedoch nicht erwünscht.

5.2.3 Menügrenzen

Wenn das Ende des Scrollmenüs erreicht ist, muss geeignet reagiert werden. Es soll gestoppt werden, damit es nicht ins Unendliche scrollt. Das Stoppen, von zum Beispiel Listen, erfolgt bei Android und iPhone grundlegend verschieden.

Ein iPhone lässt das Menu bouncen. Das heißtt, es ist möglich über den Bildschirmrand hinweg zu scrollen, wird daraufhin aber wieder an den Endpunkt zurück gebounced. Bei Android stoppt das Menu und es ist kein Bounce- Effekt zu sehen. Da Spiele für Android und iPhone entwickelt werden, ist es wichtig, dass beide Effekte implementiert sind, damit der User auf Apple und Android Geräten jeweils den für sie richtigen Effekt sehen.

5.2.4 Unterklasse von Menu

Die Klasse ist von Menu abgeleitet. Menu ist eine Klasse der Nuroengine-x, welche eine Hilfsklasse für das Erstellen von GUI Elementen darstellt. Sie implementiert, die in Cocos2D-x standardisierten, GUI Elemente und erweitert sie um weitere. Daher ist diese Klasse auch von CCMenu abgeleitet (Quellcode 5.13).

So können dem Scrollmenü auf einfache Art neue Elemente hinzugefügt werden.

Quellcode 5.13: Basisklasse

```

1 class ScrollMenu : public Menu
2 {
3 }
```

5.2.5 Nutzen der ScrollMenu Klasse

ScrollMenu hinzufügen

In Quellcode 5.14 wird ein Scroll Menü hinzugefügt.

Quellcode 5.14: ScrollMenu hinzufügen

```

1 ScrollMenu* pkScrollMenu = new ScrollMenu( this ,
2                                         CCPoint( 0, 0),
3                                         CCSIZE( 160, 480),
4                                         CCPoint());
5
6 pkScrollMenu->init();

```

Parameter

Für die meisten Parameter sind Default-Werte angegeben (Quellcode 5.15), damit das ScrollMenü nur noch positioniert werden muss. Dies erspart bei der Nutzung des ScrollMenüs Zeit und Aufwand die richtigen Werte, für zum Beispiel die Scrollgeschwindigkeit, herauszufinden.

- Position
- Größe
- Scrollbarer Bereich
- Abstand zwischen Items
- Bounce an / aus
- Vertikal / Horizontal
- z-Ausrichtung
- Anzahl Zeilen / Spalten, je nach Ausrichtung
- Maximale Scrollgeschwindigkeit
- Verlangsamungswert beim Scrollen
- Bouncegeschwindigkeit

Quellcode 5.15: Default Werte

```

1 ScrollMenu( const cocos2d::CCPoint& rkPosition,
2             const cocos2d::CCSize& rkSize,
3             const cocos2d::CCSize& rkScrollingArea,
4             const cocos2d::CCPoint& rkPadding = cocos2d::CCPoint()
5
6             ,
7             const bool bBounce = false ,
8             const bool bVertically = false ,
9             const int nZOrder = 0,
10            const unsigned int unNumColRow = 1,
11            const float fMaxSpeed = 30.0 ,
12            const float fAccelerationValue = 0.9 ,
13            const float fBounceSpeed = 10.0 );

```

Elemente hinzufügen

Quellcode 5.16 zeigt das Hinzufügen von Elementen zu dem ScrollMenu. Dabei können alle Elemente, die Bestandteil von Menu sind, hinzugefügt werden.

Quellcode 5.16: Elemente hinzufügen

```

1 pkScrollMenu->addButton( "square_80_80_a.png",
2                               "square_80_80_a.png",
3                               menu_selector( TestMenu::onPlay2),
4                               Iphone3Point( fPosition, 0.0f),
5                               10);

```

5.2.6 Performance Analyse

Aufgrund der Ergebnisse des Performance Tests (Abb. 5.4), lässt sich sagen, dass die FPS (Frames per second) ausreichend sind. Ein Einbruch der FPS tritt erst ab einer Elementanzahl von etwa 800 ein. Diese Anzahl an Elementen wird jedoch nicht erreicht. Der Benutzer würde bei solch einer Anzahl zu lange für das Scrollen durch das Menü brauchen.

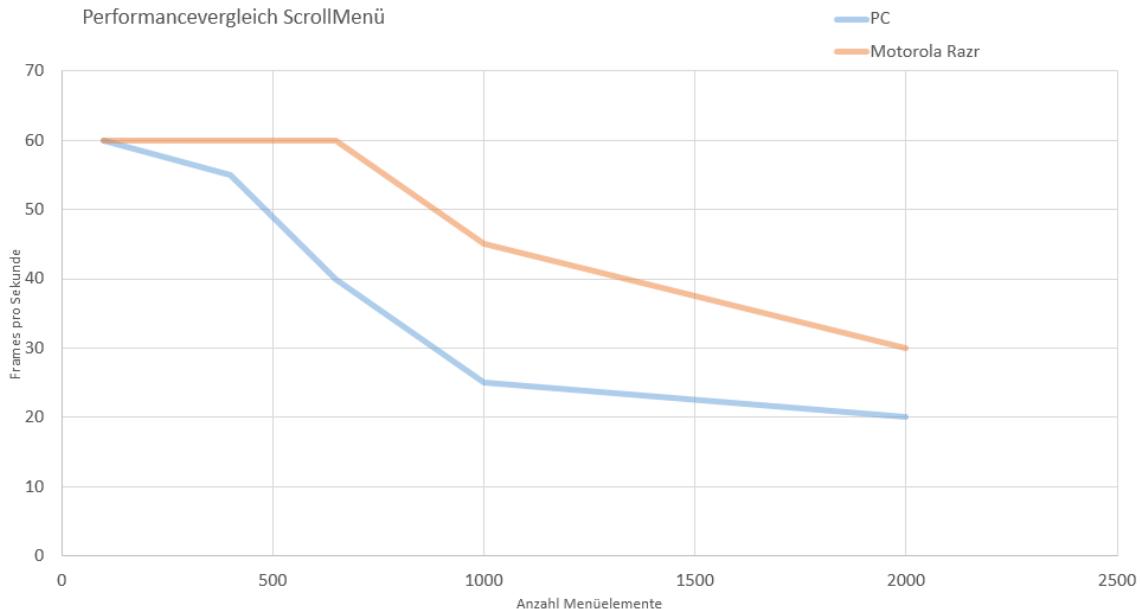


Abb. 5.4: Performance Analyse (FPS)

Es ist also möglich, bis zu 800 Elemente zu einem Scroll Menu hinzuzufügen, ohne dass die FPS einbrechen.

Aufgrund der Architektur von Cocos2D-x sind die größten Performanceeinbußen durch den dynamic_cast zustande gekommen. Dies lässt sich durch die Frameworkarchitektur jedoch nicht vermeiden.

Erklärung des Problems mit dem dynamic_cast:

Aufgrund falscher Polymorphie in Cocos2D-x ist ein dynamic_cast hier unumgänglich. Auch Cocos2D-x an sich sollte nicht geändert werden, da eventuell die Engine auf eine neue Cocos2D-x Version upgradet werden könnte. Somit wären alle Änderungen verworfen, oder eventuell nur mit großem Aufwand in die neue Version zu mergen. Mit richtiger Polymorphie wäre dieser Cast jedoch zu vermeiden.

Bezüglich der Performance des Scroll Menüs besteht hier jedoch trotzdem noch Optimierungspotential. So könnte mit Clipping die Anzahl der Elemente, die gezeichnet werden müssen, reduziert werden. Dies würde jedoch den Aufwand für den Nutzen überschreiten.

Optimierung:

Dynamic Casts sind sehr teuer und sollten deshalb vermieden werden. Wenn jedoch ein vorhandenes Framework erweitert und dies von vornherein so programmiert wurde, dass sich dynamic_casts, oder generell casts nicht vermeiden lassen, muss man diesen Flaschenhals, bezüglich der Performance, akzeptieren.

5.3 WebView

Die WebView Library implementiert die Schnittstelle zu gerätespezifischen Browsern. Des Weiteren wird auf der Android Seite, der Browser mit einer Anbindung an diese Schnittstelle realisiert. Die Webview ist elementarer Bestandteil eines Browergames auf Cocos2D-x Basis, da Cocos2D-x keinen Browser anbietet.

Folgende Funktionen werden unterstützt:

- Webseite initialisieren
- httaclues Authentifizierung setzen
- Callbacks registrieren
- Einen registrierten Callback aufrufen
- C++ Funktion ruft JavaScript auf
- JavaScript Funktion ruft C++ auf
- JavaScript ruft C++ auf und gibt einen Wert an JavaScript zurück

5.3.1 Struktur des Systems

Um den Aufbau und die Kommunikationswege besser zu verstehen, wird näher auf die Struktur des Systems eingegangen.

Die Kommunikation muss in mehreren Richtungen erfolgen:

- Von Cocos2D-x (C++) zu Android (Java) zu JavaScript/HTML
- Von JavaScript/HTML zu Android (Java) zu Cocos2D-x (C++)
- Von JavaScript/HTML zu Android (Java) zu Cocos2D-x(C++) zurück zu Android (Java) / zu JavaScript/HTML

Java wird bei einem anderen OS durch die gerätespezifische Sprache ersetzt.

Diese drei Kommunikationswege müssen existieren, damit JavaScript mit der Spielelogik kommunizieren kann und die Spielelogik mit der JavaScript Seite kommunizieren kann.

Dabei stellt die dritte Variante einen Spezialfall dar. Ein Anwendungsbeispiel wäre, wenn JavaScript C++ seitig einen Lokalisierungsstring benötigt. Dann wird die Anfrage von JavaScript zu C++ gesendet und C++ liefert den Lokalisierungsstring zurück an JavaScript. Alternative Anwendungsbeispiele sind denkbar.

Die Kommunikation zu Java von C/C++ aus und umgekehrt wurde in Abschnitt 4.2 JNI beschrieben.

Abbildung 5.5 zeigt die einzelnen Komponenten, die für die WebView Library relevant sind.

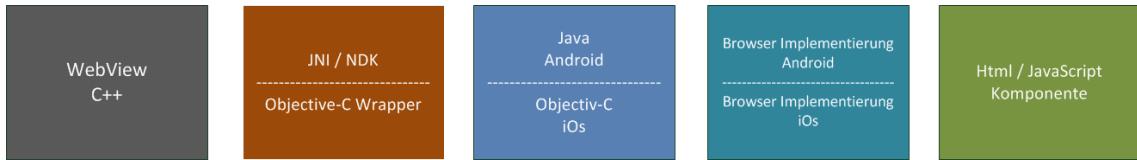


Abb. 5.5: WebView Komponenten

Die WebView besteht aus fünf Klassen und der jeweiligen WebView Implementierung auf den verschiedenen OS (Operating System) (Android, iOS).

- **WebView**

Erzeugt die Instanz und stellt pure virtuelle Methoden zur Verfügung, welche für die jeweilige Instanz implementiert werden müssen. Funktionen die instanzunabhängig sind, sind hier ebenfalls implementiert (nicht virtuell).

- **WebViewAndroid**

Implementiert die virtuellen Methoden der WebView für Android (Kommunikation mit Android).

- **WebViewWindows**

Da der Schwerpunkt der Nuroengine-x auf Entwicklung für Mobile Games liegt, ist diese Klasse zwar vorhanden, aber noch nicht implementiert.

- **WebViewIos**

Diese Klasse wurde, auf Basis der WebView, von einem externen Entwickler implementiert. Diese Klasse ist analog zu der WebViewAndroid Klasse, lediglich an iOS angepasst.

- **WebViewParameter**

Hilfklasse zur Erstellung stringbasierter Parameter.

Die jeweilige WebView Implementierung, für die verschiedenen OS, sind mit den jeweiligen SDKs der OS implementiert.

5.3.2 Instanzen der WebView Klasse

WebView ist eine abstrakte Klasse und als Singleton realisiert. Das heißt, dass entschieden wird welche Instanz des Browsers erzeugt wird. Unter Android ist dies die Android Instanz (AndroidWebView) und verhält sich analog bei den anderen Instanzen (Quellcode 5.17).

Die Unterscheidung ist mittels eines Macros von Cocos2D-x realisiert:

Quellcode 5.17: Instanzrückgabe via Macro

```

1 WebView& WebView::instance()
2 {
3 #if ( CC_TARGET_PLATFORM == CC_PLATFORM_WIN32)
4     static WebViewWindows s_kWebView;
5 #endif
  
```

```

6
7 #if ( CC_TARGET_PLATFORM == CC_PLATFORM_ANDROID)
8     static WebViewAndroid s_kWebView;
9 #endif
10
11 #if ( CC_TARGET_PLATFORM == CC_PLATFORM_IOS)
12     static WebViewIos s_kWebView;
13#endif
14
15     return s_kWebView;
16 }
```

Je nach Betriebssystem wird der jeweilige Header eingebunden und die jeweilige Instanz zurückgegeben.

5.3.3 Init

Zur Initialisierung des Browsers wird die Init Methode aufgerufen (Quellcode 5.18). Dabei kann sowohl eine Url (*http oder https*) zu einer Webseite / Webserver, auf dem das Spiel liegt, sowie nach Bedarf auch eine lokale, sich in der App befindlichen Datei, angegeben werden. Die Unterscheidung wird dabei auf OS Seite übernommen.

Quellcode 5.18: WebView initialisieren

```

1 WebView::instance().init( "JavascriptInterfaceTest.html");
2 WebView::instance().init( "http://www.urlzueinemgameserver.de");
3 WebView::instance().init( "https://www.sichereurlzueinemgameserver.de");
```

5.3.4 Callbacks

Aufgrund der Erweiterbarkeit der Webview, in Bezug auf Methoden für die Kommunikation zwischen C++ und der Webview, sind Callbacks die erste Wahl. So kann erst später, bei dem jeweiligen Spiel die Logik implementiert werden.

Unter einem Callback versteht man in diesem Falle eine Klasse, die es ermöglicht, eine sogenannte Rückruffunktion zu implementieren.

Abstrakter Callback

Ein abstrakter Callback ist eine, auf Templates basierte, abstrakte Klasse, die eine Abstraktion eines Callbacks darstellt. Es wird zwischen zwei verschiedenen Typen unterschieden:

- Abstrakter Callback TT
- Abstrakter Callback TV

Die Klassen für abstrakte Callbacks sind bereits Bestandteil der Nuroengine-x und mussten daher nicht eigens implementiert werden, sondern konnten genutzt werden.

Abstrakter Callback TT

- Rückgabetyp - Template
- Parametertyp - Template

Die Nutzung dieses Callbacks bietet sich an, wenn, an die Klasse die den Callback ausführt, ein TemplateType zurückgegeben werden soll.

Abstrakter Callback TV

- Rückgabetyp - Template
- Parametertyp - Void

Die Nutzung dieses Callbacks bietet sich in allen Fällen, an in der die Klasse keine Rückgabeinformation über den Callback benötigt.

5.3.5 Callback registrieren

Callbacks werden bei der Registrierung (Quellcode 5.19) in einer Map gespeichert. So lässt sich komfortabel, via String, auf ein Callback zugreifen. Methoden lassen sich mit der registerCallback Methode registrieren. Diese registrierten Methoden können zum Beispiel Java seitig aufgerufen werden. Die WebView leitet diesen Aufruf dann an die registrierte Methode weiter.

Hierbei hat die WebView Klasse keine Information darüber, was diese Methode, die mit dem Callback registriert wurde, ausführt. Sie dient lediglich als Vermittler zwischen der Spielelogik auf der C++ Seite, und der OS Seite und somit dem Browser (JavaScript / HTML).

Quellcode 5.19: registerCallback Methode

```

1 void WebView::registerCallback(
2     const String& rkCallbackName,
3     const AbstCallbackTT < const String,
4                                     const vector < WebViewParameter > >&
5                                         rkCallback
6 )
7     m_kCallbacksString[rkCallbackName] = rkCallback.clone();
8 }
```

Deklaration und Registrierung eines Callbacks:

Um einen Callback zu registrieren, muss dieser zuerst deklariert werden und kann dann an die registerCallback Methode (Quellcode 5.20) übergeben werden.

```
1 SecureCallbackTT < TestMenu,  
2                     const bool,  
3                     const std::vector < WebViewParameter > >  
4                     kCallback( this, &TestMenu::testCallback );  
5  
6 WebView::instance().registerCallback( "testCallback" , kCallback );
```

Exkurs Map in C++:

Eine Map in C++ ist ein assoziativer Container, was bedeutet, dass die Elemente nicht, wie bei Listen, via Position angesprochen werden, sondern über einen beliebigen, aber festen Schlüssel. Dabei ist der erste Parameter einer Map der Identifier und der zweite Parameter der Typ, der gespeichert werden soll. Beide können aufgrund der Template Funktionalität frei gewählt werden.

5.3.6 Javascript aufrufen

Um mit JavaScript zu kommunizieren, kann ein Request an Java gesendet werden. Auf Seiten Javas wird dieser Request dann an JavaScript weitergeleitet.

Es ist wichtig zu verstehen, dass bei einem JavaScript Aufruf mehrere Komponenten benötigt werden. Zum einen ist es die WebView, welche die Methode callJavaScript zur Verfügung stellt und auf der Java Seite eine Methode, die diesen Aufruf entgegen nimmt und an JavaScript weiterleitet.

C++ seitiges Aufrufen von JavaScript

Quellcode 5.21, Zeile 4 ruft auf JavaScript Seite die Funktion TestFunction auf und übergibt dieser die Parameter.

Quellcode 5.21: JavaScript aufrufen

```
1 std::vector < WebViewParameter > kParameter;
2 kParameter.push_back( true);
3 kParameter.push_back( String( "StringTest1"));
4 WebView::instance().callJavaScript( "TestFunction", kParameter);
```

C++ seitiges Aufrufen von JavaScript - Implementierung

Bevor der JavaScript Aufruf an Java weitergeleitet werden kann, müssen die Parameter erst zu einem String geparsed werden (Quellcode 5.22).

```

3           const std::vector < WebViewParameter >&
4           rkParameter)
5   {
6       String kParameter;
7       for( int nIndex = 0; nIndex < rkParameter.size(); nIndex++)
8   {
9       if ( nIndex != 0)
10      {
11          kParameter+=";";
12      }
13
14      String sTemp = rkParameter[nIndex];
15      kParameter += sTemp;
16  }
17 // Javascriptaufruf zu Java weiterleiten
18 m_pkJavaClass->callMethod( "request", rkMethodName, kParameter);
19 }
```

Java seitiges Aufrufen von JavaScript - Implementierung

Nachdem der Aufruf von C++ an Java weitergeleitet wurde, wird dieser Aufruf durch Java an JavaScript weitergeleitet (Quellcode 5.23).

Mit der Angabe ”javascript” in loadUrl ist es möglich, eine JavaScript Funktion aufzurufen.

Quellcode 5.23: Java aufruf JavaScript

```

1 public static void request( String rkMethodName, String rkParameters)
2 {
3     //Parameter parsen
4     //...
5     m_kWebView.loadUrl( "javascript:"+rkMethodName + parsedParameters);
6 }
```

5.3.7 WebView Implementierung Android seitig

Auf Seiten von Android wird eine so genannte WebView implementiert, welche einen Browser mit minimaler visueller Darstellung zur Verfügung stellt. Dies bedeutet, dass sich innerhalb dieser WebView **keine** weiteren GUI Elemente, wie zum Beispiel Adresseingabe, Reloadbutton und weitere, befinden.

Für diese Implementierung wird eine Klasse erstellt (*WebViewImplementation*). Als Parameter kann eine Activity übergeben werden, die später die Haupt-Activity der Android Anwendung (*sozusagen die main*) ist, an die die WebView gehängt wird (Quellcode 5.24 Zeile 3).

Damit der Browser auch den kompletten Bildschirm ausfüllt, ist es notwendig ein Layout zu erstellen und an die Activity zu hängen (Quellcode 5.24 Zeile 6).

Danach wird ein Objekt der Klasse WebView erzeugt, das wie oben beschrieben einen "Browser" implementiert (Quellcode 5.24 Zeile 8).

Damit sich die WebView auch innerhalb einer View befindet, ist es notwendig, die WebView zu der View hinzuzufügen (Quellcode 5.24 Zeile 15). Diese View befindet sich, wie bereits erläutert, in der Main-Activity.

Quellcode 5.24: WebView Implementierung Methode

```

1 private static WebView m_kWebView = null;
2
3 public WebViewImplementation( Activity kActivity)
4 {
5     m_kActivity = kActivity;
6     m_kView = new RelativeLayout( m_kActivity);
7     m_kContext = m_kActivity.getApplicationContext();
8     m_kWebView = new WebView( m_kContext);
9
10    RelativeLayout.LayoutParams kChildParams=
11        new RelativeLayout.LayoutParams(
12            LayoutParams.FILL_PARENT,
13            LayoutParams.FILL_PARENT);
14
15    m_kView.addView( m_kWebView, kChildParams);
16 }
```

5.3.8 WebView Einstellungen

Es besteht die Möglichkeit die WebView zu konfigurieren (Quellcode 5.25), wodurch die Scroll-Bars deaktiviert und Java aktiviert wird.

