

Entwicklung eines Second Screen Gaming Frameworks

CHRISTOPH F. LIPPHART

MASTERARBEIT

eingereicht am
Fachhochschul-Masterstudiengang

INTERACTIVE MEDIA

in Hagenberg

im September 2014

© Copyright 2014 Christoph F. Lipphart

Diese Arbeit wird unter den Bedingungen der *Creative Commons Lizenz Namensnennung–NichtKommerziell–KeineBearbeitung Österreich* (CC BY-NC-ND) veröffentlicht – siehe <http://creativecommons.org/licenses/by-nc-nd/3.0/at/>.

Erklärung

Ich erkläre eidesstattlich, dass ich die vorliegende Arbeit selbstständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen nicht benutzt und die den benutzten Quellen entnommenen Stellen als solche gekennzeichnet habe. Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungsbehörde vorgelegt.

Hagenberg, am 29. September 2014

Christoph F. Lipphart

Inhaltsverzeichnis

Erklärung	iii
Vorwort	viii
Kurzfassung	ix
Abstract	x
1 Einleitung	1
1.1 Motivation	1
1.2 Zielsetzung	2
1.3 Aufbau der Arbeit	2
2 State of the Art	3
2.1 Remote Controls	3
2.1.1 Mobile Geräte zum Steuern von Computern	3
2.1.2 Mobile Geräte als Game-Controller	4
2.2 Second Screen für Fernsehgeräte	5
2.2.1 Zeitbasierte Zusatzinformationen	6
2.2.2 Interaktives Fernsehen	7
2.3 Second Screen Gaming	8
2.3.1 Ursprünge von Second Screen Gaming	8
2.3.2 Integriertes Second Screen Gaming	12
2.3.3 Standalone Second Screen Apps	14
3 Kategorisierung von Second Screen Gaming	16
3.1 Interaktion mit dem Second Screen	16
3.1.1 Passiv	16
3.1.2 Aktiv	17
3.2 Second Screens in Abhängigkeit eines Spiels	18
3.2.1 Abhängig	18
3.2.2 Unabhängig	19
3.3 Einordnung von Second Screen Anwendungen	19
3.3.1 Aufgabenliste	19

3.3.2	Spiele für Zwischendurch	19
3.3.3	Online-Mehrspieler	20
3.3.4	Kartenvisualisierung	20
3.3.5	Inventar	22
3.3.6	Sammelbare Zusatzinhalte	23
3.4	Gegenüberstellung	24
4	Eigener Ansatz	25
4.1	Abgrenzung	25
4.2	Szenarien anhand eines Jump and Run	26
4.2.1	Zusatzinformationen abrufen	26
4.2.2	Rätsel lösen	26
4.2.3	Unterwegs Spielerfahrungspunkte sammeln	27
4.2.4	Ein Making-of ansehen	28
4.2.5	In einem Mehrspieler mitspielen	28
4.2.6	Zusammenfassung	30
4.3	Anforderungen	31
4.3.1	Funktionale Anforderungen	31
4.3.2	Nicht funktionale Anforderungen	31
4.4	Entwurf des Frameworks	32
4.4.1	Slave-Konzept	32
4.4.2	Subapp-Konzept	35
4.4.3	Begriffsübersicht	37
4.5	Subapp	38
4.5.1	Logik	38
4.5.2	Szene	38
4.5.3	Versionsverwaltung	41
4.5.4	Ressourcenbündel	41
4.5.5	Ablage	42
4.5.6	Sandbox	44
5	Kommunikation	46
5.1	Wahl des Übertragungsweges	46
5.2	Hauptverbindung	47
5.3	Eventprotokoll	49
5.3.1	Event	49
5.3.2	Übertragung	50
5.3.3	Sicherstellung der Verbindung	50
5.4	Konkreter Ablauf	51
5.4.1	Erkennung	51
5.4.2	Verbindungsanfrage	52
5.4.3	Annahme der Verbindung	53
5.4.4	Installation	54
5.4.5	Beenden der Verbindung	56