Quellcode 5.25: WebView Einstellungen

```

1 m_kWebView.setVerticalScrollbarOverlay( false);
2 m_kWebView.setVerticalScrollbarEnabled( false);
3 m_kWebView.setHorizontalScrollbarOverlay( false);
4 m_kWebView.setHorizontalScrollbarEnabled( false);
5
6 WebSettings kWebSettings = m_kWebViewSettings();
7 kWebSettings.setJavaScriptEnabled( true);
```

5.3.9 HTTP-Authentifizierung

Weiterhin muss der httpAuthRequest Handler überschrieben werden. Wird eine Seite geöffnet, die eine HTTP-Authentifizierung erwartet, wird diese Methode (Quellcode 5.26) aufgerufen um den Benutzernamen und das Passwort zu übergeben.

Quellcode 5.26: Http Authentifizierung

```

1 public WebViewImplementation( Activity kActivity)
2 {
3     ...
4     //http auth
5     m_kWebView.setWebViewClient( new WebViewClient()
6     {
7         @Override
8         public void onReceivedHttpAuthRequest( WebView view
9                 HttpAuthHandler handler,
10                 String host,
11                 String realm)
12         {
13             handler.proceed( m_kUsername, m_kPassword);
14         }
15     });
16     ...

```

5.3.10 Kommunikation von Java zu C++ sowie JavaScript Schnittstelle

Auf der Android Seite wird eine WebView implementiert, die den Browser (WebView Implementierung auf der Anroid Seite) darstellt. Diese implementiert ein JavaScript Interface, welches die Kommunikation zwischen JavaScript und Java ermöglicht (Quellcode 5.27 und 5.28).

Quellcode 5.27: Implementierung JavaScript Interface

```

1 /**
2 * JavaScriptInterface Interface for JavaScript to Call C++
3 */
4 final class JavaScriptInterface
5 {
6     private Handler m_khandler = new Handler();
7
8     public JavaScriptInterface()
9     {
10
11     }
12
13 /**
14 * Call Game Logic without Parameter
15 * @param sCppFunction Method in C++
16 */
17 public void callNativeMethod( final String sCppFunction)
18 {
19     m_khandler.post(new Runnable() {
20         public void run() {
21             callback( sCppFunction, " " );

```

```

22         }
23     });
24 }
25
26 /**
27 * Call Game Logic with Parameters
28 * @param sCppFunction Method in C++
29 * @param sParameters Parameters
30 */
31 public void callNativeMethod( final String sCppFunction,
32                             final String sParameters)
33 {
34     m_khandler.post(new Runnable() {
35         public void run() {
36             callback( sCppFunction, sParameters);
37         }
38     });
39 }
40
41 /**
42 * Call Game Logic with Parameters and return the returned value
43 * to javascript
44 * @param sCppFunction Method in C++
45 * @param sParameters Parameters
46 */
47 public String callNativeMethodReturn( String sCppFunction,
48                                     String sParameters)
49 {
50     return callbackReturn( sCppFunction, sParameters);
51 }
52 }
```

Dieses Interface übernimmt die Kommunikation mit C++ mit so genannten nativen Methoden. Die Grundfunktionalität solcher nativen Methoden wurde bereits in Kapitel 4.2 JNI erklärt.

Quellcode 5.28: Native Methoden bekannt machen

```

1 /**
2 * Native callback
3 * @param sCppFunction Method in C++
4 * @param sParameters Parameters
5 */
6 private native void callback( String sCppFunction, String sParameters);
7
8 /**
9 * Native callbackReturn
10 * @param sCppFunction Method in C++
11 * @param sParameters Parameters
```


5.3.12 Nutzen der WebView Klasse

Die Nutzung der WebView Klasse auf Seiten von C++ gestaltet sich einfach. Zuerst muss die WebView initialisiert werden und anschließend kann gegebenenfalls eine HTTP-Authentifizierung gesetzt werden und ein Aufruf an eine JavaScript Methode geschickt werden (Quellcode 5.31).

Quellcode 5.31: Nutzen der WebView

```
1 WebView::instance().init("http://server_url.de");
2 WebView::instance().httpAuth("username", "password");
3 std::vector<WebViewParameter> kParameter;
4 kParameter.push_back( 4410);
5 WebView::instance().callJavaScript( "Test1", kParameter);
```

Weiterhin kann die WebView auch JavaScript seitig genutzt werden.

5.3.13 Nutzung des JavaScript Interfaces

Auf Seiten von HTML / JavaScript kann auf die WebViewImplementierung auf der Java Seite zugegriffen werden. Dies geschieht mit dem JavaScriptInterface, welches auf der Java Seite implementiert ist.

Dieses Interface ist unter dem Namen *WebViewInterface* registriert. So kann mittels den Aufrufen in Quellcode 5.32 auf dieses Interface zugegriffen werden.

Dabei gibt es zwei Arten von Aufrufen, eine void und eine mit String als Rückgabeparameter.

Quellcode 5.32: Javascript Interface nutzen

```
1 <input type="button" onclick="WebViewInterface.callNativeMethod('
    testCallback','true')" value="Callback"/>
2
3 document.getElementById('localizedText').innerHTML = WebViewInterface.
    callNativeMethodReturn('localizeText', 'InfoText');
```

6 Isometric Tiled Map Engine

In diesem Kapitel werden Grundlagen von Tiled Maps, mit Schwerpunkt auf Isometric Tiled Maps, beschrieben. Zu den Grundlagen gehören zum Beispiel, die verschiedenen Arten von Tiled Maps, deren Erstellung und Analyse, sowie die Performanceoptimierung, bis hin zur Nutzung und Implementierung dieser.

6.1 Grundlagen

Tiled Maps haben ihren Ursprung bereits in den ersten Generationen von Computerspielen. Eines der bekanntesten Spiele, welches auf Tiled Maps basiert, ist Diablo oder auch andere Titel, wie Age of Empires, Dofus und weitere. Bei Dofus besteht sogar die Möglichkeit, sich das Tiled Map Raster anzeigen zu lassen und ist ein gewünschter Bestandteil des Spiels. Auch heute, im Zeitalter des mobilen Spielens, haben Tiled Maps ihre Daseinsberechtigung. So findet man Isometric Tiled Maps in ziemlich jedem Genre an Spielen, von Strategie, über Aufbau bis hin zu Roleplay Games und noch weiteren.

Unter einer Tiled Map versteht man die Darstellung einer virtuellen Spielwelt durch ein Raster. Das Raster besteht dabei aus einzelnen Tiles, welche zusammen eine Tiled Map ergeben. Dabei hat jede Tile in diesem Raster eindeutige Koordinaten.

Dabei muss zwischen verschiedenen Arten von Tiled Maps unterschieden werden. Es existieren die nicht isometrischen und die isometrischen, in welchen jeweils verschiedene Typen zu finden sind. In vielen Quellen wird die nicht isometrische Tiled Map als rechteckige Tiled Map beschrieben. Da dies jedoch nicht unbedingt der Fall ist, wird im Folgenden der Begriff nicht isometrisch genutzt.

6.1.1 Tiled Maps Arten

Nicht isometrisch

Nicht isometrische Tiled Maps eignen sich für reine 2D Spiele auf Basis von Grids. Ein einfaches Beispiel ist Vier gewinnt.

Isometrisch

Isometrische Tiled Maps sind hervorragend dazu geeignet, einen "pseudo" 3D Effekt zu erzeugen, auch 2.5D genannt. Das bedeutet, dass unter Nutzung der isometrischen Tiled Maps, Tiefe in das Spiel gebracht wird. Durch eine perspektivische Verzerrung lässt sich das Gehirn täuschen.

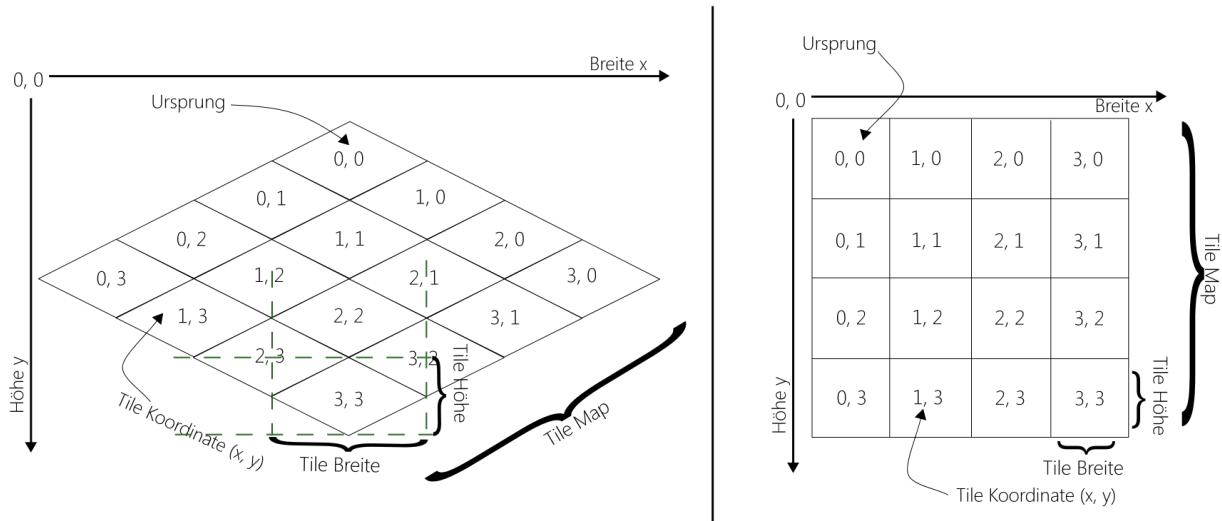


Abb. 6.1: Unterscheidung isometrisch (links) - Nicht isometrisch (rechts)

Bei isometrischen Tiled Maps muss zwischen verschiedenen Arten der Ausrichtung der Tiles zueinander unterschieden werden:

Diamond

Diamond Maps (Abb. 6.2) nutzen, wegen Ihrer Diamantenform, nicht den gesamten Bildschirm aus. So entstehen am Rand Flächen, die nicht nutzbar sind. Einige Spiele, die Diamond Maps nutzen sind: *Age of Empires* und sehr viele mobilen Spiele.

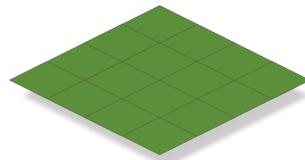


Abb. 6.2: Isometrisch Diamond

Staggered

Staggered Maps können, aufgrund Ihrer Form, den kompletten Bildschirm ausnutzen. Daher macht es Sinn, Staggered Tiled Maps zu verwenden, wenn der Platz ausgenutzt werden soll.

Unter Staggered Maps versteht man das versetzte Ausrichten der Tiles. Auf Abbildung 6.3 ist zu erkennen, dass jeweils zwei Zeilen zueinander einen Versatz von einer Tile haben. Einige Spiele, die Staggered Maps nutzen, sind: *Diablo* und *Fallout*.

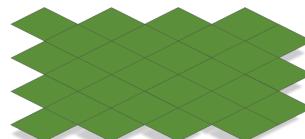


Abb. 6.3: Isometrisch Staggered

Slide

Slide Maps (Abb. 6.4) sind eine spezielle Art von Maps, bei der jede Reihe um eine Spalte versetzt dargestellt wird.

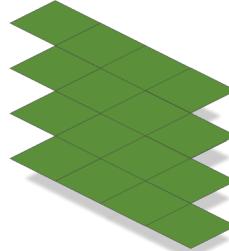


Abb. 6.4: Isometrisch Slided

6.1.2 Unterscheidung zwischen Isometrie und Perspektive

Isometrisch ist nicht gleichbedeutend mit perspektivisch. Isometrie bedeutet, die Gleicheit aller Längen, gleicher Objekte, unabhängig von der Distanz der Objekte in der View.

Bei perspektivischer Darstellung erscheinen die weiter hinten liegenden Objekte kleiner. Die perspektivische Darstellung spiegelt also die Darstellung, wie sie in der realen Welt existiert, wieder (Abb. 6.5).

Bei Spielen, basierend auf Tiled Maps, würde eine perspektivische Ansicht der Übersicht schaden und zudem ein verzerrtes Bild der Welt wiedergeben. Daher wird bei isometrischen Tiled Map Spielen die isometrische Ansicht (*oder eine verwandte Art der orthographischen Ansicht*) genutzt.

Es muss jedoch zwischen der "reinen" isometrischen Ansicht und der isometrischen Ansicht in Spielen unterschieden werden.

Angenommen es wird als Ausgangsobjekt ein Würfel genutzt, so sind bei der **reinen isometrischen Ansicht**, alle Seiten gleich lang. Weiterhin sind die Winkel zwischen den Projektionen der x, y und z-Achse alle 120°.

Bei der **isometrischen Ansicht** in Spielen, trifft die Definition der "reinen isometrischen Ansicht" nicht komplett zu. Hier sind lediglich zwei Winkel gleich. Dies basiert auf dem 2:1 Pixelmuster, welches in Abschnitt 6.1.4 noch erläutert wird.

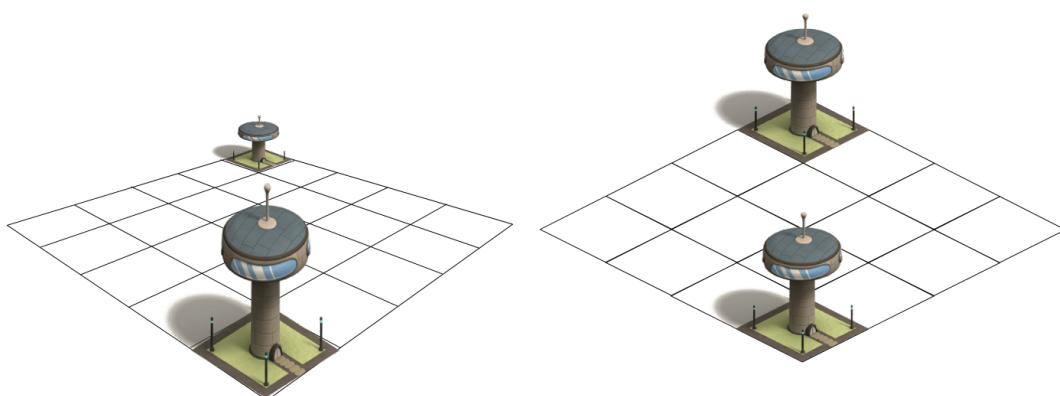


Abb. 6.5: Perspektivische Darstellung (links) gegen isometrische Darstellung (rechts)

6.1.3 Erzeugung regelmäßiger Pixelmuster

Um Artefakte zwischen einzelnen Tiles zu verhindern, muss ein regelmäßiges Pixelmuster erzeugt werden. Linien werden durch die Rasterisierung in diskreten Stufen dargestellt. Durch dies können bei bestimmten Winkeln unregelmäßige Pixelmuster (*Artefakte*) auftreten.

Ein Winkel von $26,565^\circ$ ($\arctan(0.5)$) erzeugt ein regelmäßiges Pixelmuster. Dagegen erzeugt ein Winkel von 30° oder 35° ein unregelmäßiges Pixelmuster, siehe Abbildung 6.6.

Regelmäßige Pixelmuster lassen sich so ohne Artefakte zusammensetzen. Bei unregelmäßigen Pixelmustern kann es zu Überschneidungen von Pixeln, oder auch zu Leerräumen zwischen einzelnen diskreten Stufen kommen.

Die "reine isometrische Ansicht" würde kein regelmäßiges Pixelmuster erzeugen. Bei dieser wird ein Winkel von $35,264^\circ$ genutzt, was einem ($\arctan(\sin(45^\circ))$) entspricht.

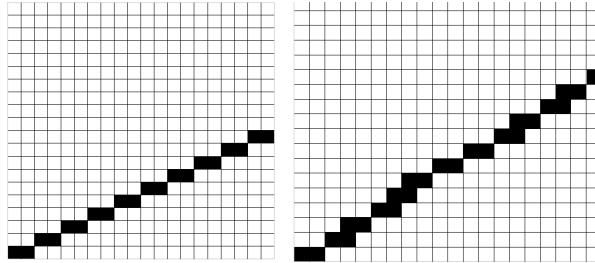


Abb. 6.6: Regelmäßiges Pixelmuster (links) - unregelmäßiges Pixelmuster (rechts)

6.1.4 Faktor 2:1

Der Begriff 2:1 in isometrischen Tiled Maps definiert die Darstellung einer Tile, durch die Anordnung der Pixel. In Abschnitt 6.1.3 wurde die Erzeugung regelmäßiger Pixelmuster erklärt, auf welcher der Faktor 2:1 basiert. In Prosa bedeutet es zwei Pixel horizontal und ein Pixel vertikal.

Mit dem Faktor 2:1 wird also ein Winkel von 26.57° beschrieben.

6.1.5 Isometrische Tile (Grundfläche) erstellen

Auf Basis der Grundlagen, bezüglich Pixelmuster 6.1.3 und dem 2:1 Faktor 6.1.4, lassen sich isometrische Grundflächen erstellen.

Auf Abbildung 6.7 sind die Einzelschritte der Erstellung aufgeführt:

- Quadratische Grundfläche erstellen
- Grundfläche um 45° rotieren
- Höhe um die Hälfte verringern

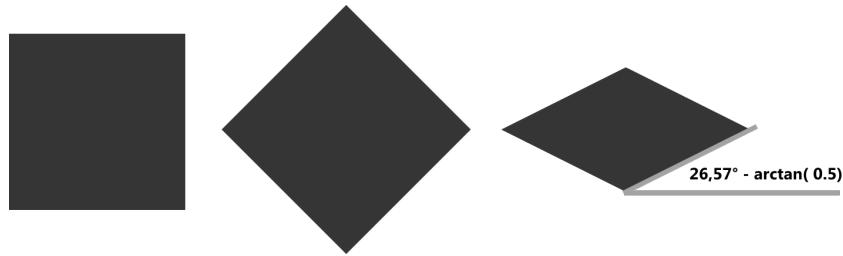


Abb. 6.7: Erstellungsschritte der isometrischen Grundfläche

Nach diesen Schritten entsteht ein Winkel von 26.57° (Abb. 6.8).

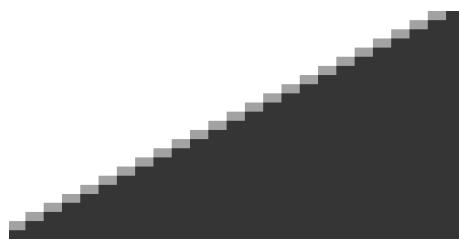


Abb. 6.8: Grundfläche vergößert

6.1.6 Isometrische Grafiken erstellen

Da ein Spiel nicht nur aus einer Engine besteht, sondern auch aus Grafiken, müssen diese, speziell in isometrischem Stil, erstellt werden. Es ist also wichtig, vorher genau zu definieren, in welchem Winkel und welcher Ansicht die Grafiken erstellt werden müssen. Diese Festlegung muss durchweg bei allen Grafiken eingehalten werden, da sonst die Ansicht darunter leidet.

Je nach Anwendungsgebiet wird eine isometrische Grafik verschieden erstellt. So wird zum Beispiel bei einem Aufbauspiel ein Gebäude eventuell aus mehreren Sichten benötigt. In diesem Fall bietet es sich an, das Gebäude als 3D Objekt zu erstellen und daraufhin in den richtigen Winkeln zu rendern.

Hier bietet es sich an, für Blender oder andere 3D-Tools, zum Beispiel ein RenderScript zu erstellen. Dieses Script rendert automatisiert die verschiedenen Gebäude, mit immer derselben Beleuchtung und aus dem gleichen Winkel, in den benötigten Rotationsrichtungen.

Wird jedoch eine Grafik nur aus einer Rotationsrichtung benötigt, oder der Aufwand mehrere Rotationsrichtungen zu erstellen ist gering, dann wäre das Script zu viel Arbeit. In diesem Fall wird die Grafik auf herkömmliche Art und Weise in Grafikprogrammen, wie zum Beispiel Adobe Photoshop oder Adobe Illustrator, erstellt.

6.1.7 Blender Einstellungen zur Erstellung isometrischer Renderings

In diesem Abschnitt werden exemplarisch die Einstellungen erläutert, die für das Rendern von isometrischen Objekten relevant sind. Dies stellt keine Referenzlösung dar, sondern nur eine Möglichkeit.

Orthographische Kamera

Wegen der Gründe die in Abschnitt 6.1.2 erläutert wurden, wird hier eine orthographische Ansicht verwendet.

Dazu wird die Kamera im Outliner ausgewählt und unter Object Data, die Lens Eigenschaften, auf Orthographic geändert.

Somit sind alle Seiten im Würfel gleichlang, egal wie nah oder weit die Kamera von dem Objekt entfernt ist.

Isometrischer Winkel

Um eine isometrische Ansicht auf das Objekt zu bekommen, muss der Kamerawinkel eingestellt werden. Daher wird in den Transform Einstellungen der Kamera, die Rotation um die x-Achse auf 60° und um die z-Achse auf 45° gestellt.

Das Objekt befindet sich dabei an $(0, 0, 0)$, lediglich die Kamera wird in Bezug auf das Objekt verändert.

Kameragröße

Da Objekte verschiedene Höhen besitzen können, wie zum Beispiel ein normales Haus im Vergleich zu einem Hochhaus, muss in Abhängigkeit des Objektes die Kamera richtig eingestellt werden.

Transparenz

Um das gerenderte Asset auch in einem Spiel nutzen zu können, ist es wichtig, dass leere Flächen nicht weiß oder schwarz gerendert werden, sondern transparent sind, da das Asset für das Spiel freigestellt wird.

Beleuchtung

Die Beleuchtungseinstellungen können nicht allgemein angegeben werden. Dies muss projektabhängig entschieden werden. Es bietet sich an, verschiedene Renderings mit verschiedenen Lichtmodi zu erstellen und im Team zu besprechen, welches Rendering zu dem besten Ergebnis führt.

Andere 3D-Tools

Diese Einstellungen lassen sich weitestgehend auch auf andere 3D Tools übernehmen. Zusammengefasst gesagt, sind hier die wichtigen Faktoren, die Rotation der Kamera zu dem Objekt und die Clippingeinstellungen der Kameraansicht.

6.2 Analyse

6.2.1 Ausschlussanalyse Cocos2D-x Tiled Map

Aufgrund der großen Einschränkungen der Tiled Maps in Cocos2D-x, wird die Isometric Tiled Map Engine zwar mit und für Cocos2D-x entwickelt, jedoch werden hierbei nicht die Module, die Cocos2D-x dafür anbietet, verwendet.

Probleme der Tiled Maps in Cocos2D-x:

- Tiled Map Erstellung nur mit TMX (Tile Map XML) Dateien
- Sehr enge Kopplung
- Mangelnde Möglichkeiten Objekte dynamisch zur Laufzeit zu ändern
- Performanceoptimierung im Nachhinein schwer zu realisieren
- Mangelnde Erweiterbarkeit
- Schlechte Wartbarkeit
- Eventuelle spätere Probleme bei Updates auf eine neue Cocos2D-x Version

6.2.2 Anforderungsanalyse

Im Folgenden sind alle Anforderungen an die Isometric Tiled Map Engine gelistet, die während der Analysephase ermittelt wurden. Dabei handelt es sich sowohl um **Pflichtanforderungen**, als auch um **optionale Anforderungen**. Diese beschreiben nur die Gesamtheit der Isometric Tiled Map Engine, wie sie in der Anforderungsphase ermittelt wurde.

Isometrische Ansicht

Die Grundfunktionalität dieser Engine ist die isometrische Ansicht. Diese wird durch einen speziellen Blickwinkel auf Objekte erzeugt.

Verschiedene Basiswinkel

Der Basiswinkel, in welchem die Tiled Map und damit auch ihre Objekte dargestellt werden, soll direkt oder indirekt beeinflussbar sein. Dies bedeutet, dass zum Beispiel, abhängig von der Tilegröße, der Winkel definiert wird.

Bewegungsrichtung

Im Falle, dass die Map größer als der sichtbare Bereich des Displays ist, muss die Möglichkeit bestehen die Map zu bewegen. Somit müssen sich auch alle dazugehörigen Objekte beim Bewegen der Map mitverschieben.

- Definierte Bewegungsrichtungen sind Norden, Süden, Osten und Westen
- Freie Bewegung anhand eines Inversen-Movement-Vector (IMV)

So kann mit der Distanz zwischen zwei Vektoren der Offset zur Bewegung der Map berechnet werden.

Zoom

Um einen besseren Bedienkomfort zu gewährleisten, ist es von Vorteil, eine Zoom Funktion zu integrieren. So kann bei einem weiten Zoom Faktor schnell über die Map gescrollt werden, sowie bei einem nahen Zoom Faktor die Details des Spiels besser zur Geltung kommen.

Dabei kann auf zwei Arten gezoomt werden:

- Multitouch Zoom
- DoubleTab Zoom

Objekte dynamisch zur Laufzeit ändern

Wie bei jedem Aufbauspiel, muss es möglich sein, auf einem Objekt, während seiner Laufzeit, folgende Funktionen auszuführen:

- Hinzufügen
- Rotieren
- Bewegen
- Entfernen

Z-Order Management

Objekte sollen automatisch den richtigen z-Value zugeordnet bekommen (Abb. 6.9). Dies ist wichtig, da sonst Objekte, welche visuell hinter einem anderen Objekt liegen, falsch angezeigt werden.

Der z-Value soll auch dann automatisch ermittelt werden, wenn ein Objekt über die Map bewegt oder auch darauf platziert wird.



Abb. 6.9: Darstellung der z-Order

Reverse Touch Tunneling

Nachdem ein Touch auf eine Tile erkannt wird, wird das zugehörige Objekt zu der Tile aufgerufen. Dies vereinfacht die Auswahl, wenn sich zum Beispiel ein großes Objekt vor einem kleineren befindet. So ist es dann immernoch einfach, die dahinterliegenden Objekte zu erreichen (Abb. 6.10).

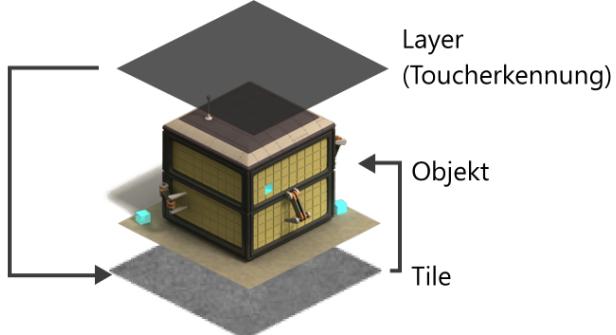


Abb. 6.10: Darstellung des Reverse Touch Tunneling

Performance-Optimierung

Die Engine soll so ausgelegt sein, dass sie mit minimalen Systemanforderungen zurecht kommt und auch bei großen Maps keine Performanceeinbrüche zu spüren sind.

Mögliche Verfahren in Bezug auf die Performance sind:

- Caching
- Object Pooling
- Optimierte Traversieren der Tile und der TileObjekte
- Proxy Maps
- Clipping
- Visibilitätsoptimierung
- Iterationsoptimierung

Callback Schnittstelle

Die Callback Schnittstelle soll Zugriff auf die wichtigsten Attribute der Tile und TileObjekte bieten. So ist es denkbar, die aktuelle Tile, auf welcher der Touch erkannt wird, an die Klasse, in der die TileMap genutzt wird, zurück zu geben (Callback). Somit steht in dieser Klasse die Funktionalität der Tile und der TileObjekte zur Verfügung.

Different Tile Map Support

Als Grundfunktionalität wird eine Diamond Map implementiert. Es soll jedoch im Nachhinein möglich sein, weitere Tiled Map Typen hinzuzufügen.

- Diamond
- Slide
- Staggered

Erweiterbare Map

Die Tiled Map soll sich zur Laufzeit vergößern lassen. Der blaue Bereich stellt eine beispielhafte Erweiterung der Map dar (Abb. 6.11).

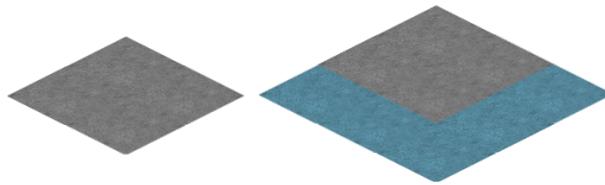


Abb. 6.11: Map vergößern

Visuelle Debugmöglichkeiten

Visuelle Debugmöglichkeiten sollen optional aktivierbar sein, welche in diesem Fall durch eine Präprozessordefinition realisiert werden. Dabei sollen die grundlegenden Informationen visuell dargestellt werden.

Wireframe

Die Tiles der Tiled Map sollen als Wireframe dargestellt werden, da dies die Platzierung der Objekte für den Spieler vereinfacht.

Optimierung der Darstellung beim Auftreten von Artefakten

Da Artefakte vermieden werden sollen, ist auf die optimale Darstellung der Sprites zu achten.

Verhalten bei verschiedenen Auflösungen

Aufgrund der Nuroengine-x, welche auf Cocos2D-x basiert, werden je nach Auflösung des Gerätes andere Sprites geladen. Dabei soll die Tiled Engine dies berücksichtigen.

Hintergrund für schwarze Bereiche außerhalb der Map

Da die Map visuell vergrößert werden soll, muss der schwarze Bereich außerhalb der Map angepasst werden. Auf diesen Bereich lassen sich keine Objekte platzieren. Hierbei kann zum Beispiel ein Hintergrund hinter die Tiled Map gelegt werden, welchen den schwarzen Bereich somit überdeckt.

Objekte mit verschiedenen Basisgrößen

Objekte sollen verschiedene Basisgrößen (Grundflächen) (Abb. 6.12) besitzen können. Das heißt, dass die größeren Objekte immer ein Vielfaches der kleinsten Tile sind ($1x1$, $2x2$, $3x3$, $4x4$, ...). Die Größe des Objekts ist nur durch die Größe des Objektsprites definiert.

Eine weitere Möglichkeit, die in Betracht gezogen werden soll sind Basisgrößen, die nicht quadratisch sind ($1x2$, $2x1$, $2x3$, ...).



Abb. 6.12: Basisgrößen (Links) $3x3$, (Mittig) $2x2$, (Rechts) $1x1$ - (Grafikquelle: Nurogames)

Objekte mit verschiedenen Ausrichtungen

Gebäude besitzen verschiedene Ausrichtungen (Abb. 6.13), die dynamisch zur Laufzeit geändert werden können. Dabei soll der Dateiname automatisch erstellt werden. Das Präfix hinter dem Objektnamen gibt die Ausrichtung an. Aufgrund der Tileform, gibt es maximal vier verschiedene Ausrichtungen des Objektes.

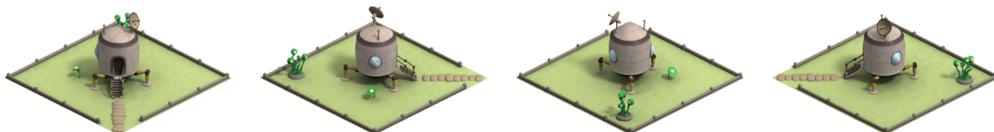


Abb. 6.13: Verschiedene Ausrichtungen - (Grafikquelle: Nurogames)

Zusammensetzbare Objekte

Ein zusammensetzbares Objekt ist zum Beispiel ein Fluss, welcher sich dynamisch verändert, wenn ein Objekt angefügt wird (Abb. 6.14). Dabei wird das jeweilig benötigte Sprite durch einen Algorithmus ermittelt. Die Nummerierung muss dabei einem festen Muster folgen.

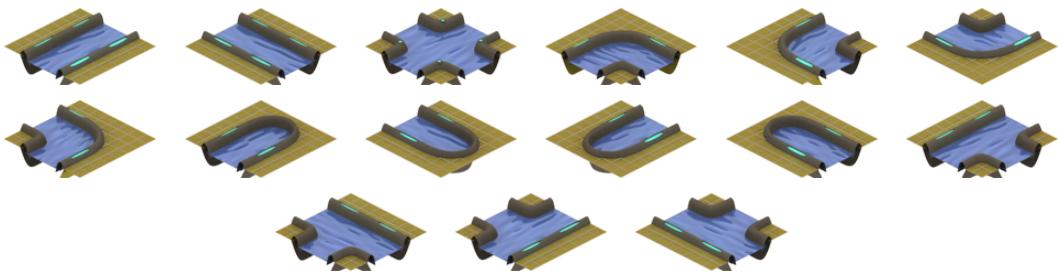


Abb. 6.14: Einzelsprites der zusammensetzbaren Objekte - (*Grafikquelle: Nurogames*)

Objekte mit Animation

Abbildung 6.15 zeigt zwei Frames eines animierten Sprites. Bei dieser Animation bewegt sich die Antenne, damit das Objekt lebhafter und nicht statisch erscheint.



Abb. 6.15: Einzelsprites einer simplen Animation mit zwei Frames - (*Grafikquelle: Nurogames*)

Objekte ohne Animation

Es soll auch möglich sein, Objekte ohne Animation hinzuzufügen. So genannte statische Objekte, welche jedoch genauso rotiert, bewegt und gelöscht werden können.

Diese sind im Vergleich zu den anderen Objekttypen aber nicht animiert und nicht zusammensetzbare.

Autonom bewegende Objekte / Pathfinding

Es ist darauf zu achten, dass die Engine so konzipiert wird, dass sich später ein Pathfinding Algorithmus darauf anwenden lässt. Dies kommt dann zum Zuge, wenn sich zum Beispiel Fahrzeuge, Charaktere oder Ähnliches auf der Tiled Map bewegen sollen.

Verschiedene Objektbau Modi

- Objekt erscheint auf einer bestimmten Tile bei Auswahl aus dem Baumenü
- Objekt erscheint erst bei TouchEnded auf einer Tile (vorher nicht sichtbar)

Object Opacity

Beim Bewegen der Objekte werden die Objekte leicht durchsichtig, wodurch auch Objekte, die hinter anderen Objekten liegen erkannt werden können. Weiterhin wird die Positionierung dadurch vereinfacht.

Map speichern und laden

Die Isometric Tiled Map soll sich, mit all ihren Komponenten und Objekten, speichern und wieder laden lassen. So kann an dem letzten Spielstand fortgesetzt werden.

Hier ist auf eine geeignete Schnittstelle zu der Map und den Objekten zu achten. So kann je nach Anwendungsfall entschieden werden, ob die Daten auf einem Server oder lokal auf dem Gerät gespeichert werden.

6.3 Entwurf / Konzeption

In diesem Kapitel wird grundlegend auf die einzelnen Klassen der Isometric Tiled Engine eingegangen. Hierbei wird die Grundfunktionalität und deren Aufbau erläutert.

6.3.1 Übersicht über die Tile Klasse

Eine Tile ist das kleinste Segment der Isometric Tiled Map Engine, kann ein Objekt beinhalten und ist für die Ausgabe des Wireframes verantwortlich. Gleichzeitig besteht hier das meiste Potential der Performanceoptimierung, da sehr viele Tiles innerhalb einer Map existieren können.

Funktionalität der Tile Klasse:

- Position

Position innerhalb der Tiled Map, an der sich die Tile befindet. Dieser Wert bleibt bezogen auf die TileMap fix.

- Koordinate

Jede Tile besitzt eine eindeutige und einmalige Koordinaten. Anhand dieser Koordinaten (x, y) kann direkt auf das Objekt zugegriffen werden.

- Tile Größe

Breite und Höhe der Tile, angegeben in Pixel.

- Tile Object

Zu jeder Tile kann ein Tile Object zugeordnet werden. So kann später bei einem Touch auf eine Tile das passende Objekt zurückgegeben werden

- Debug

Debuginformationen bezüglich der Tile werden ausgegeben.

- Visit

Visibilitätsoptimierung.

- Draw
Innerhalb der Draw Methode wird das Zeichnen der Wireframes implementiert.
- Z-Order
Bestandteil für das z-Order Management.

6.3.2 Übersicht über die TileObject Klasse

Ein TileObject umfasst alle Objekte. Die platziert werden können. Dabei wird zwischen verschiedenen Objekttypen unterschieden:

- Statische Objekte
- Animierte Objekte
- Zusammensetzbare Objekte

Hierbei haben alle Objekttypen die gleiche Grundfunktionalität:

- | | | |
|--------------|-------------|--------------------|
| • Platzieren | • Rotieren | • Textur ändern |
| • Bewegen | • Entfernen | • Deckkraft ändern |

Funktionalität der TileObject Klasse:

- rotate
Rotiert das Objekt je nach Objekttyp entweder durch Abspielen einer neuen Animation oder durch Setzen einer neuen Textur
- assetName
Gibt den Assetnamen (Dateinamen) des Assetobjektes zurück
- baseSize
Gibt die Basisgröße des Objektes zurück
- updateTexture
Erzeugt anhand einer Datei eine neue Textur und setzt die Textur des Objektes neu
- objectSize
Gibt die Größe des Objektsprites zurück
- opacity
Setzt die Deckkraft des Objektes
- buildFilePath
Erzeugt einen Dateipfad zu einem Objekt-Asset, mit einer bestimmten Rotationsrichtung, welcher wichtig für das Laden der neuen Textur, je nach Rotationsrichtung, ist

6.3.3 Übersicht über die TileObjectPath Klasse

Speziell für die zusammensetzbaren Objekte wird die Logik für diesen Objekttyp in eine eigene Klasse ausgelagert. Diese übernimmt dabei die Logik zur Bestimmung des richtigen zusammensetzbaren Objektes für bestimmte Tiles.

Diese Klasse ist für die Logik der verbindbaren Objekte, wie zum Beispiel bei Straßen oder Flüssen, zuständig. Wird ein Verbundobjekt gebaut, müssen die, um dieses Objekt liegenden, Verbundobjekte aktualisiert werden.

Funktionalität der TileObject Klasse:

- addReferenceTile
Die aktuelle Tile, auf der das Objekt hinzugefügt wird
- addNeighbourtile
Die benachbarten Tiles, die für die Bestimmung des Verbundobjekts benötigt werden
- updateTile
Ermittelt für die Tile das richtige Verbundobjekt
- updateTexture
Setzt, auf Basis des ermittelten Verbundobjekts, die neue Textur

6.3.4 Übersicht über die TileMap Klasse

Die TileMap Klasse stellt im Rahmen der Isometric Tiled Map Engine die größte Funktionalität zur Verfügung. Sie verwaltet die Tiles, die Objekte und die komplette Logik der Tile Engine in Bezug auf Tiles und Objekte und stellt die Schnittstelle dar.

Funktionalität der TileMap Klasse:

- generateMap
Erzeugt die Map anhand der angegebenen Breite und Höhe (in Anzahl Tiles)
- extend
Erweitert die Map um einen angegebenen Wert in Breite und Höhe
- tileAtTouch
Prüft anhand der Touch Position und der Tile Polygone welche Tile selektiert wird und gibt diese zurück
- saveMap
Speichert die Map in einer XML Datei, in welcher alle relevanten Informationen gespeichert werden
- addObjectStatic
Fügt ein statisches Objekt hinzu
- addObjectAnimated
Fügt ein animiertes Objekt hinzu

- addconnectableObject
Fügt ein Verbundobjekt hinzu
- addBackground
Fügt der Map einen Hintergrund hinzu
- startMoving / stopMoving
Setzen des Bewegungsmodus
- startRotating / stopRotating
Setzen des Rotationsmodus
- startDeleting / stopDeleting
Setzen des Löschmodus

6.3.5 Übersicht über die TileController Klasse

Der Tile Controller ist für die Interaktion mit der Tiled Map verantwortlich. Dabei wird dem TileController die Map als Parameter übergeben, die gesteuert werden soll. Für den Fall, dass mehrere Tiled Maps vorhanden sind, bekommt jede Tiled Map einen eigenen Controller.

Dabei erfüllt dieser folgende Funktionen:

- | | | |
|--|--|--|
| <ul style="list-style-type: none"> • Zoom • Scrollen | <ul style="list-style-type: none"> • Begrenzung • Bounce | <ul style="list-style-type: none"> • Zentriertes Ausrichten |
|--|--|--|

Funktionalität der TileController Klasse:

- updatePosition
Position der Map setzen
- initializeScroll
Initiale Touchpositionen speichern
- moveScroll
Wird bei einer erkannten Bewegung der Finger aufgerufen und berechnet den Versatz, um welchen die Map verschoben wird
- endScroll
Zurücksetzen der temporären Variablen
- beginZoom
Initiale Zoomwerte berechnen und speichern
- zoom
Zoomen der Map

6.3.6 Übersicht über die TileMath Klasse

Die TileMath Klasse stellt eine sogenannte Utility dar, welche dabei alle benötigten Berechnungen, die in der gesamten Tiled Engine benötigt werden, übernimmt.

Funktionalität der TileMath Klasse:

- pointInPolygon
Prüft, ob sich ein Punkt (x,y) innerhalb eines Polygons befindet
- convertTouchToGL
Hilfsmethode zum Umwandeln von Touches in Positionskoordinaten
- vectorsCenter
Berechnet den Mittelpunkt zwischen zwei Vektoren
- tilePolygon
Erzeugt ein Polygon als Vektor von 2D-Vektoren
- convertToScrollPoint
Berechnet, in Abhängigkeit von Touchpunkten und dem Skalierungsfaktor, den neuen Punkt, an dem sich die Map befindet
- scaleFactor
Berechnet den Skalierungsfaktor der Tiled Map
- tilePositionInDirection
Berechnet die Koordinaten der benachbarten Tiles

6.4 Implementierung

Die Funktionalität der Engine ist in sechs Klassen abstrahiert, womit eine optimale Abgrenzung erreicht wird. Dabei übernimmt jede Klasse andere Aufgaben.