5.5	SDK	57
5.5.1	Hauptverbindung und Installation der Subapp	57
5.5.2	Vermeidung einer Blockierung	58
5.5.3	Nachrichtensystem	59
6	Implementierung	60
6.1	Die Frameworks	60
6.2	Verwendete Technologien	60
6.2.1	Game-Engine: xis-engine	60
6.2.2	Kommunikationsbibliothek: Boost.Asio	61
6.2.3	Skriptsprache: Lua	61
6.2.4	Hauptspiel: Elements	63
6.3	Aufbau der Frameworks	64
6.3.1	NetEventService	64
6.3.2	Second Screen SDK	68
6.3.3	Second Screen Manager	69
6.3.4	Erweiterungssystem	72
7	Umsetzung der Prototypen	75
7.1	Begriffsdefinition	75
7.2	Umsetzung des Karten-Prototyps	75
7.2.1	GUI-System	76
7.2.2	Anpassen der Darstellung unter Einsatz von Events	76
7.2.3	Integration in Elements	77
7.2.4	Zurückkehren zum Hauptmenü	77
7.3	Umsetzung des Abenteuer-Prototyps	77
7.3.1	Vorbereitung	78
7.3.2	Aufbau der Szene	79
7.3.3	Inventar	79
7.3.4	Puzzle	80
7.4	Installation der Abenteuer-Subapp	82
7.4.1	Vorbereitung	82
7.4.2	Durchführung	84
8	Evaluierung	86
8.1	Universelle Second Screen App	86
8.1.1	Umsetzung einer Subapp	86
8.1.2	Verbindungsmenü	87
8.1.3	Spieldaten und Ressourcen der Subapp	87
8.2	App kann vom Hauptspiel gesteuert werden	88
8.2.1	NetEventService	88
8.2.2	Schwierigkeiten einer asynchronen Kommunikation	89
8.3	Unabhängig verwendbar	89
8.4	Plattformunabhängiges Framework	90

8.5	Sicherheit des Second Screen Managers	90
8.5.1	Unbefugte Installationen von Subapps vermeiden	90
8.5.2	Spieldaten der Subapp schützen	91
8.5.3	Schutz der privaten Daten des mobilen Geräts	91
8.6	Entwicklerfreiheit	91
8.6.1	Freie Wahl der Game-Engine	92
8.6.2	Ermöglichen neuer Spielmechaniken	92
8.6.3	Freiheiten in der Darstellung der Subapp	92
8.6.4	Offenheit der Anwendungsmöglichkeit	92
8.7	Entwicklerfreundlichkeit	93
8.7.1	Etablieren der Hauptverbindung	93
8.7.2	Direkte Steuerung	93
8.7.3	Subappentwicklung	94
8.8	Benutzerfreundlichkeit	95
8.8.1	Interface des Second Screen Managers	95
8.8.2	Installationsvorgang	95
8.8.3	Deinstallation einer Subapp	96
8.9	Zusammenfassung	96
9	Schlussbemerkungen	97
9.1	Zusammenfassung	97
9.2	Ausblick	98
A	Inhalt der CD-ROM	101
A.1	Masterarbeit	101
A.2	Anwendungen	101
A.3	Quellcode	101
A.4	Sonstiges	101
	Quellenverzeichnis	102
	Literatur	102
	Software und Programmbibliotheken	103
	Online-Quellen	104

Vorwort

Als ich vor 8 Jahren den Traum von Hollywood verfolgte und Kurzfilme am Kunstgymnasium drehte, war mir noch nicht im Geringsten bewusst, dass ich Geschichten mit Videospielen erzählen werde. Das änderte sich, als mir mein Freund, Thomas Pechtl, die Kunst des Programmierens beibrachte, wofür ich mich an dieser Stelle aufrichtig bedanken möchte. Dank dieser neuen Fähigkeit, verschlug es mich an den FH Oberösterreich Campus Hagenberg. Dort lernte ich mein Hobby, Videospiele zu zocken, mit meiner Leidenschaft des Programmierens zu vereinen.

Aus Hollywood wurde Silicon Valley – und eine neue Ära der Videospielindustrie ergriff die Medien: Second Screen Gaming. Umgeben von den mächtigsten Publishern der Branche, wollte ich, dass es auch der Indie-Game-Szene möglich ist, Second Screens anzubieten. Ich hoffe diese Arbeit kann Entwicklern von Independent Games ein hilfreiches Werkzeug bereitstellen, um mit „den Großen“ mitmischen zu können.

Meine Reise nach Hagenberg und die Entstehung dieser Arbeit wären ohne die Hilfe meiner Eltern niemals möglich gewesen. Deswegen möchte ich ihnen von ganzem Herzen dafür danken, dass sie mich in den fünf Jahren meines Studiums immer unterstützt haben.