Im Folgenden wird näher auf die Implementierung der einzelnen Funktionen der Isometric Tiled Map Engine eingegangen.

6.4.1 Tile

Koordinaten der Tile

Die Koordinaten der Tile (Zeile, Spalte) beschreiben die Position innerhalb der Tiled Map (Quellcode 6.1). Anhand dieser Koordinaten ist es möglich, die Koordinaten der benachbarten Tiles zu ermitteln. Aufgrund der Datenstruktur, in der die Tiles gespeichert werden, beschreiben die Koordinaten auch die Positionen innerhalb des Vektors. Somit kann auf einen Vektor, beziehungsweise dessen Tile, einfach zugegriffen werden.

Quellcode 6.1: Koordinaten der Tile

```
1 CCPoint Tile::getCoordinate()
```

```

2 {
3     return CCPoint( m_unCoordX, m_unCoordY );
4 }
```

Die beiden Variablen *m_unCoordX* und *m_unCoordY* werden im Konstruktor direkt bei der Erzeugung der Tile als Parameter übergeben.

Prüfen auf Objektbesitz einer Tile

Es ist für mehrere Bereiche relevant, ob eine Tile ein Objekt besitzt oder nicht (Quellcode 6.2). Hierauf wird im Nachfolgenden näher eingegangen.

Quellcode 6.2: Prüfen und Setzen

```

1 bool Tile::hasObject() const
2 {
3     return m_bObject;
4 }
5
6 void Tile::hasObject( const bool bHasObject )
7 {
8     m_bObject = bHasObject;
9 }
```

Setzen / Auslesen des TileObjekts zur aktuellen Tile

Diese Methoden (Quellcode 6.3) sind dafür zuständig, eine Referenz auf das Objekt der Tile zu setzen und zurückzugeben.

Quellcode 6.3: TileObject der Tile

```

1 TileObject* Tile::tileObject()
2 {
3     return m_pkTileObject;
4 }
5
6 void Tile::tileObject( TileObject* rkTileObject )
7 {
8     m_pkTileObject = rkTileObject;
9 }
```

Wireframe erzeugen

Das Zeichnen des Wireframes einer Tile erfolgt mit OpenGL. Dabei wird das TilePolygon (siehe Abschnitt 6.4.5 Erstellung des Tile Polygons) erstellt und dessen Vertices genutzt. Wireframes sind eine Alternative

zur visuellen Darstellung der Tile durch eine Bodentextur. Weiterhin wird kein Sprite für die Darstellung benötigt, womit Ressourcen gespart werden können.

Für das Zeichnen werden die Zeichenprimitive von Cocos2D-x genutzt. Hierbei handelt es sich um ccDrawPoly, durch welches die berechneten Vertices übergeben werden (erster Parameter).

Der zweite Parameter gibt die Anzahl der Vertices an und der letzte, ob das Polygon geschlossen werden soll oder nicht (Quellcode 6.4).

Quellcode 6.4: Wireframe

```

1 if ( !drawWireframeNeeded)
2 {
3     return;
4 }
5
6 float fInnerLine = 0;
7
8 if ( m_bSelected)
9 {
10     glColor4f( 0.5 , 1.0 , 0.5 , 1.0 );
11     fInnerLine = 2;
12     glLineWidth( 4 );
13 }
14 else
15 {
16     glColor4f( 1.0 , 1.0 , 1.0 , 1.0 );
17     glLineWidth( 1 );
18 }
19
20 std::vector < Vector2d> tilepoly = m_pkTileMath->tilePolygon(
21                                         CCPoint( 0,0 ) ,
22                                         getScale() ,
23                                         m_unWidth -
24                                         fInnerLine ,
25                                         m_unHeight -
26                                         fInnerLine );
27
28 CCPoint kWireframeVertices[] =
29 {
30     CCPointMake( tilepoly.at( 0 ).x() , tilepoly.at( 0 ).y() ) ,
31     CCPointMake( tilepoly.at( 1 ).x() , tilepoly.at( 1 ).y() ) ,
32     CCPointMake( tilepoly.at( 2 ).x() , tilepoly.at( 2 ).y() ) ,
33     CCPointMake( tilepoly.at( 3 ).x() , tilepoly.at( 3 ).y() )
34 };
35
36 ccDrawPoly( kWireframeVertices , 4 , true );
37 glDisable( GL_LINE_SMOOTH );

```

6.4.2 TileMap

Map erzeugen

Auf Basis der Höhe und der Breite der Map (in Anzahl Tiles) kann eine Map erstellt werden. Hierbei darf sich die Höhe und Breite unterscheiden. Es können also auch Maps erstellt werden, die nicht rein dem Diamant-Typ entsprechen (Abb. 6.16).

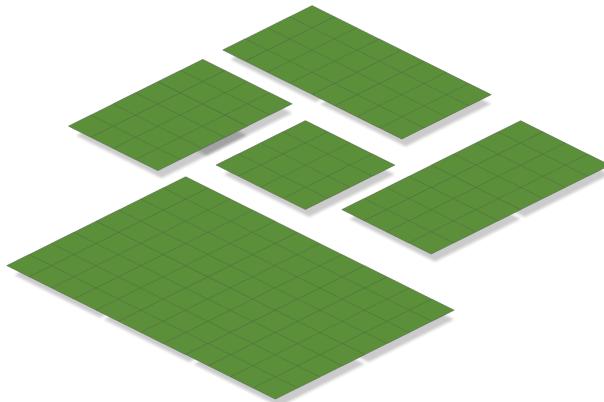


Abb. 6.16: Beispiele einiger Maps

Einzelschritte bei dem Hinzufügen von Tiles zu der Map (Quellcode 6.5):

- Über die Anzahl der Tiles in Höhe und Breite iterieren
- Position der jeweiligen Tile ermitteln
- Tile der Map hinzufügen (Kind)
- Tile dem Tile-Vector hinzufügen (wichtig für späteren Zugriff)

Quellcode 6.5: Tiles innerhalb der TileMap erzeugen

```

1 vector< Tile*> vTile;
2
3 for( unsigned int unWidthIndex = 0;
4     unWidthIndex < m_unMapWidth;
5     unWidthIndex++)
6 {
7     for( unsigned int unHeightIndex = 0;
8         unHeightIndex < m_unMapHeight;
9         unHeightIndex++)
10    {
11
12        Tile* kTile = new Tile( m_kTileSize.height,
13                               m_kTileSize.width,
14                               fTilePosX,
```

```

15                               fTilePosY,
16                               nZOrder,
17                               unWidthIndex,
18                               unHeightIndex,
19                               unTileCounter);
20
21     addChild( kTile, nZOrder);
22     kTile->debug();
23     fTilePosX += m_kTileSize.width / 2;
24     fTilePosY -= m_kTileSize.height / 2;
25     vTile.push_back( kTile);
26     nZOrder++;
27     unTileCounter++;
28 }
29
30     m_kTileVector.push_back(vTile);
31     vTile.clear();
32     nZOrder = -( m_unMapWidth * m_unMapHeight );
33     nZOrder += unWidthIndex+1;
34     fTilePosX = - ( m_kTileSize.width / 2 ) * ( unWidthIndex + 1 );
35     fTilePosY = - ( m_kTileSize.height / 2 ) * ( unWidthIndex + 1 );
36 }
```

Hintergrund

Durch den Hintergrund bekommt die Map mehr Tiefe und wirkt nicht zu statisch. Beim Hinzufügen eines Hintergrundes muss die z-Ausrichtung, in Abhängigkeit des geringsten z-Values, angepasst werden (Quellcode 6.6). Somit befindet sich der Hintergrund in jedem Fall hinter den Tiles.

Quellcode 6.6: Hintergrund hinzufügen

```

1 void TileMap::addBackground( CCSprite& rkBackgroundSprite)
2 {
3     m_kBackgroundSprite = &rkBackgroundSprite;
4     addChild( m_kBackgroundSprite, ( m_unMapWidth * m_unMapHeight ) * -2 );
5 }
```

Toucherkennung auf eine Tile

Für die Toucherkennung auf eine Tile (Quellcode 6.7) werden alle sichtbaren Tiles durchlaufen und geprüft, ob sich der Touchpunkt innerhalb des jeweiligen Tile-Polygons befindet.

Der Grund für das Iterieren über die sichtbaren Tiles wird in Kapitel 6.5.2 Iterationsoptimierung erläutert.

Quellcode 6.7: Tile an Touchposition ermitteln

```

1 for( unsigned int unColumn = 0; unColumn < m_unMapHeight; unColumn++)
```

```

2 {
3   for( unsigned int unRow = 0; unRow < m_unMapWidth; unRow++)
4   {
5     if ( m_kTileVector[unRow][unColumn]->getIsVisible() )
6     {
7       CCPoint tilePoint = m_kTileVector[unRow][unColumn]->
8         convertToWorldSpace(
9           CCPPointZero);
10
11    bool isTouchOnTile = m_pkTileMath->pointInPolygon(
12      kPoint,
13      m_pkTileMath->tilePolygon(
14        tilePoint,
15        getScale(),
16        m_kTileSize.width,
17        m_kTileSize.height));
18
19    if ( isTouchOnTile)
20    {
21      return m_kTileVector[unRow][unColumn];
22    }
23  }
24 }
25 }
26 return 0;

```

Hinzufügen von Objekten

Wird ein Objekt zum Beispiel aus dem Baumenü ausgewählt, soll dies an der Position, die der Benutzer berührt hat, hinzugefügt werden. Hierbei ist es wichtig, dass das hinzuzufügende Objekt vorerst temporär zur Verfügung gestellt wird.

Hier gibt es drei verschiedenen Methoden:

- addObjectNormal
- addObjectAnimated
- addObjectConnectable

Die verschiedenen Objekttypen werden in Abschnitt 6.4.3 TileObject näher erläutert.

Grundlegendes Touchhandling der TileMap

Vorab ist es wichtig zu verstehen, wie das Touchhandling zu handhaben ist. Innerhalb der Toucherkennung der Tiled Map erfolgt die Hauptlogik der Isometric Tiled Engine.

Innerhalb der Logik muss zwischen einem TouchBegan und einem TouchMoved unterschieden werden. Hier finden jeweils andere Überprüfungen statt.

TouchBegan:

Innerhalb des TouchBegan muss zuerst geprüft werden, ob eine Tile an der Position des Touches vorhanden ist. Danach kann ein Callback an die Schnittstelle gesendet werden, näheres in Abschnitt 6.6 Nutzung.

Im Folgenden muss unterschieden werden, ob die Tile ein Objekt besitzt, oder sich die Engine im Moment im Aufbaumodus (Modus zum Platzieren von Objekten auf der Map) befindet.

Befindet sich die Engine im Aufbaumodus, dann wird das Objekt, das zum Beispiel aus dem Bau menü gewählt wurde, platziert (Quellcode 6.8). In diesem Fall muss auch keine TouchMoved Methode aufgerufen werden, daher kann false zurückgegeben werden.

Quellcode 6.8: Aufbaumodus

```

1  else if ( m_bBuildMode )
2  {
3      CCPoint kObjectPosition = m_pkCurrentTile->getPosition() ;
4
5      if ( pkCurrentObject->baseSize() % 2 == 0 )
6      {
7          kObjectPosition.x = m_pkCurrentTile->getPosition().x;
8          kObjectPosition.y = m_pkCurrentTile->getPosition().y + m_kTileSize.
9              height/2;
10
11     pkCurrentObject->setPosition( kObjectPosition );
12     addChild( pkCurrentObject );
13     m_bBuildMode = false ;
14     placeObjectAtTiles() ;
15     return false ;
16 }
```

Befindet sich die Engine jedoch nicht im Aufbaumodus und die Tile besitzt ein Objekt, so ist es wahrscheinlich, dass eine Aktion auf diesem Objekt ausgeführt werden soll.

Diese könnten sein:

- Löschen
- Rotieren
- Bewegen

Löschen:

Wird ein Objekt gelöscht, muss das Objekt als Kind, innerhalb des MapNodes, entfernt werden, sowie die Referenz und die Instanz des Objektes gelöscht werden (Quellcode 6.9).

Quellcode 6.9: Kind Entfernen und Referenz und Instanz löschen

```
1 if ( m_bDeleteMode )
```

```

2 {
3     removeChild( pkCurrentObject, true );
4     removeObjectReferencAtTile();
5     delete pkCurrentObject;
6     return false;
7 }
```

Rotieren:

Wird ein Objekt rotiert, muss für das Objekt, das sich auf dieser Tile befindet, die Rotate Funktion aufgerufen werden (Quellcode 6.10).

Quellcode 6.10: Objekt rotieren

```

1 if ( m_bRotatingMode )
2 {
3     m_pkCurrentTile->tileObject()->rotate();
4     return false;
5 }
```

Bewegen:

Soll ein Objekt bewegt werden wird true zurückgegeben. Bei einer Bewegung des Fingers wird dann die TouchMoved Methode aufgerufen, welche wiederum das Bewegen der Objekte übernimmt (Quellcode 6.11).

Quellcode 6.11: Bewegungsmodus

```

1 if ( m_bMovingMode )
2 {
3     m_bObjectMode = true;
4     return true;
5 }
```

TouchMoved:

Innerhalb der TouchMoved Methode wird das Bewegen der Objekte auf der Map übernommen. Hier wird analog, zurr TouchBegan Methode, zuerst die Tile ermittelt, auf der sich im Moment der Touch befindet. Dieser dient wieder als Ausgangspunkt für die Berechnungen (Quellcode 6.12).

Es muss geprüft werden, ob sich an der Touch Position eine Tile befindet.

Quellcode 6.12: Tile an Position

```

1 if ( m_pkCurrentTile == 0 )
2 {
3     return;
4 }
```

Ist der Bewegungsmodus nicht aktiv, kann auch kein Objekt bewegt werden, daher return (Quellcode 6.13).

Quellcode 6.13: Auf Bewegungsmodus prüfen

```

1 if ( !m_bMovingMode )
2 {
3     return;
4 }
```

Besitzt die Tile bereits ein Objekt, so kann das momentane Objekt nicht auf dieser Tile positioniert werden. Näheres zu diesem Prüfungsverfahren in Kapitel Prüfung auf Besitz einer Objektreferenz (Quellcode 6.14).

Quellcode 6.14: Prüfung auf Objektbesitz

```

1 if ( tilesHasAlreadyObject() )
2 {
3     return;
4 }
```

Danach wird das Objekt auf der Map bewegt, siehe Bewegung von Objekten.

Bewegung von Objekten

Um Objekte an eine Position zu setzen müssen diese auf der Map bewegt werden können. Hierbei wird nicht die Position des Sprites verändert, sondern die Position des Nodes (TileObject), in dem das Objekt Sprite liegt.

Dabei sind verschiedene Informationen relevant:

- Position der Tile, auf welche der Touch erkannt wird (T1)
- Position der Tile, auf welcher der Touch bewegt wird (T1')
- Länge zwischen diesen zwei Positionen
- Position des Objektes

Auf Basis von T1 und T1' wird die Länge zwischen den zwei Positionen berechnet. Diese Länge hat je nach Richtung + oder - als Vorzeichen und beschreibt den Wert, um welchen das Objekt, ausgehend vom Mittelpunkt (M1) an Position (P1) bewegt wird (Abb. 6.17).

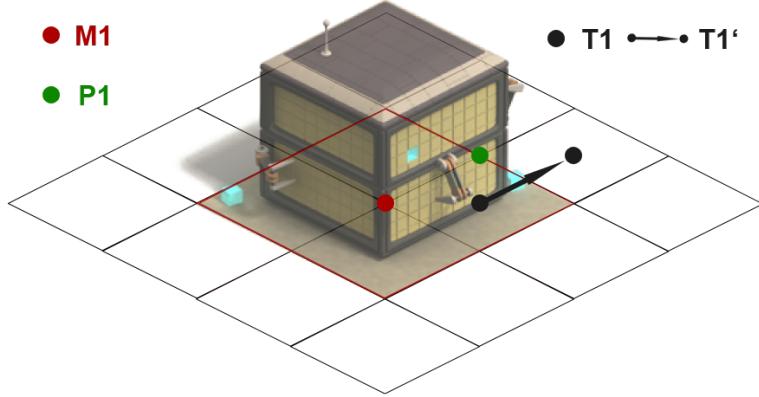


Abb. 6.17: Darstellung der Berechnung des Versatzwertes

Ohne diese Berechnung würde das Objekt bei der Bewegung springen. Dieser Effekt (*TileHopping*) wurde im Rahmen der Anforderungsanalyse auch in anderen Mobile Games auf Tiled Basis entdeckt.

TileHopping beschreibt einen Effekt, bei dem ein Objekt innerhalb einer Tiled Map auf eine falsche Tile springt (Abb. 6.18), ausgehend von einer falschen Berechnung der Touchpositionen. Objekte springen zwar von Tile zu Tile und werden nicht pixelweise bewegt, da sich das Objekt immer in den Tiles befinden muss, jedoch darf es nicht falsch springen.

Ein Beispiel, bei welchem der TileHopping Effekt auftritt, ist das Neupositionieren von Objekten ohne die Länge zu berechnen. Je nach Ankerpunkt kommt es dabei zu einem TileHopping Effekt oder eventuell auch zu einem falschen Versatz auf der Tile.

Folgende Grafik veranschaulicht das TileHopping. Hierbei sind die grauen Felder das Objekt vor dem Verschieben, die roten Felder das Objekt nach dem Verschieben und die blauen Kreise geben die Touchpunkte an.

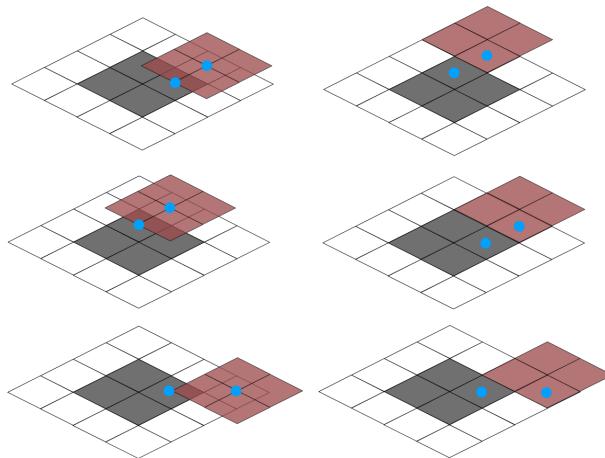


Abb. 6.18: TileHopping Ankerpunkt Mittig (links) Ankerpunkt Links unten (rechts)

Im Folgenden ist eine Definition festgelegt die, das Verfahren des korrekten Bewegens von Objekten auf der Tiled Map beschreibt:

Angenommen ein Objekt besitzt eine Basisgröße > 1 , somit ist das Objekt in n-Tiles (Objekt Tiles)

aufgeteilt, dann befindet sich die Tile, auf welcher der Touch begonnen hat, nach dem Bewegen des Fingers immernoch exakt unter dem Finger (Quellcode 6.15).

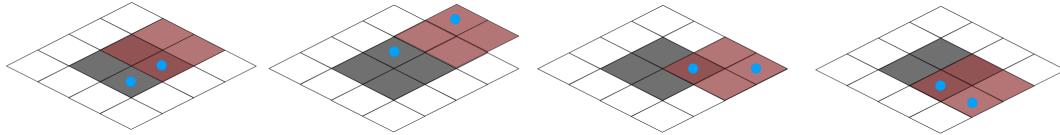


Abb. 6.19: Objekte bewegen nach Definition

Im Vergleich zu Abbildung 6.18 ist auf Abbildung 6.19 kein TileHopping mehr zu erkennen und die Definition wird erfüllt.

Quellcode 6.15: Verfahren des korrekten Bewegens von Objekten

```

1   CCPoint kTileTouchMoved = m_pkCurrentTile->getPosition();
2   CCPoint kTileTouchBegan = lastTilePoint;
3   CCPoint kDifferencePoint = CCPoint( kTileTouchBegan.x -
4                                         kTileTouchMoved.x,
5                                         kTileTouchBegan.y -
6                                         kTileTouchMoved.y );
7
8
9   pkCurrentObject->setPosition(
10     CCPoint( pkCurrentObject->getPosition().x - kDifferencePoint.x
11             ,
12             pkCurrentObject->getPosition().y - kDifferencePoint.y
13             ) );
14
15   placeObjectAtTiles();
16   lastTilePoint = kTileTouchMoved;

```

Folgende Schritte werden bei der Bewegung ausgeführt:

- Alle Referenzen auf Tiles, des sich gerade bewegenden Objektes, werden gelöscht
- Position wird wie beschrieben ermittelt
- Es wird geprüft, auf welchen Tiles sich das Objekt befindet
- Es wird geprüft, ob die Tiles, auf denen sich das Objekt befindet, bereits eine Referenz auf ein anderes Objekt besitzen
- Die Tiles auf denen das Objekt liegt, erhalten eine Referenz auf das Objekt

Dieser Prozess ist notwendig und stellt einen der komplexesten Bereiche der Tiled Engine-Logik dar.

Löschen alter Objektreferenzen

Alle Tiles, die das Objekt "berührt", bekommen eine Referenz auf das Objekt. Diese Referenz muss bei der Bewegung entfernt werden (Quellcode 6.16), da sich das Objekt nicht mehr auf dieser Tile befindet.

Die Objektreferenz wird bei der Bewegung, oder auch beim Entfernen des Objektes, gelöscht. Diese Aufgabe übernimmt die removeObjectReferenceAtTile Methode.

Dabei werden alle sichtbaren Tiles durchlaufen und geprüft, ob sich die jeweilige Tile Position innerhalb des TileObject-Polygons befindet.

Quellcode 6.16: Prüfen ob Tile innerhalb des Objekt Polygons

```

1 std::vector< Vector2d > vObjectPolygon = m_pkTileMath->tilePolygon(
2                                     kObjectPolygon,
3                                     getScale(),
4                                     m_kTileSize.width * pkCurrentObject->baseSize(),
5                                     m_kTileSize.height * pkCurrentObject->baseSize());
6
7 bool bObjectOnTile= m_pkTileMath->pointInPolygon( kTilePoint,
8                                     vObjectPolygon);

```

Ist dies der Fall, wird die Referenz gelöscht und der bool'sche Wert (*hasObject*) auf false gesetzt (Quellcode 6.17).

Quellcode 6.17: Referenz entfernen

```

1 if ( bObjectOnTile)
2 {
3     m_kTileVector[unRow][unColumn]->hasObject( false );
4     m_kTileVector[unRow][uncolumn]->tileObject( NULL );
5 }

```

Setzen der Objektreferenz

Die Tiles, auf der sich das Objekt nach der Bewegung befindet, müssen eine Referenz auf das Objekt besitzen. Diese muss sowohl bei der Platzierung, als auch bei der Bewegung gesetzt werden (Quellcode 6.18).

Quellcode 6.18: Setzen der Objektreferenz

```

1 if ( bObjectOnTile)
2 {
3     m_kVisibleTilesVector.at( unIndex)->hasObject( true );
4     m_kVisibleTilesVector.at( unIndex)->tileObject( pkCurrentObject );

```

Prüfung auf Besitz einer Objektreferenz

Existiert bereits auf einer Tile eine Referenz, auf die das Objekt bewegt werden soll, so muss der Bewegungsvorgang abgebrochen werden. Analog darf auf eine Tile, die bereits ein Objekt besitzt, kein weiteres Objekt platziert werden.

Das Prüfen läuft innerhalb der Methode, die für das Setzen der Objektreferenz zuständig ist ab (Quellcode 6.19).

Quellcode 6.19: Prüfung auf Besitzt einer Objektreferenz

```

1 if ( m_kVisibleTilesVector.at( unIndex )->hasObject() )
2 {
3     if ( m_kVisibleTilesVector.at( unIndex )->tileObject() != 
4             pkCurrentObject )
5     {
6         return;
7     }
8 }
```

z-Order Management

Das z-Order Management ist für die Berechnung des richtigen z-Wertes verantwortlich. Dabei ist der z-Order Wert abhängig von:

- Der aktuellen Tile, auf der sich das Objekt befindet
- Basisgröße des Objektes
- z-Order Muster der Tiled Map

Jede Tile bekommt bei ihrer Erstellung einen spezifischen (*nicht einmaligen*) z-Wert. Die Tile an Position (0, 0) hat den z-Wert $z = (\text{Anzahl Tiles Breite} * \text{Anzahl Tiles Höhe})$. Dies ergibt zum Beispiel bei einer 10 x 10 Map einen z-Wert von -100. Die nächste Zeile hat einen z-Wert von -99 und so weiter.

Grund für diese Art der z-Order, ist die Einfachheit des Algorithmus zur Bestimmung des richtigen z-Wertes für ein Objekt. Abbildung 6.20 zeigt die Problematik bei z-Mustern, die nicht dieser waagerechten, aufsteigenden Sortierung folgen und somit die hintereinander liegenden Objekte nicht richtig anzeigen.

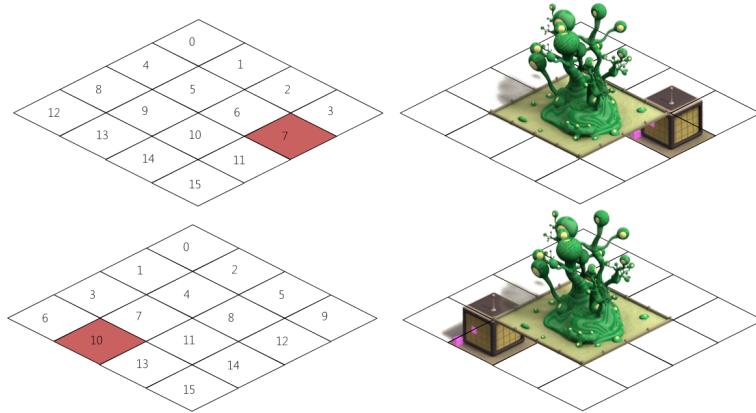


Abb. 6.20: Falsches z-Order Management

Da die Tile Objekte auch mehr als eine Tile in ihrer Größe beanspruchen können, muss in Abhängigkeit der Basisgröße ein Korrektur (Quellcode 6.20) des z-Wertes vorgenommen werden.

Diese Korrektur wird folgendermaßen berechnet:

$z\text{-Order} = (z\text{-Order der Tile, auf der der Touch erkannt wird} - \text{Basisgröße des Objektes})$

Quellcode 6.20: z-Order bei der Positionierung

```
1 reorderChild( pkCurrentObjet, m_kVisibleTilesVector.at( unIndex )->
    getZOrder - ( pkCurrentObject->baseSize() ));
```

Nach der Nutzung der waagerechten aufsteigenden Sortierung und Berechnung des Korrekturwertes sind alle z-Werte richtige, siehe Abbildung 6.21.

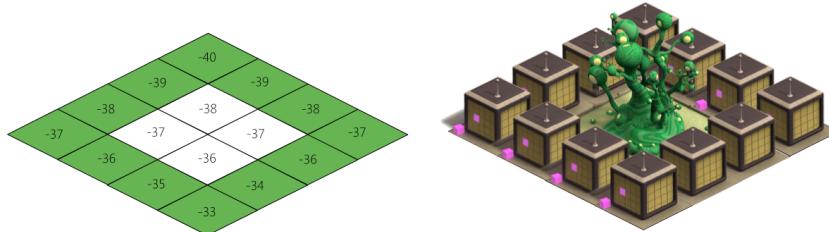


Abb. 6.21: Richtiges z-Order Management

6.4.3 TileObject

Aufbau eines Objekts

Ein Tile Objekt besitzt als Basisklasse ein CCNode. Die Gründe für die Benutzung von CCNode als Basisklasse wurden bereits in den Grundlagen Cocos2D-x (Kapitel 3.1 Node) erklärt.

Dieses CCNode besitzt hierbei einen Ankerpunkt von (0.5, 0.5). Dieser Wert ist für die Positionierung der Objekte auf der Map relevant. Innerhalb dieses CCNodes ist, je nach Objekttyp, ein Sprite oder eine Animation als Kind hinzugefügt (im Folgenden Objekt-Asset genannt). Dieses Objekt-Asset besitzt seinen Ankerpunkt bei (1.0, 0.0), also in der rechten unteren Ecke. Der Grund für diesen Ankerpunkt liegt in der Art und Weise der Objekt-Assets, welche einen Schatten auf der linken Seite besitzen. Es

ist also nicht möglich, Objekt-Assets anhand ihrer Grundfläche auf der linken Seite auszurichten, da der Schatten eventuell über die Grundfläche hinaus geht. Daher muss der Ankerpunkt auf die rechte Seite verschoben werden.

Da der Ankerpunkt des CCNode jedoch bei (0.5, 0.5) liegt und der Ankerpunkt der Objekt-Assets bei (1.0, 0.0) muss für die richtige Ausrichtung, in diesem Fall ein Versatz auf der x-Achse, berechnet werden.

Dieser ergibt sich aus der Breite der Tile und der Basisgröße des Objektes (Quellcode 6.21)

Quellcode 6.21: Versatzberechnung

```
1 CCPoint redefinedPosition= CCPoint( 0, rkTileSize.width * unBaseSize);
2 m_pkObjectSprite->setPosition( redefinedPosition);
```

Objekt Opacity

Wird ein Objekt auf der Map bewegt, wird die Deckkraft aller anderen Objekte, auf den in der Initialisierung angegebenen Wert, reduziert (Quellcode 6.22). So ist einfacher ersichtlich, welches Objekt bewegt wird und auch die Sichtbarkeit hinter sehr hohen Objekten ist ebenso gewährleistet. Dies vereinfacht für Benutzer die Bewegung der Objekte auf der TiledMap.

Als Eingabe erwartet diese Methode ein GLubyte. Das heißt, es können Werte von 0 bis 255 übergeben werden. Dabei bedeutet 0 keine Deckkraft und 255 maximale Deckkraft.

Quellcode 6.22: Opacity von Sprites setzen

```
1 void TileObject::opacity( const unsigned unOpacity)
2 {
3     if ( m_bAnimation)
4     {
5         m_pkAnimation->setOpacity( unOpacity);
6     }
7     else
8     {
9         m_pkObjectSprite->setOpacity( unOpacity);
10    }
11 }
```

Animierte Objekte

Animierte Objekte (Quellcode 6.23) besitzen Animationen, die gestartet werden könne. Dabei wird eine Animation, die Default Animation, immer automatisch gestartet. Durch die updateAnimation Methode kann jedoch auch eine andere Animation gestartet werden.

Hierbei gibt der erste Parameter den Link zu dem Animationsfile an und der zweite den Namen der Animation, die gestartet werden soll.

Die Klasse Animation ist Bestandteil der Nuroengine-x und wird hier genutzt.

Quellcode 6.23: Animierte Objekte

```

1 TileObject::TileObject( const baseLib::String& rkAnimation,
2                         const baseLib::String& rkAnimationName,
3                         const unsigned int unRotationDirections,
4                         unsigned int unBaseSize)
5 :
6     CCNode(),
7     m_kAnimation( rkAnimation ),
8     m_kAnimationName( rkAnimationName ),
9     m_nRotationDirection( unRotationDirections ),
10    m_nBaseSize( unBaseSize )
11 {
12     initialize();
13     m_pkAnimation = new Animation();
14
15     if ( !m_pkAnimation->initialize( rkAnimation ) )
16     {
17         msg::error( "TileObject::TileObject" ,
18                     "Failed to create animation from file '" + rkAnimation + "'!\n"
19                     );
20     }
21     m_pkAnimation->setAnchorPoint( CCPoint( 1.0 , 0.0 ) );
22     m_pkAnimation->play( rkAnimationName , true );
23     addChild( m_pkAnimation );
24 }
```

Zusammensetzbare Objekte

Zusammensetzbare Objekte (Quellcode 6.24) wie zum Beispiel Straßen oder Flüsse besitzen einen Algorithmus, der die benachbarten zusammensetzbaren Objekte richtig zueinander ausrichtet und aktualisiert. Dabei folgt der Algorithmus einer genau definierten Reihenfolge, was heißt dass die verschiedenen Objekte passend zum Algorithmus benannt werden müssen (Kapitel 6.4.4 TileObjectPath).

Quellcode 6.24: Zusammensetzbare Objekte

```

1 TileObject::TileObject( const String& rkConnectableObjectImage ,
2                         const unsigned int unBaseSize ,
3                         const CCSIZE rkTileSize )
4 :
5     CCNode(),
6     m_kTileObjectImage( rkConnectableObjectImage ) ,
7     m_bConnectable( true ) ,
8     m_bAnimation( false ) ,
9     m_nRotationDirection( 10 ) ,
10    m_nBaseSize( unBaseSize ) ,
11    m_kTileSize( rkTileSize )
```

```

12 {
13     initialize();
14
15     m_pkObjectSprite = CCSprite::spriteWithFile( buildFilePath());
16     m_pkObjectSprite->setAnchorPoint( CCPoint( 1.0, 0.0));
17     m_pkObjectSize = m_pkObjectSprite->getContentSize();
18     CCPoint redefinedPosition = CCPoint();
19     redefinedPosition.x += rkTileSize.width * unBaseSize;
20     m_pkObjectSprite->setPosition( redefinedPosition);
21     addChild( m_pkObjectSprite);
22 }
```

Statische Objekte

Unter statischen Objekten werden die Objekte verstanden, welche weder animiert, noch zusammensetzbar sind.

Initialisierungsmethode für Tile Objekte

Die Initialisierungsmethode (Quellcode 6.25) übernimmt Aufgaben, die für alle Typen von Tile Objekten redundant sind. Dies sind das Setzen des Ankerpunktes und das Berechnen und Setzen der Content-Größe.

Quellcode 6.25: Initialisierung

```

1 void TileObject:: initialize()
2 {
3     setAnchorPoint( CCPoint( 0.5, 0.5));
4
5     setContentSize( CCSIZE( m_nBaseSize * m_kTileSize.width,
6                             m_nBaseSize * m_kTileSize.height));
7 }
```

Rotieren von Objekten

Bei der Rotation von Objekten muss zwischen statischen und animierten Objekten unterschieden werden.

Statische Objekte besitzen für jede Rotation jeweils ein eigenes Sprite, wohingegen Animationen für jede Rotation eine eigene Animation besitzen. Daher muss hier eine Fallunterscheidung stattfinden (Quellcode 6.26).

Rotationsrichtungen statischer Objekte:

Die verschiedenen rotationsbasierenden Sprites werden dynamisch zur Laufzeit geladen. Dies bedeutet, dass nach dem Aufruf der Rotationsmethode, das Objekt eine neue, auf der Rotation basierenden Textur, zugeordnet bekommt.

Für jede Rotationsrichtung existiert ein eigenes Sprite. Dabei ist der Dateiname folgendermaßen zusammengesetzt:

- <Objektname>_<Rotationsrichtung>.<Dateityp>
- Beispiel: Object175_0.png

In Quellcode 6.26 ist das Zusammensetzen des Dateinamen zu sehen.

Rotationsrichtungen animierter Objekte:

Eine Animation für eine Richtung kann aus mehreren Einzelsprites (Frames der Animation) bestehen. Daher wird bei der Rotation animierter Objekte keine neue Textur, wie bei statischen Objekten, geladen, sondern die jeweilige Animation für die Richtung gestartet (Quellcode 6.26).

Die Animationen können mit einem Spriteanimation-Tool erstellt werden. Dabei gilt für die Isometric Tiled Map Engine folgende Konvention für die Benennung der Animationen:

- Rotation_<Rotationsrichtung>
- Beispiel: Rotation_0

Quellcode 6.26: Objekte rotieren

```

1 const String TileObject::ObjectSpriteFileExtension = ".png";
2 const String TileObject::ObjectRotationPrefix = "Rotation";
3
4 void TileObject::rotate()
5 {
6     if ( m_RotationDirection < m_nRotationDirections-1)
7     {
8         m_nRotationDirection++;
9     }
10    else
11    {
12        m_nRotationDirection = 0;
13    }
14
15    if ( m_bAnimation)
16    {
17        m_pkAnimation->play( ObjectRotationPrefix + String(
18                         m_nRotationDirection));
19    }
20    else
21    {
22        CCTexture2D* pkRotationTexture = CCTextureCache::sharedTextureCache()
23                         ->addImage( buildFilePath());
24        m_pkObjectSprite->setTexture( pkRotationTexture);
25    }
26 String TileObject::buildFilePath()
```

```

27 {
28     return m_kTileObjectImage +
29         "_" +
30         String( m_nRotationDirection ) +
31         ObjectSpriteFileExtension;
32 }
```

6.4.4 TileObjectPath

Hinzufügen der Referenz Tile

Als Referenz Tile ist die Tile zu verstehen, auf der aktuell das ConnectableObject platziert ist und der Pfad, beziehungsweise das passende Objekt, bestimmt wird. Dies muss der TileObjectPath Klasse hinzugefügt werden (Quellcode 6.27).

Quellcode 6.27: Referenz Tile

```

1 void TileObjectPath::addReferenceTile( Tile* pkTile)
2 {
3     m_pkReferenceTile = pkTile;
4 }
```

Hinzufügen benachbarter Tiles

Anhand der benachbarten Tiles kann das richtige Objekt bestimmt werden, welches für den Pfadverlauf passend ist.

Dabei wird folgendes geprüft:

- Besitzt die Tile ein TileObject? Falls dies nicht der Fall ist, wird keine weitere Berechnung benötigt.
- Ist das TileObject ein ConnectableObject? Falls das Objekt kein ConnectableObject sind ebenfalls keine weiteren Berechnungen nötig
- Wenn es sich um ein ConnectableObject handelt, wird geprüft, ob es sich um den selben Typ von Objekt handelt.
- Treffen alle vorherigen Fälle zu, dann wird anhand von Koordinaten geprüft, in welcher Richtung sich das Objekt befindet (rechts, links, unten, oben) und für die jeweilige Richtung ein bool'scher Wert auf true gesetzt (Quellcode 6.28). Anhand dieser bool'schen Werte lässt sich das passende Objekt ermitteln (Kapitel 6.4.4 Aktualisierung des tiles mit dem passenden Pfadobjekt).

Quellcode 6.28: Richtungsbestimmung

```

1 void TileObjectPath::addTile( Tile* pkTile)
2 {
3     if ( !pkTile->hasObject() )
4     {
```

```
5         return ;
6     }
7
8     if( pkTile->tileObject()->assetName() !=
9         m_pkReferenceTile->tileObject()->assetName() )
10    {
11        return ;
12    }
13
14    if( !pkTile->tileObject()->isConnectableObject() )
15    {
16        return ;
17    }
18
19    if( pkTile->getCoordinate().x < m_pkReferenceTile->getCoordinate().x
&&
20        pkTile->getCoordinate().y == m_pkReferenceTile->getCoordinate().y
)
21    {
22        m_bNorthEast = true;
23    }
24
25    if( pkTile->getCoordinate().x == m_pkReferenceTile->getCoordinate().x
&&
26        pkTile->getCoordinate().y > m_pkReferenceTile->getCoordinate().y )
27    {
28        m_bSouthEast = true;
29    }
30
31    if( pkTile->getCoordinate().x > m_pkReferenceTile->getCoordinate().x
&&
32        pkTile->getCoordinate().y == m_pkReferenceTile->getCoordinate().y
)
33    {
34        m_bSouthWest = true;
35    }
36
37    if( pkTile->getCoordinate().x == m_pkReferenceTile->getCoordinate().x
&&
38        pkTile->getCoordinate().y < m_pkReferenceTile->getCoordinate().y )
39    {
40        m_bNorthWest = true;
41    }
42 }
```

Aktualisierung des Tiles mit dem passenden Pfadobjekt

Das Ermitteln des passenden Pfadobjekts erfolgt anhand der gesetzten bool'schen Richtungsvariablen. Angenommen alle vier Richtungsvariablen sind auf *true* gesetzt, dann bedeutet dies, dass alle vier Tiles um die Referenztile herum ein Pfadobjekt besitzen. Daher muss das Referenzobjekt eine Kreuzung darstellen (Abb. 6.22). Analog wird das passende Pfadobjekt für die anderen Möglichkeiten ermittelt (Quellcode 6.29).

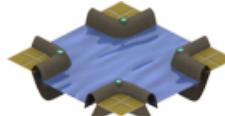


Abb. 6.22: Kreuzungsobjekt

Quellcode 6.29: Animierte Objekte

```

1 void TileObjectPath::updateTiles()
2 {
3     if ( m_bNorthEast && m_bNorthWest && m_bSouthEast && m_bSouthWest )
4     {
5         updateTexture( "_2.png" );
6     }
7     // ...

```

Nachdem das passende Objekt ermittelt wurde, wird die Textur aktualisiert (Quellcode 6.30).

Quellcode 6.30: Setzen der neuen Textur

```

1 void TileObjectPath::updateTexture( const String& rkTexturePraefix )
2 {
3     m_pkReferenceTile->tileObject()->updateTexture(
4             m_pkReferenceTile->tileObject()->assetName() +
5             rkTexturePraefix );
6 }

```

Richtungen zurücksetzen

Nachdem das passende Objekt ermittelt wurde und die Textur gesetzt ist, müssen die bool'schen Richtungsvariablen zurückgesetzt werden, so dass beim erneuten Prüfen des richtigen Objektes die Variablen wieder gesetzt werden können (Quellcode 6.31).