Ein großer Dank gilt auch meinem Betreuer Rimbert Rudisch-Sommer, der durch seine konstruktiven Vorschläge und sein wertvolles Feedback einen wesentlichen Beitrag zum Entstehen dieser Arbeit geleistet hat.

Weiters möchte ich mich bei meiner Freundin Angelika bedanken, die durch ihre kontinuierliche Ermutigung dafür gesorgt hat, dass ich mein Ziel trotz so mancher Sackgassen, in die der Entstehungsweg dieser Arbeit geführt hat, nie aus den Augen verloren habe.

Ebenso bedanke ich mich bei Hermann Neustifter, der sich mit seinem geschulten Auge auf den zahlreichen Seiten auf Fehlersuche begab und meine kreative Rechtschreibkunst ins rechte Licht rückte.

Zu guter Letzt danke ich meinen treuen Freunden aus der Steiermark, welche mir von der Ferne Halt, Energie und Kraft geschenkt haben. Ich freue mich die Strapazen gut überstanden zu haben und erfolgreich in die Zukunft blicken zu können, denn

*„Mehr als die Vergangenheit interessiert mich die Zukunft,
denn in ihr gedenke ich zu leben.“ – Albert Einstein*

Kurzfassung

Wenn Spielfirmen einen Second Screen für einen ihrer Triple-A Titel anbieten, handelt es sich bevorzugt um eine eigenständige mobile App. Entwicklungen mit den universellen Second Screen Apps von Sony und Microsoft sind an die jeweilige Spielkonsole gebunden. Um speziell für die Indie-Game-Szene eine Möglichkeit zu schaffen Second Screens in Spiele zu integrieren, wird ein einfach zu verwendendes System benötigt, bei welchem nicht für jede mobile Plattform eine eigene App entwickelt werden muss. Dafür wurde ein Framework entworfen, welches sich aus einem Second Screen und einem SDK für dessen Steuerung zusammensetzt. Der Second Screen ist eine universell einsetzbare, plattformunabhängige, mobile App, welche für unterschiedliche Spiele als Second Screen agiert. Das verhindert, dass für jeden benötigten Second Screen eine eigenständige App je mobiler Plattform programmiert werden muss und ermöglicht zudem unter Einsatz des Frameworks, dass Second Screen Anwendungen mit geringstem Aufwand umgesetzt werden können.

Abstract

If game studios offer a second screen for one of their Triple-A games, they tend to implement it as a unique mobile app. Second screen gaming experiences that were developed using the universal second screen apps by Sony or Microsoft depend on their game consoles. To provide the opportunity for indie game developers to integrate second screen experiences in their own games without the necessity of developing a unique app for each platform requires an easy-to-use system. Therefore, a special framework was developed which consists of a cross-platform mobile app and its controlling SDK. The mobile app offers the ability to act as second screen to a diversity of games. Because of that, there is no more need to draft a unique app per platform for each second screen. Furthermore, using the framework helps to create second screen gaming applications the easy way.

Kapitel 1

Einleitung

In der Videospielbranche boomen Second Screen Apps seit der Einführung der *WiiU* von *Nintendo*. Es ist eine faszinierende, alte Technologie, welche mithilfe der mobilen Geräte neu aufgerollt wird. Die Entwicklungen von Second Screen Anwendungen reichen bis ins Jahr 1982 zurück, als *Nintendo* erste Handhelds veröffentlichte, die zwei Bildschirme besaßen.

1.1 Motivation

Heute ermöglicht Second Screen Gaming u. a. das Transferieren der HUD-Elemente vom Spielbildschirm auf einen Smartphone- oder Tabletbildschirm. Dadurch kann die Immersion gesteigert werden – andernfalls könnte deren Überlagerung am Spielbildschirm eine Grenzsicht zwischen Spieler und der virtuellen Welt darstellen. Zudem wurde es möglich mit einem mobilen Gerät gegen einen PC- oder Konsolenspieler ein Mehrspieler-Spiel anzutreten.