Quellcode 6.31: Zurücksetzen der Richtungsvariablen

```

1 void TileObjectPath::clearTiles()
2 {
3     m_bNorthEast = false;
4     m_bNorthWest = false;
5     m_bSouthEast = false;

```

```

6     m_bSouthWest = false;
7 }

```

Anwendungsbeispiel

Abbildung 6.23 zeigt mögliche Kombinationen von Flusstiles, die zusammengefügt richtig angezeigt werden und einen zusammenhängenden Fluss bilden.

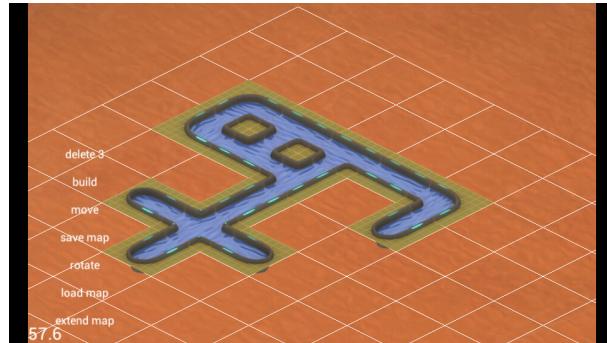


Abb. 6.23: TileObjectPath angewendet

6.4.5 TileMath

Erstellung des Tile Polygons

Für die Erstellung des Tile Polygons wird die Position der Tile, der Skalierungsfaktor, sowie die Breite und die Höhe benötigt. Auf Basis dieser Werte lassen sich die Vertices des Polygons berechnen (Quellcode 6.32).

Die Nutzung des Tile Polygons erfolgt in mehreren Bereichen:

- Erkennung eines Touches auf eine Tile (pixelgenau)
- Darstellung des Wireframes

Die Berechnung (Quellcode 6.32, Zeile 8 - 21) der Vertices (Abb. 6.24) erfolgt auf der Grundlage, des im Zentrum liegenden, Ankerpunktes der Tile.

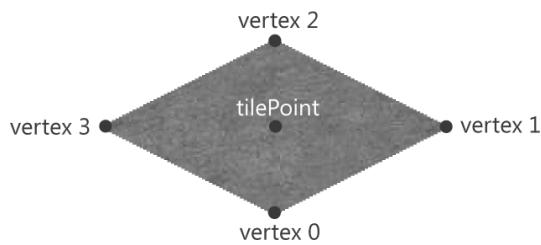


Abb. 6.24: Vertices eines Polygons

Der Skalierungsfaktor der Tiled Map ist für die Anpassung des Polygons, an die Größe der Tile in der View, verantwortlich. Auf Abbildung 6.25 sind verschiedene TilePolygone zu erkennen. Dabei stellt der

hellgrüne Bereich die Tile und der dunklere Bereich das Tile Polygon dar. Links auf der Abbildung ist die Tile mit dem Standard Skalierungsfaktor zu erkennen. Diese Tile passt zu dem TilePolygon. Mittig ist die Skalierung der Tile erhöht, das Tile Polygon jedoch nicht angepasst. Letztendlich zeigt die rechte Grafik das TilePolygon nach dem Anpassen durch Quellcode 6.32.

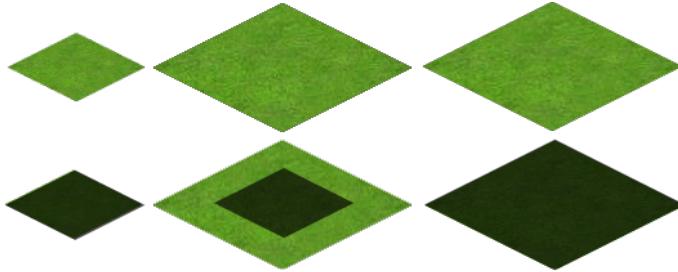


Abb. 6.25: Anpassung des Polygons an den Skalierungsfaktor

Quellcode 6.32: Tile Polygon erstellen

```

1 vector < Vector2d > TileMath::tilePolygon( CCPoint& rkTilePoint,
2                                         const float fScale,
3                                         const float fTileWidth,
4                                         const float fTileHeight)
5 {
6     vector < Vector2d > polygonVector;
7
8     polygonVector.push_back( Vector2d( rkTilePoint.x - fTileWidth / 2 *
9                                     fScale,
10                                    rkTilePoint.y ) );
11
12    polygonVector.push_back( Vector2d( rkTilePoint.x,
13                                      rkTilePoint.y + fTileHeight / 2 *
14                                      fScale));
15
16    polygonVector.push_back( Vector2d( rkTilePoint.x + fTileWidth / 2 *
17                                     fScale,
18                                     rkTilePoint.y ) );
19
20    polygonVector.push_back( Vector2d( rkTilePoint.x, rkTilePoint.y -
21                                     fTileHeight / 2 * fScale));
22
23    return polygonVector;
24 }
```

Ermitteln ob ein Punkt innerhalb eines n-Gons liegt

Die Überprüfung, ob ein Punkt innerhalb eines konvexen¹ n-Gons² liegt, ist essentieller Bestandteil der Erkennung eines Touches auf ein Tile (Quellcode 6.33).

Der Grund für die Implementierung der Erkennung für n- Gons liegt in der Erweiterbarkeit der Tile Engine. Obwohl in diesem Fall nur Quads benötigt werden, soll es zukünftig auch möglich sein, die Tile Engine auch für hexagonale Maps zu nutzen. Der Algorithmus auf Basis von n-Gons bleibt bei Quads jedoch der gleiche, was eine Erweiterbarkeit gewährleistet.

Ein Polygon besteht aus mindestens drei Vertices. Die Überprüfung erfolgt in Quellcode 6.33, Zeile 4 - 9.

Im Folgenden wird der Algorithmus erläutert:

Das Prüfen, ob ein Punkt innerhalb eines Polygons liegt, kann anhand einer geraden Linie auf der x-Achse in die rechte Richtung und der Anzahl an Edges, die diese Linie berührt, bestimmt werden.

Ist die Anzahl an berührten Edges ungerade, liegt der Punkt innerhalb des Polygons. Ist die Anzahl gerade, liegt der Punkt außerhalb des Polygons. Abbildung 6.26 und Abbildung 6.27 veranschaulicht dies.

Dieses Verfahren stellt nur eine der viele möglichen Lösungen da, ist für diese Anwendungsfall jedoch ausreichend.

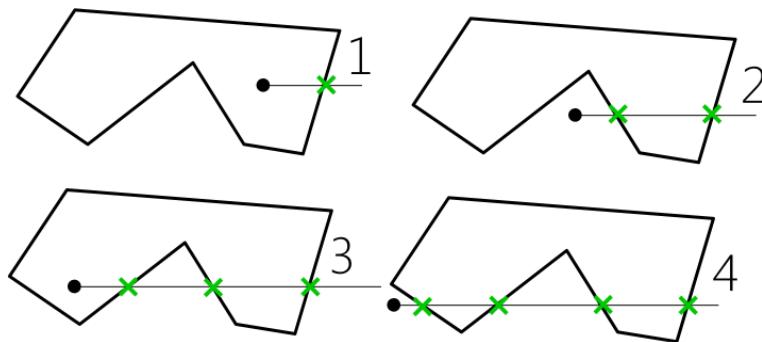


Abb. 6.26: Punkt innerhalb des Polygons (links)- Punkt außerhalb des Polygons (rechts)

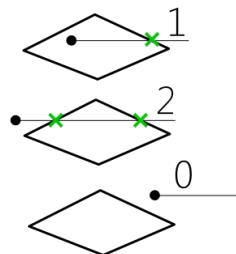


Abb. 6.27: Beispiel des Verfahrens mit einer Tile

In Tabelle 6.1 ist zu erkennen, dass alle Vertices durchlaufen werden. In jedem Durchlauf wird die jeweilige Kante (Edge) zwischen diesen zwei Vertices überprüft.

¹geometrische Figur ohne einspringende Stellen

²Ein Polygon mit n Seiten

Iteration	i (Vertex)	j (Vertex)
1	0	3
2	1	0
3	2	1
4	3	2

Table 6.1: Iteration und die dazugehörigen Vertices

Wird die erste if-Abfrage ein wenig abstrahiert (Quellcode 6.33 ab Zeile 17), ergibt sich folgendes Schema:
 $(([A1] \&& [A2]) —— ([A3] \&& [A4])) \&& ([A5] —— [A6])$

Noch einen Schritt weiter abstrahiert, ergibt sich folgendes Schema:

$([A1][A2][A3][A4]) \&& ([A5][A6])$

Die ersten Ausdrücke $[A1]/[A2]/[A3]/[A4]$ prüfen, ob sich der Punkt innerhalb der Kante befindet.

Hierbei muss jedoch zwischen zwei Arten unterschieden werden. So muss getrennt geprüft werden ob sich der y-Punkt innerhalb einer unter dem Punkt liegenden Edge befindet oder überhalb dem Punkt liegenden Edge.

Die letzten beiden Ausdrücke $[A5]/[A6]$ prüfen, ob sich die x-Position auf der rechten Seite der Edge befindet.

Sind die Ausdrücke true, dann erfolgt die Prüfung Anhand des erläuterten Algorithmus, anhand der Edges und den Schnittpunkten.

Quellcode 6.33: Punkt innerhalb eines n-Gons

```

1 bool TileMath::pointInPolygon( const CCPoint& rkPoint,
2                                     std::vector< Vector2d > polygon)
3 {
4     if( polygon.size() < 3)
5     {
6         msg::error( "TileMath::pointInPolygon",
7                     "Polygon Points < 3");
8         return false;
9     }
10
11    int j = polygon.size() - 1 ;
12
13    bool isPointInside = false;
14
15    for (unsigned int i = 0; i < polygon.size(); i++)
16    {
17        if ( ( polygon.at( i ).y() < rkPoint.y() && polygon.at( j ).y() >=
18               rkPoint.y || )
19               polygon.at( j ).y() < rkPoint.y && polygon.at( i ).y() >=
20               rkPoint.y) &&
21        ( polygon.at( i ).x() <= rkPoint.x || polygon.at( j ).x() <=
22         rkPoint.x) )

```

```

20
21     {
22         isPointInside ^= ( polygon.at( i ).x() +
23                             ( rkPoint.y - polygon.at( i ).y() ) /
24                             ( polygon.at( j ).y() - polygon.at( i ).y() ) *
25                             ( polygon.at( j ).x() - polygon.at( i ).x() ) <
26                             rkPoint.x );
27     }
28     j = i;
29 }
30
31     return isPointInside;
32 }
```

Skalierungsfaktor der Tiled Map

Für die Realisierung des Zooms muss der Skalierungsfaktor (Quellcode 6.34), um welchen die Map vergrößert oder verkleinert wird, ermittelt werden.

Dieser wird durch die Lnge der Strecke zwischen den zwei Touchpunkten vorgegeben.

Quellcode 6.34: Skalierungsfaktor ermitteln

```
1 const float TileMath::scaleFactor( CCPoint& rkFirstPoint,
2                                     CCPoint& rkSecondPoint,
3                                     const float fDistance)
4 {
5     float fLength = sqrt( float( pow( rkSecondPoint.x - rkFirstPoint.x, 2)
6                             +
7                             pow( rkSecondPoint.y - rkFirstPoint.y, 2)
8                             ) );
9
10    float fDiff = fLength - fDistance;
11 }
```

Berechnung des Scrollpunktes

Die Berechnung des Scrollpunktes ist für die Bewegung der Map innerhalb des Tile Controllers relevant. Hierbei wird Offset, um welcher die Map verschoben werden soll, in Bezug auf den Skalierungsfaktor, berechnet (Quellcode 6.35).

Quellcode 6.35: Scrollpunkt berechnen

```

4  {
5      return CCPoint( (rkFirstPoint.x - rkSecondPoint.x) * fMultFactor,
6                      (rkFirstPoint.y - rkSecondPoint.y) * fMultFactor);
7 }

```

Koordinaten der benachbarten Tiles ermitteln

Das Ermitteln der Koordinaten benachbarter Tiles ist für den Wegebau-Algorithmus erforderlich (Quellcode 6.36). Tabelle 6.2 veranschaulicht beispielhaft verschiedene Koordinaten in Bezug auf eine Eingabekoordinate.

Quellcode 6.36: Position Tile Direction

```

1 const CCPoint TileMath::tilePositionInDirection(
2                                     const int unX,
3                                     const int unY,
4                                     const TileMath::GeographicDirection eDirection)
5                                     const
6 {
7     switch( eDirection)
8     {
9         case TileMath::NORTH:
10        {
11            return CCPoint( unX - 1, unY - 1);
12        }
13        break;
14
15        case TileMath::SOUTH:
16        {
17            return CCPoint( unX + 1, unY + 1);
18        }
19        break;
20
21        case TileMath::WEST:
22        {
23            return CCPoint( unX + 1, unY - 1);
24        }
25        break;
26
27        case TileMath::EAST:
28        {
29            return CCPoint( unX - 1, unY + 1);
30        }
31        break;
32
33        default:

```

```

34         {
35             msg::warn( "TileMath::tilePositionInDirection",
36                         "Unhandled case '%d'\n",
37                         eDirection);
38
39             return CCPPoint( -1, -1);
40         }
41     }
42
43     return CCPPoint( -1, -1);
44 }
```

Richtung	x-Input	y-Input	x-Output	y-Output
Nordosten	1	0	0	0
Nordosten	3	3	2	3
Nordwesten	2	1	2	0
Nordwesten	3	3	3	2
Südosten	2	2	2	3
Südosten	2	1	2	2
Südwesten	2	3	3	3
Südwesten	0	1	1	1

Table 6.2: Ausgabekoordinaten in Bezug auf Eingabekoordinaten

6.4.6 TileController

Struktur

Diese Klasse benutzt den Target Touch Dispatcher (Kapitel 3.12 Touch Events). Wird ein Touch erkannt, wird dieser in einem Vektor gespeichert (Quellcode 6.37). So ist es möglich, auch ohne Multitouchaktivierung, in Cocos2D-x „pseudo“ Multitouch zu realisieren.

Quellcode 6.37: Initiales speichern der Touches

```
1 m_Touches.push_back( pkTouch );
```

Wird ein Touch released (*der Finger befindet sich nicht mehr auf dem Display*), wird dieser aus der Liste gelöscht (Quellcode 6.38).

Quellcode 6.38: TouchBegan

```
1 m_Touches.clear();
```

Es werden zwei Fälle unterschieden:

- Ein Touch gespeichert

Ist ein Touch gespeichert, kann davon ausgegangen werden, dass die Map nur bewegt werden soll

- Zwei Touches gespeichert

Trifft dies zu, hat der Benutzer wahrscheinlich vor zu Zoomen

Diese Touches werden nur in der TouchBegan Methode gespeichert. Hierbei sind die Initial Werte für spätere Berechnungen notwendig.

Aktuelle Werte, bezüglich der Bewegung der Finger, werden in der TouchMoved Methode ausgelesen und dort direkt verarbeitet. Eine Zwischenspeicherung dieser Werte ist nicht nötig (Quellcode 6.39).

Quellcode 6.39: Touches verarbeiten

```
1 //code mit erkennen ob touch oder multitouch
2 switch( m_Touches.size() )
3 {
4     case 0:
5     {
6         return false;
7     }
8     break;
9
10    case 1:
11    {
12        //Single Touch
13    }
14    break;
15
16    case 2:
17    {
18        //MultiTouch 2 Touches
19    }
20
21 }
```

Mehr als zwei erkannte Touch-Eingaben werden nicht benötigt, da Gesten mit mehr als zwei Toucheingaben auf mobilen Geräten kein Standard sind.

Pinch-To-Zoomen

Pinch ist eine der grundlegenden Multi-Touch-Gesten. Es ist für alle Benutzer, die ein iOS oder Android Gerät besitzen intuitiv. Abbildung 6.28 veranschaulicht die Funktionsweise vom Pinch-To-Zoom.

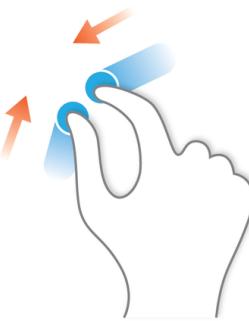


Abb. 6.28: Pinch Geste - (Quelle: Wikipedia (Creative Commons license))

Um die genaue Funktionsweise des Pinch zum Zoomen zu verstehen, sollte man sich den Standard vor Augen halten. Dieser besagt, dass der Punkt unter den Fingern in Bezug auf das zu vergößernden Objekts relativ zu dem Objekt statisch ist.

Für die **Skalierung** der Map wird für jeden Finger die initiale Touchposition (x, y) gespeichert.

Werden die Finger bewegt, wird die neue Position ebenfalls gespeichert. So lässt sich der Offset zwischen der initialen Touchposition und der Movement Touchposition berechnen. Dieser Offset beschreibt den Zoom-Faktor, jedoch nicht den entgültigen Zoom (Quellcode 6.40).

Quellcode 6.40: Skalierung berechnen

```

1 CCTouch* pkFirstTouch = m_Touches.at( 0 );
2 CCTouch* pkSecondTouch = m_Touches.at( 1 );
3 CCPoint kFirstTouchPoint = pkFirstTouch->locationInView( pkFirstTouch->
    view() );
4 CCPoint kSecondTouchPoint = pkSecondTouch->locationInView( pkSecondTouch->
    view() );
5
6 float fNewDistance = m_pkTileMath->vectorLength( rkFirstTouchPoint,
7                                         rkSecondTouchPoint );
8
9 float fOldDistance = m_pkTileMath->vectorLength( kFirstTouchPoint,
10                                         kSecondTouchPoint );
11
12 float fScaleFactor = fNewDistance / fOldDistance;
13 m_pkTileMap->setScale( m_pkTileMap->getScale() * fScaleFactor )

```

Nach den genannten Schritten ist zu erkennen, dass die Map nach dem Zoomen nicht an der Position ist, die benötigt wird (siehe definierter Standard). Daher ist es nötig eine **Translation** durchzuführen (Quellcode 6.41), um die Map relativ zu den Fingern auszurichten.

Quellcode 6.41: Translation

```

1 float fNewXPosition = rkFirstTouchPoint.x -

```

```

2           ( kFirstTouchPoint.x - m_pkTileMap->getPosition
3               () .x ) *
4               fScaleFactor;
5
6   float fNewYPosition = rkFirstTouchPoint.y -
7           ( kFirstTouchPoint.y - m_pkTileMap->getPosition
8               () .y ) *
9               fScaleFactor;
m_pkTileMap->setPosition( fNewXPosition, fNewYPosition );

```

Das bedeutet, dass man den Mittelpunkt, in Bezug auf den Skalierungsfaktor, berechnen muss und an diesem daraufhin ausrichtet.

Nach diesen zwei Berechnungen erfolgt der Zoom und die Ausrichtung der Map wie es im Standard definiert ist.

Bewegen der Map

Für die Bewegung der Map, die ebenfalls auf Touchpunkten basiert, wurde in der TileMath bereits die convertToScrollPoint Methode erläutert, welche bei der Bewegung zum Einsatz kommt.

Weiterhin werden die Methoden ccTouchBegan und ccTouchMoved benötigt. Wird ein Touch erkannt (began), ist es notwendig diesen Touchpunkt temporär zwischenspeichern (Quellcode 6.42) initializeScroll Methode.

Die Touchposition, die sich aus dem Touch ergibt, muss jedoch noch in Bezug auf die Tiled Map berechnet werden.

Quellcode 6.42: initializeScroll

```

1 m_kFirstTouch = m_pkTileMap->convertTouchToNodeSpace( pkTouch );

```

Wird eine Touchbewegung erkannt (moved), dann wird die Tiled Map Position an die neu berechnete Position gesetzt (Quellcode 6.43 und 6.44).

Quellcode 6.43: moveScroll

```

1 updatePosition( m_pkTileMath->convertToScrollPoint(
2             rkPoint,
3             m_kFirstTouch,
4             m_pkTileMap->getScale() ) );

```

Quellcode 6.44: Update Position

```

1 m_pkTileMap->setPosition( m_pkTileMap->getPosition().x + rkPoint.x,
2                             m_pkTileMap->getPosition().y + rkPoint.y );

```

Prüfung der Berechtigung für den Controller

Vor Beginn einer jeden Aktion innerhalb des Controllers, muss dieser prüfen, ob die Berechtigung eine Aktion auszuführen besteht.

Diese Prüfung basiert auf dem Object Mode. Ist dieser Modus aktiv, wird in dem Moment eine Aktion auf einem Objekt ausgeführt, wie zum Beispiel das Bewegen eines Objektes. Dabei darf sich die Map nicht bewegen und nicht vergrößern oder verkleinern.