Dafür wurden von Triple-A-Spiele-Hersteller¹ bereits interessante Lösungsvorschläge entwickelt. Jedoch wird die Kommunikation zwischen dem Spiel und dem Second Screen i. Allg. über Onlineserver ermöglicht. Bei den Recherchen wurde in diversen Spielen beobachtet, dass u. a. deren Kartenvisualisierungen² am Second Screen Latenzprobleme aufweisen. Teilweise handelte es sich bei der dargestellten Position um die Stelle, an der sich der Spieler vor fünf Sekunden befunden hatte. Anzustreben wäre jedoch eine möglichst aktuelle Positionsanzeige. Umgesetzt wurden diese Second Screens jeweils als eigenständige mobile App. Von den Spielkonsolenherstellern *Sony* und *Microsoft* wird hingegen das Konzept verfolgt, eine Second Screen App je aktueller mobiler Plattform für Konsolenspiele anzubieten.

Nachdem es auf der einen Seite umständlich ist, für jede mobile Platt-

¹Triple-A Spiele sind i. Allg. hochqualitative Spiele von großen Spieleherstellern, welche mit einem außergewöhnlich hohen Budget finanziert wurden.

²In der Kartenvisualisierung wird die Spielwelt abstrakt am Second Screen dargestellt und die aktuelle Position des Spielers markiert.

form eine eigene App als Second Screen zu entwerfen und auf der anderen Seite das Problem besteht, dass eine Second Screen Programmierung unter Verwendung der Konsolen-eigenen Second Screen Apps auf Konsolenspiele eingeschränkt ist, wird in der vorliegenden Arbeit eine Lösung vorgestellt, welche es mit wenig Aufwand zulässt einen Second Screen plattformunabhängig in einem eigenen Spiel zu integrieren.

1.2 Zielsetzung

Es stellt sich nun die Frage, wie ein plattformunabhängiger Second Screen entwickelt werden kann, dessen Integration in ein Spiel mit geringem Aufwand möglich ist und der zudem keine eigene Programmierung für jede mobile Plattform benötigt. Diese Arbeit schlägt daher eine universelle mobile App vor, welche für unterschiedliche Spiele als Second Screen fungieren kann. Um den benötigten Entwicklungsaufwand der Kommunikation mit der mobilen App reduzieren zu können wird ein SDK vorgestellt, welches in ein Spiel integriert werden kann. Im Zuge dieser Arbeit wurde ein solches System entworfen und auf seine Praxistauglichkeit überprüft.

1.3 Aufbau der Arbeit

In der vorliegenden Arbeit wird zunächst in Kapitel 2 auf Anwendungen eingegangen, welche die gleichzeitige Verwendung eines PCs und eines mobilen Geräts erfordern. Zudem wird die Bedeutung von *Second Screen* in Bezug auf Fernsehkonsum und Videospiele anhand von Beispielen näher erklärt.

Um die gefundene Vielfalt an Second Screen Gaming Anwendungen überschaubarer erfassen zu können, werden in Kapitel 3 vier Kategorien aufgestellt, denen Anwendungsbeispiele zugeordnet werden.

Anhand dieser werden in Kapitel 4 Szenarien formuliert und Anforderungen an das Framework aufgestellt. Unter der Berücksichtigung der Szenarien und der Anforderungen werden zwei Konzepte präsentiert, welche die Entwicklung eines eigenen Second Screens ermöglichen. Die benötigte Kommunikation – zwischen Spiel und Second Screen – wird in Kapitel 5 vorgestellt und unter Verwendung des SDKs abstrahiert.

Anschließend wird in Kapitel 6 eine mögliche Implementierung der vorgestellten Konzepte gezeigt und anhand der Umsetzung von Prototypen in Kapitel 7 auf ihre Praxistauglichkeit überprüft.

In Kapitel 8 folgt schließlich die Evaluierung der entstandenen Frameworks anhand der gestellten Anforderungen.

Den Abschluss bildet eine Zusammenfassung der Arbeit in Kapitel 9, in welchem darüber hinaus mögliche Verbesserungen und Erweiterungen vorgeschlagen werden.

Kapitel 2

State of the Art

Dieses Kapitel beschäftigt sich mit der Verwendung von Smartphones und Tablets als Zusatzgeräte für TV, PC und Videospielkonsolen. Der erste Abschnitt stellt Beispiele vor, die eine Verwendung von Smartphones und Tablets als Remote Controller für einen Computer oder eine Spielkonsole ermöglichen. Im Anschluss werden Möglichkeiten aufgezeigt wie mobile Geräte während des Fernsehkonsums zusätzliche Informationen bereitstellen können. Abschließend wird auf die Entstehung der Nutzung eines zweiten Bildschirms während dem Videospielen von den Anfängen bis heute eingegangen. Dabei wird der Begriff *Second Screen* für den Videospielbereich definiert um ihn anschließend in der Konzeption und Implementierung der eigenen Anwendung zu verwenden.