Angenommen, das Objekt wird bewegt und die Map würde sich mit dem Objekt mitbewegen, dann wäre keine Bewegung des Objektes sichtbar. Da sich beides mit dem selber Wert bewegen würde. Daher wird bei jedem erkannten Touch geprüft, ob der ObjectMode aktiv ist (Quellcode 6.45).

Quellcode 6.45: Object Mode prüfen

```
1 if ( m_pkTileMap->isObjectMode() )
2 {
3     return false;
4 }
```

6.5 Performanceoptimierung

Performanceoptimierung ist in der Spielebranche ein sehr wichtiger Faktor. Ziel der meisten Spiele ist es, auf möglichst vielen Geräten optimal zu laufen. Dies erhöht die Zufriedenheit der Spieler und somit auch die Einnahmen durch den Verkauf.

Bei Spielen auf mobilen Geräten kommt der Faktor der Performanceoptimierung noch stärker zum Einsatz. Bei Android existieren sehr viele verschiedene Geräte. Diese haben alle eine andere Systemspezifikation. Taktung, Speicher und die Displaygröße sind zum Beispiel Faktoren, in welchem sich die Geräte sehr unterscheiden.

Technische Daten	Google G1	Motorola Razr	Samsung Galaxy S III
Display	3,2" 480 x 320 Pixel	4,3" 540 x 960 Pixel	4,8" 1280 x 720 Pixel
Betriebssystem	Android 1.6	Android 4.0	Android 4.0.4
Speichergröße	192 MB RAM	1 GB RAM	1 GB RAM
Prozessor	528 MHz	1,2 GHz Dual-Core	1,4 Ghz Quad-Core

Table 6.3: Android Geräte im Vergleich

Technische Daten	iPhone (erste Generation)	iPhone 4S	iPhone 5
Display	3,2" 320 x 480 Pixel	3,5" 640 x 960 Pixel	4,0" 1136 x 640 Pixel
Betriebssystem	iOS 3.1.3	iOS 5.1	iOS 6
Speichergröße	128 MB DRAM	512 MB DRAM	1 GB RAM
Prozessor	412 MHz	1 GHz	1 Ghz Dual-Core

Table 6.4: iPhone Geräte im Vergleich

Es ist gut zu erkennen, dass sich die Geräte (Tabelle 6.3 und 6.4) in Bezug auf ihre Leistung enorm unterscheiden. Dies macht es für Spieleentwickler komplizierter, für möglichst viele Geräte die Spiele, ohne große Abzüge, spielbar zu machen.

Im Folgenden werden Performanceoptimierungs-Verfahren erläutert, die im Rahmen der Isometric Tiled Map Engine zum Einsatz kommen.

6.5.1 Clipping- und Visibilityoptimierung

Nach Analyse des in Cocos2D-x verantwortlichen Codes für das Zeichnen der Objekte auf dem Bildschirm, war festzustellen, dass Cocos2D-x alle Elemente zeichnet, die sich in einem Szenengraphen der aktuellen Szene befinden. Das heißt, dass auch die Elemente, die momentan in der View nicht sichtbar sind gezeichnet werden.

Dies führt bei vielen Elementen zu enormen Performanceeinbrüchen. Durch Überschreiben der visit Methode (Quellcode 6.46) lässt sich das Zeichnen, auf die sich in der View befindlichen Elemente, beschränken.

Für die Darstellung dieser Methode wurde die Zeichenoptimierung sichtbar gemacht (Abb. 6.29). Dies erfolgt normalerweise außerhalb der View und ist nicht erkennbar.

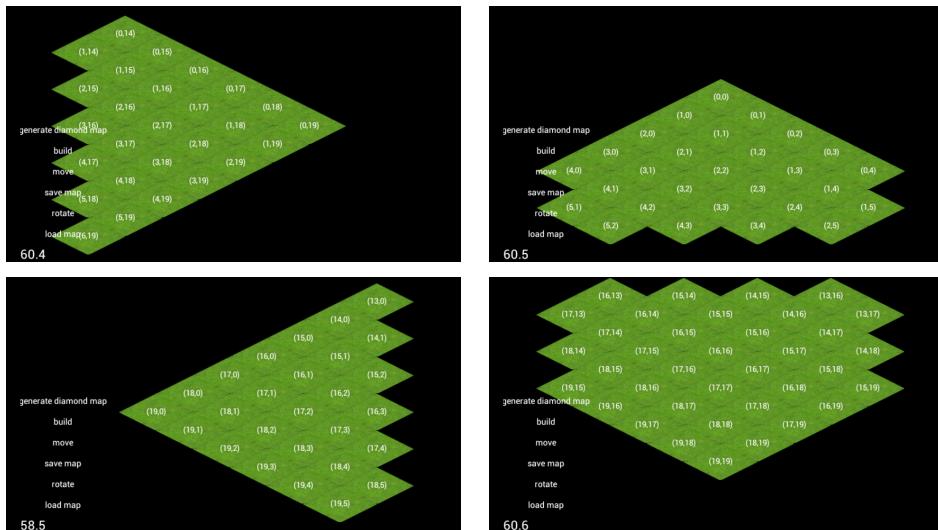


Abb. 6.29: Clipping der darzustellenden Objekte innenliegend

Im Rahmen der Isometric Tiled Map Engine führt dies zu einem enormen Performanceboost. So ist es möglich, ohne Performanceeinbrüche über 40.000 Elemente hinzuzufügen. Die Performance wird somit nur noch durch die Anzahl erzeugter Objekte reduziert.

Die Prüfung, ob ein Element in der View sichtbar ist, erfolgt durch die Koordinaten der Tile, sowie durch den Skalierungsfaktor, die Breite und Höhe des Objektes (insofern vorhanden) und die Position der Map.

Diese Werte werden daraufhin mit den Positionen der Ränder des Gerätes verglichen. Liegt ein Wert außerhalb dieses Bereichs, muss das Element nicht gezeichnet werden und die draw Methode wird nicht aufgerufen.

Quellcode 6.46: Sichtbare Objekte zeichnen

```
1 void Tile::visit()
```

```

2 {
3     CCPoint test = getParent()->getPosition();
4
5     float fMapScale = getParent()->getScale();
6
7     float fScaledTileWidth = m_unWidth * fMapScale;
8
9     float fScaledTileHeight = m_unHeight * fMapScale;
10
11    //snipp if object isnt in view
12    if ( m_bObject )
13    {
14        fScaledTileHeight = m_pkTileObject->getSpriteSize().height;
15
16        fScaledTileWidth = m_pkTileObject->getSpriteSize().width;
17    }
18    //outline with object hight
19    if ( !(( getPosition().y * fMapScale) + test.y + fScaledTileHeight < 0
20            ||
21            ( getPosition().y * fMapScale) + test.y > m_kDeviceSize.height
22            ||
23            ( getPosition().x * fMapScale) + test.x < 0 ||
24            ( getPosition().x * fMapScale) + test.x - fScaledTileWidth >
25            m_kDeviceSize.width))
26    {
27        drawWireframeNeeded = true;
28
29        CCNode::visit();
30    }
31    else
32    {
33        setVisible( false );
34    }

```

6.5.2 Iterationsoptimierung

Verfahren (1) - Vergleichverfahren

Dieses Verfahren beschreibt den Fall ohne Iterationsoptimierung. Dabei muss über die maximale Anzahl an Tiles iteriert werden.

Verfahren (2) - Visibilitätsverfahren

Um die Iterationen bei Berechnungen über die Tiles zu reduzieren, wird auf die Visibilitätsoptimierung zurückgegriffen. Innerhalb der Visibilitätsoptimierung wird berechnet, ob eine Tile sichtbar ist oder nicht.

Mit Hilfe dieser Informationen kann man nach jedem Bewegen der Map (Quellcode 6.47) alle sichtbaren Tiles ermitteln (Quellcode 6.48). Nur über diese Tiles muss somit auch iteriert werden, denn auf Tiles die nicht sichtbar sind, kann auch keine Änderung stattgefunden haben.

Quellcode 6.47: Iterationsoptimierung - Visibilität

```

1 void TileController::ccTouchEnded( CCTouch* pkTouch, CCEvent* pkEvent )
2 {
3     m_pkTileMap->updateVisibleTiles();
4 }
```

Quellcode 6.48: Iterationsoptimierung Sichtbare Tiles speichern

```

1 void TileMap::updateVisibleTiles()
2 {
3     m_kVisibleTileVector.clear();
4
5     for( unsigned int unColumn = 0; unColumn < m_unMapHeight; unColumn++)
6     {
7         for( unsigned int unRow = 0; unRow < m_unMapWidth; unRow++)
8         {
9             if ( m_kTileVector[unRow][unColumn]->getIsVisible() )
10             {
11                 m_kVisibleTileVector.push_back( m_kTileVector[unRow][unColumn] );
12             }
13         }
14     }
15 }
```

Verfahren (3) - Nachbarverfahren

Im Falle, dass sich die Performanceansprüche nochmals erhöhen, bestände innerhalb der Iterationsoptimierung noch Optimierungspotential durch das Ermitteln der benachbarten Tiles eines Objektes. Diese Tiles genügen für die benötigten Operationen in Bezug auf die Interaktion mit den Objekten.

Die Anzahl an Iterationsschritten in diesem Verfahren ist abhängig von der Basisgröße der Objekte. Je größer die Basisgröße eines Objektes ist, umso mehr Nachbarn besitzt dieses.

Verfahren im Vergleich

Im Vergleich der Verfahren (Abb. 6.30), ist zu erkennen, dass bei einer erhöhten Anzahl Tiles der Performanceboost merklich sichtbar wird.

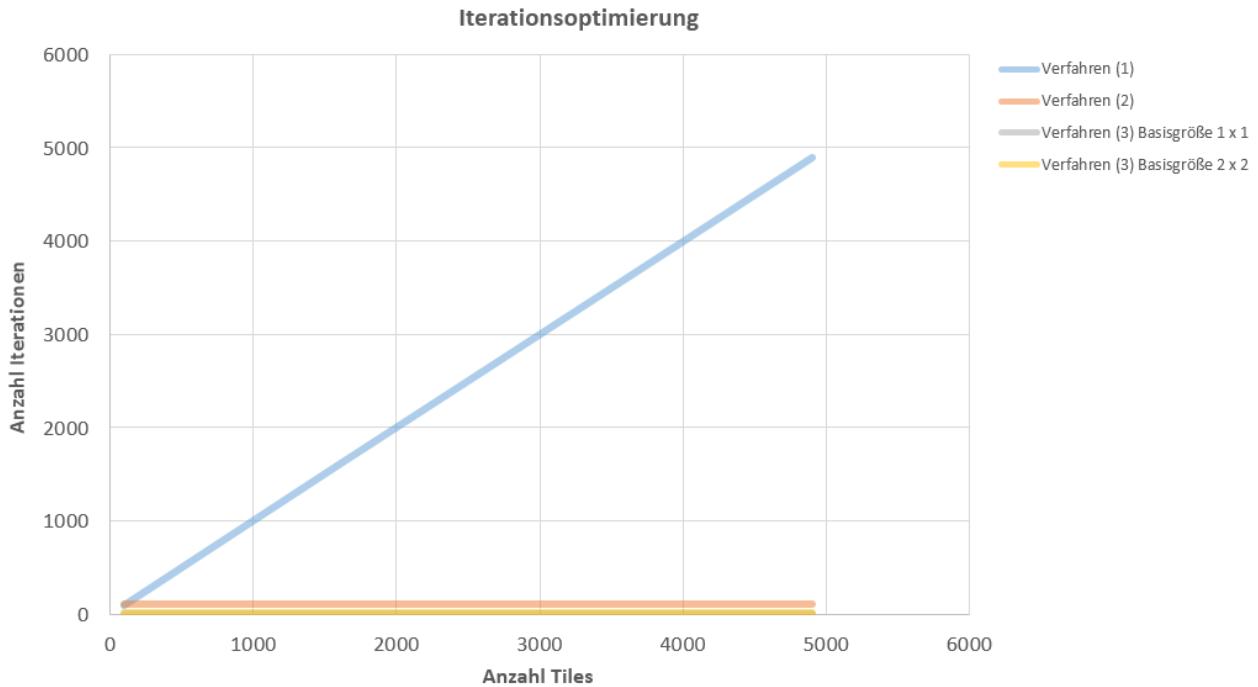


Abb. 6.30: Vergleich der verschiedenen Verfahren

Es ist zu erkennen, dass in Vergleichsverfahren ohne Iterationsoptimierung, alle Tiles durchlaufen werden müssen. Dies führt zu erheblichen Performanceeinbrüchen, da in jedem Iterationsschritt Berechnungen stattfinden.

Verfahren (2) hingegen reduziert die Iterationsschritte auf die, in der View sichtbaren Elemente.

Verfahren (3) reduziert je nach Basisgröße die Iterationsschritte auf einen Wert der unterhalb des in Verfahren (2) benötigten Iterationsschritten liegt.

Auf Abbildung 6.30 ist zu erkennen, dass Verfahren (1) im Gegensatz zu den beiden anderen Verfahren bei einer erhöhten Anzahl von Tiles zunehmend kritisch wird.

Die anderen Verfahren garantieren, unabhängig von der Tileanzahl, eine gleichbleibende Performance (in Bezug auf die Iterationen der Tiles).

6.5.3 Performancevergleich

Auf Abbildung 6.31 sind die Werte vor und nach der Performanceoptimierung im Vergleich zu sehen:

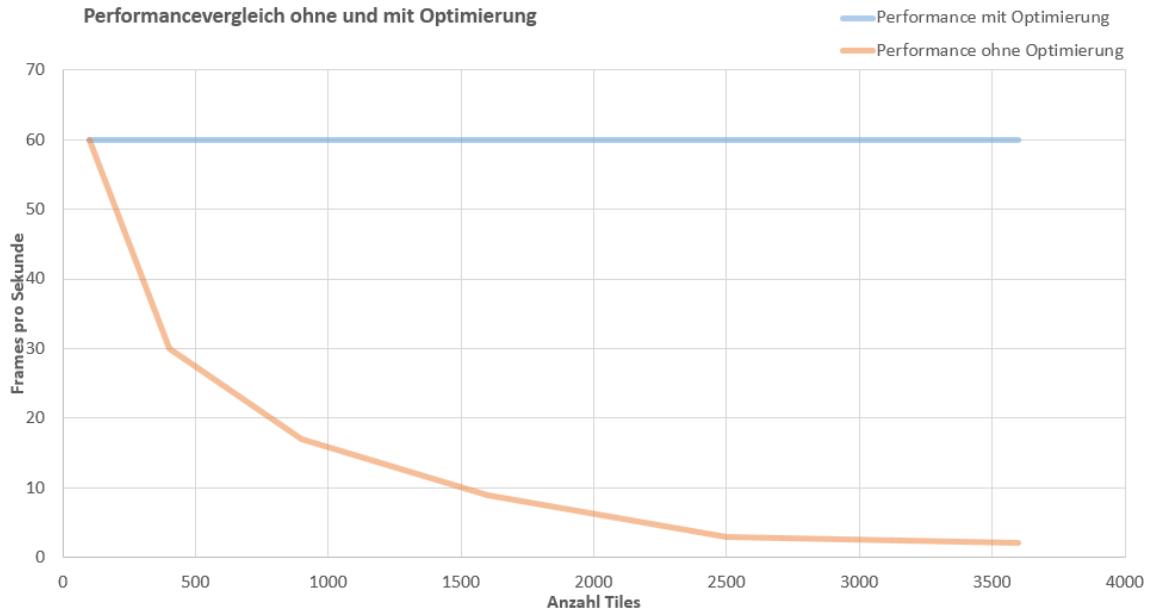


Abb. 6.31: Vergleich der verschiedenen Verfahren

6.5.4 Weitere Optimierungsmöglichkeiten

Im Rahmen der Speichernutzung besteht noch Optimierungspotential, da im Moment alle Tiles und Objekte im Speicher gehalten werden. Eine Möglichkeit die Speichernutzung zu optimieren besteht in der Nutzung des sogenannten Object Pooling.

Unter Object Pooling versteht man das Instanziieren einer Menge von Objekten, die dann im Speicher gehalten werden und bei Bedarf genutzt werden können.

6.6 Nutzung

6.6.1 Schnittstellen

Die TileEngine bietet verschiedene Schnittstellen für maximale Konfiguration- und Nutzungsmöglichkeiten an:

- Bei der Initialisierung der Tiled Map steht eine große Auswahl an Konfigurationsmöglichkeiten zur Verfügung.
- Auf Seiten der Tiled Map existiert eine Callbackfunktion. Mit Hilfe dieser kann auf die komplette Funktionalität einer Tile und eines TilesObjects zugegriffen werden.
- Weiterhin können bei der Tiled Map durch Toggle-Methoden verschiedene Modis gesetzt werden, die das User-Tiled Map Verhalten beeinflussen (bewegen, rotieren, entfernen von Objekten).

6.6.2 Tiled Map initialisieren

Das Initialisieren der Tiled Map erfolgt mit vier Parametern:

- Anzahl Tiles Breite
- Anzahl Tiles Höhe
- Größe der Tile (Breite, Höhe)
- Callback

Danach wird mit generateMap die Tiled Map mit den Tiles erzeugt. Optional kann die Position der Tiled Map auch festgelegt werden (Quellcode 6.49).

Quellcode 6.49: TileMap initialisieren

```
1 m_pkTileMap = new TileMap( 10, 10, CCSIZE( 138, 69 ),
    kOnSelectedTileCallback );
2 m_pkTileMap->initialize();
3 m_pkTileMap->setPosition( 350, 200 );
4 addChild( m_pkTileMap, 0 );
```

6.6.3 Controller initialisieren

Dem Controller muss als Parameter die Tiled Map übergeben werden. So arbeitet die Toucherkennung des Controllers unabhängig von der Toucherkennung der Tiled Map (Quellcode 6.50).

Quellcode 6.50: Controller initialisieren

```
1 TileController* pkTileController = new TileController( *m_pkTileMap );
2 addChild( pkTileController, 0 );
```

6.6.4 Hintergrund hinzufügen

Das Hinzufügen eines Hintergrundes erfolgt analog zur Erstellung eines Sprites. Es muss lediglich die addBackground Methode aufgerufen werden. Mit Hilfe dieser wird der Hintergrund der Tiled Map hinzugefügt (Quellcode 6.51).

Quellcode 6.51: Hintergrund der Map hinzufügen

```
1 CCSprite* background = CCSprite::spriteWithFile( "background/hg.jpg" );
2 background->setPosition( CCPoint( 50,-350 ) );
3 m_pkTileMap->addBackground( *background );
```

6.6.5 Hinzufügen von Objekten

Es können drei Arten von Objekten hinzugefügt werden. Statische, animierte und verbindbare Objekte. Die Logik hinter diesen verschiedenen Objekten übernimmt dabei die Engine (Quellcode 6.52).

Parameter:

- Assetpfad
- Anzahl Rotationsrichtung
- Basisgröße

Quellcode 6.52: Objekte der Map hinzufügen

```
1 m_pkTileMap->addObjectNormal( "building/1x1/Object175" , 4 , 1 );
2 m_pkTileMap->addObjectAnimated( "FastFish1.pshs" , "Swim" ,4 , 1 );
3 m_pkTileMap->addObjectConnectable( "building/1x1/Object275" , 1 );
```

6.6.6 Opacity Modus

Standardmäßig ist der Opacity Modus inaktiv, kann jedoch aktiviert werden. Zudem besteht die Möglichkeit die Deckkraft einzustellen (Quellcode 6.53).

Ist dieser Modus aktiv, wird bei jeder Bewegung der Objekte die Deckkraft aller anderen Objekte reduziert.

Quellcode 6.53: Opacity Modus

```
1 m_pkTileMap->opacityMode( true );
2 m_pkTileMap->opacityValue( 155 );
```

6.6.7 Setzen der Tiled Map Modis

Auf Basis der Tiled Engine lassen sich, mit gegebenen Mitteln, verschiedene Modis setzen (Quellcode 6.54).

- Bewegen (startMoving / endMoving / isMoving)
- Entfernen (startDeleting / endDeleting / isDeleting)
- Rotieren (startRotating / endRotating / isRotating)

Quellcode 6.54: Modis setzen

```
1 if ( m_pkTileMap->isRotating() )
2 {
3     m_pkTileMap->stopRotating();
4 }
```

```

5 else
6 {
7     m_pkTileMap->startRotating();
8 }
```

Das Setzen der anderen Modi erfolgt analog.

6.6.8 Nutzung des Callbacks

Callbacks ermöglichen sehr viele Anwendungsmöglichkeiten durch den Zugriff auf die aktuelle Tile und dessen Objekt.

So besteht zum Beispiel die Möglichkeit, ein Menü, zum Bearbeiten des Objektes, direkt über dem Objekt anzeigen zu lassen. Dies ist jedoch erst im späteren Spiel zu implementieren. Dieser Callback fungiert nur als Schnittstelle, sodass später sinnvoll die Spielelogik entworfen werden kann (Quellcode 6.55).

Quellcode 6.55: Callback nutzen

```

1 SecureCallbackTT < TestScene,
2                         const bool,
3                         tileEngineLib::Tile*>
4                         kOnSelectedTileCallback( this , &TestScene::
5                                     onSelectedTile);
6 ...
7 const bool TestScene::onSelectedTile( tileEngineLib::Tile* rkTile)
8 {
9     msg::debug( "TestScene::onSelectedTile",
10                " Tile Position: %f, %f",
11                rkTile->getPosition().x, rkTile->getPosition().y);
12
13    return true;
14 }
```

6.6.9 Beispielhafte Anwendung der Isometric Tiled Map Engine

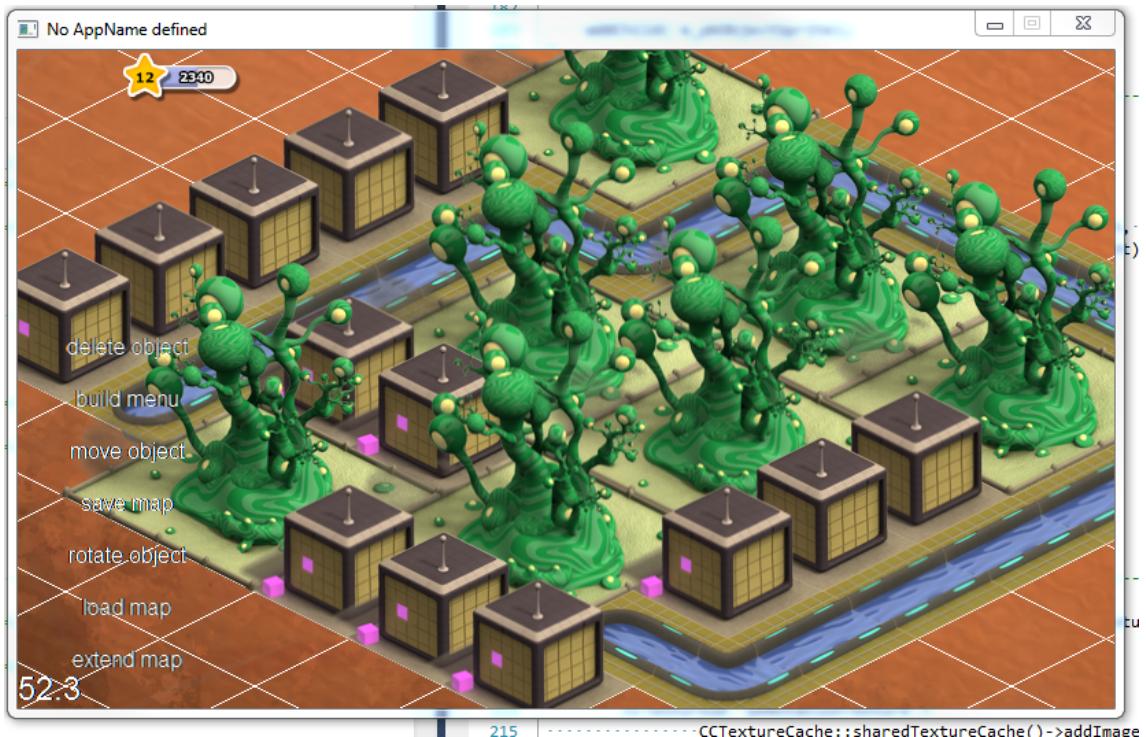


Abb. 6.32: Beispielhafte Anwendung

6.7 Ausblick und Einordnung des Cocos2D-x Frameworks

Funktionen und deren Realisierung zum Zeitpunkt der Abgabe dieser Arbeit im Überblick:

Funktion	implementiert ja/teilweise
Isometrische Ansicht	ja
Basiswinkel	ja
Bewegungsrichtung	ja
Zoom	ja
Objekte dynamisch zur Laufzeit ändern	ja
z-OrderManagement	ja
Reverse Touch Tunneling	ja
Callback Schnittstelle	ja
Different Tile Map Support	Teilweise (Diamond und Slide)
Visuelle Debugmöglichkeiten	ja
Wireframe	ja
Objekte mit verschiedenen Basisgrößen	ja
Objekte mit verschiedenen Ausrichtungen	ja
Objekte mit Animation	ja
Map speichern und laden	teilweise (speichern)

Die Grundfunktionalität der Engine ist implementiert.

6.7.1 Probleme während der Entwicklungsphase

Eines der Hauptprobleme ist das Positionierungssystem von Cocos2D-x. Dieses bezieht sich immer auf den Elternteil des hinzugefügten Kindes. Dies macht die Positionsberechnung komplex. Speziell in Verbindung von Touch- und Positions punkten ist der Umrechnungsaufwand sehr hoch.

So wird das TileObject nicht der Tile als Kind hinzugefügt, sondern der Tiled Map. Somit ist die Position der Tile und des TileObjektes gleich. Andernfalls wäre die Position des TileObjects mit der des Tiles zu verrechnen gewesen. Hier ist darauf zu achten, dass die Struktur so klein wie möglich gehalten wird. So lassen sich mühsame Berechnungen sparen.

Eine weitere Schwierigkeit besteht darin, eine geeignete Schnittstelle zur Verfügung zu stellen. Dieses Problem wird jedoch mit der Callback Methode sehr gut gelöst.

Aufgrund der hohen Anforderung an die Performance, müssen in Bezug auf den Aufbau und Softwarearchitektur Abzüge gemacht werden, da diese sich sonst negativ auf die Performance auswirken.

6.7.2 Vorteile gegenüber der in Cocos2D-x integrierten TiledMap Lösung

Im Gegensatz zu der in Cocos2D-x integrierten Lösung für Tiled Maps, bietet diese Isometric Tiled Map Engine folgende Vorteile, beziehungsweise Verbesserungen:

- Die Isometric Tiled Map Engine ist auch bei einer sehr großen Anzahl an Elementen noch performant
- Es ist nicht notwendig die Tiled Map vorher in einem Editor zu erstellen, da sie dynamisch zur Laufzeit verändert werden kann
- Die Schnittstelle und auch die damit verbundenen Möglichkeiten gehen weit über die, der in Cocos2D-x, integrierten hinaus
- Die Weiterentwickelbarkeit ist durch die Dokumentation und durch die Entstehung innerhalb der Firma einfacher
- Die Portierbarkeit auf eine andere Engine ist einfacher, da nur sehr wenige Cocos2D-x Elemente genutzt wurden (wie zum Beispiel Sprites)
- Die Tiled Engine lässt Objekte durch die Möglichkeit animierter Objekte lebhafter wirken

7 Fazit

Das Arbeiten in der Wirtschaft unterscheidet sich bekanntlich sehr von dem Arbeiten während des Studiums an der Hochschule und an deren Projekten.

Die Vorstellungen der Menschen von der Gamesbranche weichen doch sehr von denen in der Realität ab. Es ist eine sehr anspruchsvolle Arbeit, in der man sich selten auf einen Themenbereich spezialisieren kann.

Die Komplexität der Projekte innerhalb der Gamesbranche, unterscheidet sich in der Hinsicht, dass sehr schnell, auf die sich ändernden Anforderungen innerhalb des Spielemarktes, reagiert werden muss. Zudem findet gerade in der Gamesbranche noch sehr viel Grundlagenforschung statt, denn mit stetig neuen Anforderungen, wie zum Beispiel an eine Engine, stößt man immer wieder in Bereiche, in denen es noch keine etablierte Lösung gibt.

Im Rahmen dieser Arbeit, die sich hauptsächlich mit der Erweiterung einer Engine befasst, wird klar, wieviel Konzeptions- und Entwicklungsaufwand in solchen neuen Modulen steckt.

Sehr viele verschiedene Bereiche, die innerhalb eines Spieleentwicklungsstudios zusammenarbeiten, machen die Aufgaben dieser Arbeit sehr interessant. Auch innerhalb des Entwicklungsteams gibt es täglich neue Fragestellungen, die sich über sehr viele Bereiche erstrecken. Sei es Computergrafik, Buildprozesse, Entwurfs- und Strukturmuster oder tiefe Einblicke in die inneren Ebenen von Betriebssystemen und Frameworks aufgrund von Performanceoptimierungen.

Insbesondere die Arbeit an der Isometric Tiled Map Engine war in vielerlei Hinsicht sehr interessant und lehrreich. Zum einen die An- und Einbindung an die bestehende Nuroengine-X und dem Cocos2D-x Framework, aber auch die Zusammenarbeit mit 3D-Artists und deren Anforderungen an die Isometric Tiled Map Engine.

Zum Zeitpunkt dieser Bachelorarbeit war die Assetproduktion für Spiele auf Basis dieser Tiled Engine bereits angelaufen. Im Laufe der Zeit wird sie aufgrund neuer Anforderungen noch an Funktionalität zunehmen. Mit der Isometric Tiled Map Engine bietet sich für die Firma eine Möglichkeit potentielle Spiele, auf Basis von Tiled Maps, schneller zu entwickeln.

Für die Weiterentwicklung wurde eine ausführliche Dokumentation mit Doxygen erstellt.

Abschließend kann gesagt werden, dass diese Arbeit sowohl fordernd als auch fördernd war. Speziell weil sich die Spieleentwicklung über so weit gefächerte Themengebiete erstreckt war der Informationsgewinn sehr hoch.

Anhang

Abkürzungsliste

NDK Native Development Kit

SDL Software Development Kit

Lua Erweiterbare Skriptsprache

RAM Random Access Memory

DRAM Dynamic Random Access Memory

Mhz Megahertz

Ghz Gigahertz

JNI Java Native Interface

TMX Tile Map XML

GUI Graphical User Interface

JDK Java Development Kit

ADT Android Development Tools

CDT C/C++ Development Tooling

TTF True Type Font

FPS Frames per second

HTML Hypertext Markup Language

PDA Personal Digital Assistant

OpenGL ES OpenGL for Embedded Systems

JS Java Script

PNG Portable Network Graphics

OS Operating System / Betriebssystem

font Windows Font File

App Application / Anwendung

Literaturverzeichnis

- [1] Steffen Itterheim, Andreas Löw, Learn cocos2d Game Development with iOS 5, Apress, 2011
- [2] Rod Strougo, Ray Wenderlich,Learning Cocos2d, Addison-Wesley, 2012
- [3] Mario Andres Pagella, Isometric Social Real-Time Games with HTML5, CSS3, and JavaScript, O'REILLY, 2011
- [4] Meyers Scott, Effektiv C++ Programmieren, Addison-Wesley, 1998

Internetquellen

- [1] Cocos2D-x.org: Reference
<http://www.cocos2d-x.org/embedded/cocos2d-x/classes.html> (Abgerufen am 02.04.2012)
- [2] Cocos2D-x.org: Wiki
<http://www.cocos2d-x.org/projects/cocos2d-x/wiki> (Abgerufen am 04.04.2012)
- [3] Cocos2D-iphone.org: Dokumentation
<http://www.cocos2d-iphone.org/wiki/doku.php/> (Abgerufen am 10.04.2012)
- [4] Android SDK Dokumentation
<http://developer.android.com/reference/> (Abgerufen am 15.05.2012)
- [5] Android NDK
<http://developer.android.com/tools/sdk/ndk/index.html> (Abgerufen am 20.06.2012)
- [6] JNI Dokumentation
<http://docs.oracle.com/javase/1.4.2/docs/guide/jni/spec/jniTOC.html> (Abgerufen am 20.06.2012)
- [7] OpenGL ES Documentation
<http://www.khronos.org/opengles/> (Abgerufen am 25.06.2012)
- [8] Axonometric projections - technical overview
www.compuphase.com/axometr.html (Abgerufen am 22.07.2012)
- [9] Game Engine Programming
www.cs.uu.nl/docs/vakken/mgpeg (Abgerufen am 01.08.2012)
- [10] Game Programming Patterns
www.gameprogrammingpatterns.com (Abgerufen am 01.08.2012)
- [11] Isometric Projection Wikipedia
www.en.wikipedia.org/wiki/Isometric_projection (Abgerufen am 01.08.2012)

Abbildungsverzeichnis

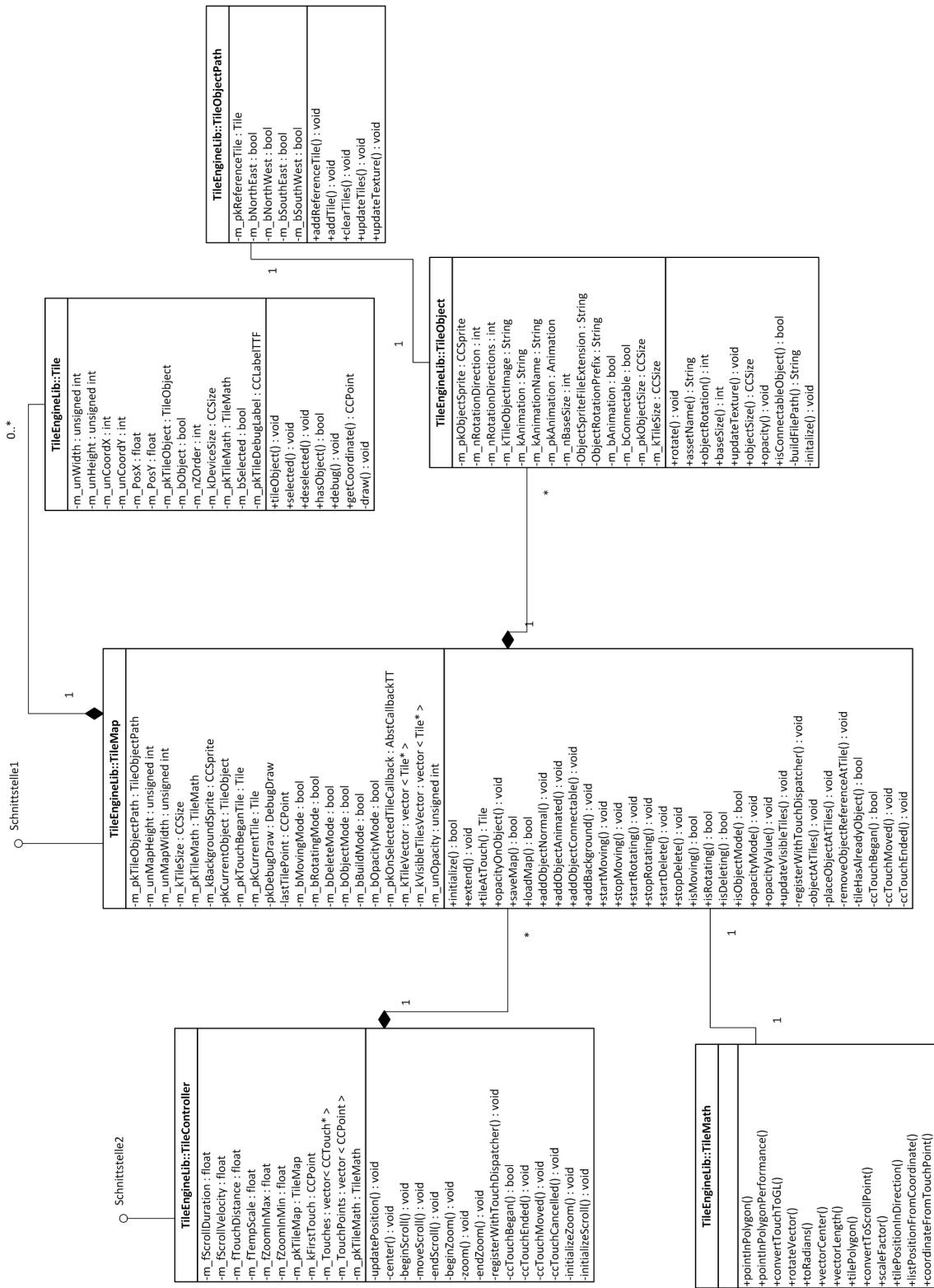
2.1 Cocos2D-x Logo Landscape	3
2.2 Aufbau des Frameworks beispielhaft mit Android	5
3.1 Layer übereinander angeordnet	12
3.2 Layer visuell sichtbar gemacht	12
3.3 Aufbau Szenestruktur eines Spiels	13
3.4 Szenengraph	14
3.5 Spritesheet - (<i>Grafikquelle: Nurogames</i>)	18
3.6 Position der Ankerpunkte	26
3.7 Rotation um Ankerpunkte (links bei 1,1) und (rechts bei 0,5,0,5)	27
3.8 Positionierung in Bezug auf Ankerpunkte des Layers (links bei 0,0) und (rechts bei 0,5,0,5)	27
4.1 JNI als Bindeglied zwischen Java und C/C++	32
5.1 Vergleich Swipedistanz (links) - Bewegungsdistanz (rechts)	40
5.2 Anwendung des ScrollMenüs in einem Spiel - (<i>Quelle: Nurogames</i>)	43
5.3 Darstellung beim Scrollen des Menüs nach links	44
5.4 Performance Analyse (FPS)	47
5.5 WebView Komponenten	49
6.1 Unterscheidung isometrisch (links) - Nicht isometrisch (rechts)	60
6.2 Isometrisch Diamond	60
6.3 Isometrisch Staggered	60
6.4 Isometrisch Slided	61
6.5 Perspektivische Darstellung (links) gegen isometrische Darstellung (rechts)	61
6.6 Regelmäßiges Pixelmuster (links) - unregelmäßiges Pixelmuster (rechts)	62
6.7 Erstellungsschritte der isometrischen Grundfläche	63
6.8 Grundfläche vergößert	63
6.9 Darstellung der z-Order	66
6.10 Darstellung des Reverse Touch Tunneling	67
6.11 Map vergößern	68
6.12 Basisgrößen (Links) 3x3, (Mittig) 2x2, (Rechts) 1x1 - (<i>Grafikquelle: Nurogames</i>)	69
6.13 Verschiedene Ausrichtungen - (<i>Grafikquelle: Nurogames</i>)	69
6.14 Einzelsprites der zusammensetzbaren Objekte - (<i>Grafikquelle: Nurogames</i>)	70
6.15 Einzelsprites einer simplen Animation mit zwei Frames - (<i>Grafikquelle: Nurogames</i>)	70
6.16 Beispiele einiger Maps	78
6.17 Darstellung der Berechnung des Versatzwertes	84
6.18 TileHopping Ankerpunkt Mittig (links) Ankerpunkt Links unten (rechts)	84
6.19 Objekte bewegen nach Definition	85
6.20 Falsches z-Order Management	88
6.21 Richtiges z-Order Management	88

6.22 Kreuzungsobjekt	95
6.23 TileObjectPath angewendet	96
6.24 Vertices eines Polygons	96
6.25 Anpassung des Polygons an den Skalierungsfaktor	97
6.26 Punkt innerhalb des Polygons (links)- Punkt außerhalb des Polygons (rechts)	98
6.27 Beispiel des Verfahrens mit einer Tile	98
6.28 Pinch Geste - (<i>Quelle: Wikipedia (Creative Commons license)</i>)	104
6.29 Clipping der darzustellenden Objekte innenliegend	107
6.30 Vergleich der verschiedenen Verfahren	110
6.31 Vergleich der verschiedenen Verfahren	111
6.32 Beispielhafte Anwendung	115

Tabellenverzeichnis

4.1	Feldbezeichner	33
6.1	Iteration und die dazugehörigen Vertices	99
6.2	Ausgabekoordinaten in Bezug auf Eingabekoordinaten	102
6.3	Android Geräte im Vergleich	106
6.4	iPhone Geräte im Vergleich	106

Klassendiagramm



Ehrenwörtliche Erklärung

Hiermit erkläre ich, **Sascha Heyer**, geboren am **30.04.1989** in **Zweibrücken**, ehrenwörtlich,

- dass ich meine Bachelorarbeit mit dem Titel

Konzeption und Umsetzung einer Isometric Tiled Map Engine für das Cocos2D-x Framework

selbstständig und ohne fremde Hilfe angefertigt habe und keine anderen als in der Abhandlung angegebenen Hilfen benutzt habe;

- dass ich die übernahme wörtlicher Zitate aus der Literatur sowie die Verwendung der Gedanken anderer Autoren an den entsprechenden Stellen innerhalb der Arbeit gekennzeichnet habe.

Ich bin mir bewusst, dass eine falsche Erklärung rechtliche Folgen haben kann.

Kaarst, 27.09.2012

Sascha Heyer

Sperrvermerk

Die vorliegende Bachelorarbeit **Konzeption und Umsetzung einer Isometric Tiled Map Engine für das Cocos2D-x Framework** enthält zum Teil Informationen, die nicht für die Öffentlichkeit bestimmt sind. Der Inhalt darf daher nur mit der ausdrücklichen schriftlichen Genehmigung des Verfassers und **Nurogames GmbH** an Dritte weitergegeben werden.

Die Arbeit ist nur den Korrektoren sowie erforderlichenfalls den Mitgliedern des Prüfungsausschusses zugänglich zu machen.