2.1 Remote Controls

Kabellose Eingabegeräte erleichtern die Steuerung von Computern, wenn der Computerbildschirm für eine Präsentation auf eine große Fläche projiziert oder auf einem Großbildfernseher dargestellt wird. [16] beschreibt, dass es sehr unhandlich ist, eine Funktastatur während der Präsentation in Händen zu halten und deswegen komfortablere Lösungen entwickelt wurden. Dazu zählen die *Remote Control Apps*, die auf einem Smartphone oder Tablet ausgeführt werden können um einen Computer zu steuern.

2.1.1 Mobile Geräte zum Steuern von Computern

Heutige Smartphones verfügen üblicherweise über die Funktion von einem drahtlosen Netzwerk, welches ihnen erlaubt mit anderen Geräten über peer-to-peer Ad-Hoc Verbindung oder Wi-Fi-Access-Points zu kommunizieren [16]. Ein dafür entwickelter Maus- und Tastatursimulator am Computer kann so Befehle von Smartphones aus dem Netzwerk entgegennehmen und ausführen.

Zum Beispiel schafft die App *Remote Control Collection* [32] von *Steppschuh* die Möglichkeit Tastatur- und Mausbefehle am Computer mit dem Smartphone oder Tablet über Touchgesten einzugeben. Darüber hinaus kann in der PRO Version der Bildschirm vom Computer am mobilen Gerät dargestellt werden. Remote Control Collection wurde bereits für Android und BlackBerry veröffentlicht. Die App benötigt ein Serverprogramm am Computer, welches die Befehle über das Netzwerk entgegennimmt und ausführt. Aktuell ist ein Serverprogramm für Windows, unter dem Namen *RemoteControlServer*¹, verfügbar. An der iOS App und einem Serverprogramm für Linux und Mac OS wird gearbeitet.

2.1.2 Mobile Geräte als Game-Controller

Mit der zuvor erklärten Technologie kann ein Smartphone oder Tablet auch als ergänzendes oder ersetzendes Eingabegerät während des Videospiels benutzt werden. Zum Beispiel besteht für den Anwender die Möglichkeit ein Spiel statt mit Maus- oder Tastatureingaben mit Touchgesten oder der Orientierung des mobilen Gerätes zu steuern. Daraus ergeben sich neue Interaktionsmöglichkeiten welche bewusst vom Spiel eingesetzt werden können.

Zwei Remote-Control-Apps werden nachfolgend vorgestellt die in Bezug zu Videospielen stehen. Anschließend folgt eine kurze Erklärung, wie die Orientierung von einem Smartphone oder Tablet eruiert und benutzt werden kann.

Power-Grid

Mit der mobilen App *Power-Grid* von *ROCCAT* können PC Spiele mittels Smartphone oder Tablet gesteuert werden [31]. Ein Anwender kann eine benutzerdefinierte Eingabemaske erstellen (s. Abb. 2.1), welche aus Buttons besteht, die je mit einer Bezeichnung, einer Grafik und einem Befehl ausgestattet sind. Angeordnet werden die Buttons in einem Grid. Für den Windows PC gibt es eine Serveranwendung, welche die Befehle entgegennimmt und ausführt.

Das Besondere bei *Power-Grid* ist einerseits, dass die benutzerdefinierten Eingabemasken, die sogenannten *Grids*, über die Community geteilt werden können und andererseits gibt es eine SDK, die eine individuellere Anpassung ermöglicht.

Beyond Touch

Die Begleitapp *Beyond Touch* zum *PlayStation 3* (PS3) Spiel *Beyond Two Souls* ist für iOS und Android 2013 erschienen. Beyond Touch ersetzt aus-

¹Der RemoteControlServer kann unter <http://remote-control-collection.com/download/server/windows/> für Windows heruntergeladen werden.



Abbildung 2.1: Verschiedene benutzerdefinierte Oberflächen, welche für die *Power-Grid* App erstellt werden können.

schließlich den PlayStation 3 Wireless-Controller und ermöglicht eine alternative Spielsteuerung mit Toucheingabe. Um die Geräte miteinander zu verbinden, müssen sich die PS3, mit dem laufenden Spiel, und das Gerät auf dem *Beyond Touch* ausgeführt wird, im selben Netzwerk befinden.

Orientierung des Geräts als Eingabe

Heutige Smartphones und Tablets besitzen einen Beschleunigungssensor und einen digitalen Kompass. Damit lässt sich die Orientierung des Gerätes in drei Freiheitsgraden im Raum feststellen. Diese Funktion wird unter anderem dafür genutzt um zu unterscheiden ob das Gerät im Hochformat oder Querformat gehalten wird um den Bildschirm dementsprechend anzupassen.

Spiele können die Information der Orientierung als Steuerung für Spielelementen nutzen. Zum Beispiel könnte das Gerät in einem Rennspiel als Lenkrad fungieren, welches durch Neigen ein Fahrzeug nach links oder rechts lenkt. Nähere Informationen und welche Möglichkeiten es gibt ein UFO in einem dreidimensionalen Raum mit der Orientierung zu steuern sowie dessen Evaluierung können in [6] nachgeschlagen werden.

2.2 Second Screen für Fernsehgeräte

Der Begriff *Second Screen* (Zweiter Bildschirm) beschreibt die Verwendung von Smartphones und Tablets während des Fernsehens um beispielsweise Mails abzurufen, soziale Plattformen zu besuchen oder Zusatzinformationen zu laufenden Sendungen abzufragen. Laut einer Studie von *Nielsen*² im Q2 2013, verwenden ca. die Hälfte der Smartphonebesitzer (46%) und der Tabletbesitzer (43%) ihre Geräte während dem Fernsehen als Second Screen.

²<http://www.nielsen.com/us/en>

Nachfolgend wird auf verschiedene Systeme eingegangen, die es erlauben, entweder Zusatzinformationen zu dem im Fernsehen Gesehenen zu erhalten, oder interaktiv an einer Sendung teilzunehmen.

2.2.1 Zeitbasierte Zusatzinformationen

Second Screen Apps können die Suche nach Zusatzinformationen zum Fernsehprogramm vereinfachen indem sie die Informationen zeitgerecht anbieten. Dazu wird die App zu einem laufenden Film zeitlich synchronisiert und kann so vorprogrammierte Aktionen ausführen. Die Synchronisation wird je nach App manuell vom Nutzer oder automatisch mit *Audio-Sync* vorgenommen. Beim *Audio-Sync* synchronisiert sich die App anhand der Tonspur vom Film von selbst.

APP – der Film

Diese Technik wird unter anderem in der App *APP - der Film*³ verwendet. Die begleitende App zum gleichnamigen Film aus den Niederlanden beinhaltet Trailer, Informationen über Schauspieler und Fotos vom Film. Zu definierten Zeiten werden während dem Film zusätzliche Kameraperspektiven einer Szene am mobilen Gerät angezeigt, die dem Zuseher mehr Einblicke und Hintergrundinformationen zur Geschichte geben. Wird die App auf einem Smartphone mit aktivierter Vibrationsfunktion ausgeführt, so vibriert das Gerät, wenn eine SMS-Konversation zweier Charaktere angezeigt wird. Gegen Ende des Films geben diese Konversationen dem Zuseher mehr Aufschluss über den Antagonisten bevor der Protagonist herausfindet wer sein wahrer Gegner ist.

Smart-TV

Das System von Bernadette Zlabinger in Abb. 2.2 zeigt, dass Zusatzinformationen auch direkt von einem Smart TV auf ein mobiles Gerät gesendet werden können. Dafür wird ein Meta-Daten-Server benötigt, auf dem Informationen zu einem Film anhand von Zeitstempel abgelegt sind. Auf einem weiteren Server liegen die Filmdaten bereit.

Während der Film vom Media-Server abgespielt wird fordert die Second Screen App stetig Zusatzinformationen am Smart-TV an. Sind laut des aktuellen Zeitstempel des Films am Meta-Daten-Server Einträge vorhanden sendet der Smart-TV diese Daten zum anfragenden Smartphone oder Tablet.

Nähere Informationen zu diesem System können [18] entnommen werden.

³<http://www.appdefilm.nl>

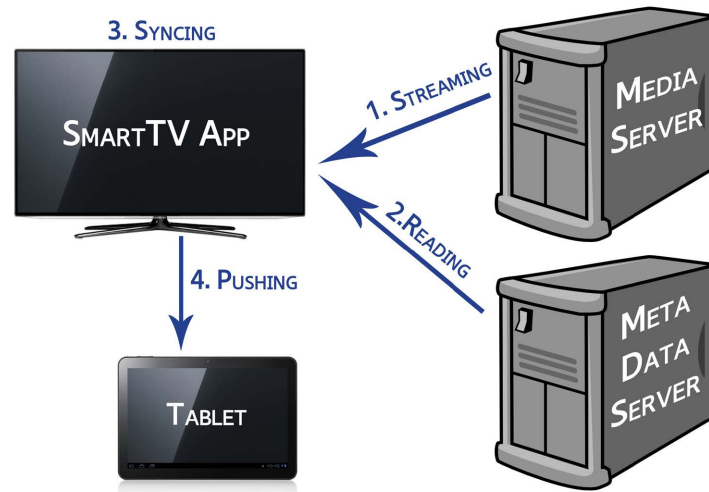


Abbildung 2.2: Die Architektur des Systems von Zlabinger setzt sich aus *Streaming*, *Reading*, *Syncing* und *Pushing* zusammen [18].

2.2.2 Interaktives Fernsehen

Neben dem Recherchieren nach Hintergrundinformationen zeigen aktuelle Trends zunehmend das Interesse an mehr Interaktionsmöglichkeiten während des Fernsehkonsums. Zum Teil gibt es schon Angebote, bei denen die Zuseher das aktuelle Fernsehprogramm aktiv mitgestalten können. Dazu zählen eine Reihe von Fernsehshows, bei welchen durch einen Anruf oder eine SMS eine Stimme abgegeben werden kann. Das Ergebnis der ausgewerteten Stimmen könnte bestimmen welcher Kandidat in die nächste Runde eines Wettbewerbes kommt oder wie eine fiktive Handlung fortgesetzt wird. In *Call-In* Sendungen können Zuschauer Bestandteil des Programms werden, indem sie live mit dem Moderator telefonieren und/oder als Kandidat einer Quizsendung teilnehmen.

Als Beispiel gibt es seit 12. Mai 2014 in Deutschland die TV-Show *Quizduell* [46] von ARD⁴. Dabei treten vier Kandidaten aus dem Studio gegen das *Team Deutschland* an. Dieses Team besteht aus Nutzern der Smartphone-App *Quizduell* [19], die online mitspielen.

Der ORF⁵ ermöglichte es seinen Fernsehzusehern während den live übertragenen *TV-Duellen*, der Nationalratswahl 2013 und der Europawahl 2014, mit Twitterpostings Fragen zu stellen, die zum Teil von den geladenen Gästen beantwortet werden mussten.

⁴Arbeitsgemeinschaft der öffentlich-rechtlichen Rundfunkanstalten der Bundesrepublik Deutschland (<http://ard.de>)

⁵Österreichischer Rundfunk (<http://orf.at>)

2.3 Second Screen Gaming

Second Screens können direkt während dem Videospielen verwendet werden um das Spiel in vielfältiger Weise zu ergänzen. Ähnlich zum Second Screen für den TV-Bereich handelt es sich hierbei heute oft um eine App auf einem mobilen Endgerät, welche über das Internet oder Heimnetzwerk mit dem Spiel verbunden ist. Dadurch können beispielsweise zusätzliche Inhalte über den zweiten Bildschirm verfügbar gemacht werden.

Eine andere Anwendungsmöglichkeit sind Aufgaben, die vom externen Bildschirm aus gelöst werden müssen. Währenddessen können die Möglichkeiten zur Interaktion mit dem Interface auf dem Hauptbildschirm teilweise eingeschränkt sein. Das Spiel kann nur fortgesetzt werden, wenn die Aufgabe auf dem zweiten Bildschirm gelöst oder abgebrochen wurde. Dadurch wird der Zweitbildschirm direkt in das Spielgeschehen miteinbezogen und dient als ergänzendes Eingabegerät.

Eine Second Screen App muss aber nicht unbedingt immer mit dem Spiel verwendet werden, sondern kann auch diverse Standalone-Möglichkeiten bieten. Zum Beispiel könnten von ihr aus Informationen oder Gegenstände, welche gesammelt wurden, auch dann zugänglich gemacht werden, wenn das Hauptspiel gerade nicht läuft. Eine weitere Möglichkeit sind Minispiele, die ein Anwender auch unterwegs am Second Screen spielen kann, um beispielsweise Spielgeld oder Erfahrungspunkte zu sammeln. Somit sind die Möglichkeiten einer Second Screen App sehr offen.

Damit eine Applikation als Second Screen App bezeichnet werden kann, muss zum Hauptspiel zumindest irgendeine Verbindung bestehen. Das heißt, dass zwischen dem Hauptspiel und der zweiten Applikation Daten geteilt werden müssen, die auch einen relevanten Einfluss auf das Spielgeschehen haben. Eine App die lediglich die Hintergrundgeschichte des Spiels zum Lesen anbietet wäre beispielsweise keine Second Screen Applikation, sondern würde als Merchandising-Artikel gehandhabt werden.

2.3.1 Ursprünge von Second Screen Gaming

In diesem Abschnitt wird auf die Entwicklung von Second Screen Gaming ab 1982 bis heute eingegangen (vgl. [34]). Abbildung 2.3 zeigt die Entwicklung der Spielkonsolen und Handhelds, die eine Second Screen Funktion integrieren in chronologischer Veröffentlichungsreihenfolge.

Nintendo Handhelds und Transfer Kit

Die Ursprünge des Second Screens liegen im Jahr 1982 als *Nintendo* Handhelds veröffentlichte, die zwei Bildschirme besaßen. Das ermöglichte erstmals eine größere Bildschirmfläche für die mobilen Spiele, während die Geräte klein blieben.



(a) 1982: *Game Watch*



(b) 1998: *Sega Dreamcast*



(c) Apr. 1999: *Transfer Pak*



(d) Dez. 1999: *PocketStation*



(e) 2001: *GameCube-GBA Link Cable*



(f) 2004: *Nintendo DS*



(g) 2011: *PS Vita* und *PS3*



(h) 2012: *SmartGlass* für *Xbox 360*

Abbildung 2.3: Spielkonsolen und Handhelds, welche Second Screen Funktionen integrieren [34].

In den 90ern startete *Nintendo* den Verkauf des *Transfer Paks*. Damit konnten erstmals Daten von einem *GameBoy* auf den *Nintendo 64* übertragen werden. Zum Beispiel nutzte *Pokémon Stadium* den *Transfer Pak* weitgehend um gesammelte *Pokémons* von einem *GameBoy* zu übertragen, mit denen anschließend auf der Konsole gespielt werden konnte.

Visual Memory Unit

Die *Visual Memory Unit* (VMU) für den *Sega Dreamcast* war einer der ersten Versuche eine Konsole mit einem ergänzenden zweiten Bildschirm zu etablieren. Während die Hauptfunktion der VMU, die einer Speicherkarte für den Dreamcast darstellte, konnte diese auch als tragbarer Handheld genutzt werden. Im berühmten *Sonic Abenteuer* wurde ein *Tamagotchi* ähnliches Minispiel integriert, das dem Spieler auf der VMU erlaubte sich um eine kleine Kreatur zu kümmern und diese später in das Dreamcast Hauptspiel zu übertragen. Weiters konnte die VMU mit dem Dreamcast Controller verbunden werden um anschließend als erweiternder Bildschirm zu agieren. Diese Funktion wurde von vielen Spielen für die Darstellung von Statistiken genutzt.

PocketStation

Die 1999 exklusiv für den japanischen Markt entwickelte *PocketStation* von *Sony* ist ein Zusatzgerät für die erste *PlayStation*. PlayStation Besitzer konnten für ihre gekauften PlayStation Spiele dazugehörige Minispiele auf die PocketStation herunterladen und diese unterwegs als Zeitvertreib spielen. Obwohl Millionen von Geräten in Japan verkauft wurden, gab es keinen Verkaufsstart in Europa oder Nordamerika.

GameCube - Game Boy Advance Link Cable

Die Besonderheit von Second Screens, das Spielgeschehen aus verschiedenen Perspektiven zu verfolgen und zu verändern, wurde ab Dezember 2001 erlebbar [36]. *Nintendo* veröffentlichte das *GameCube - Game Boy Advance Link Cable* in Japan. Im Februar 2002 folgte die Veröffentlichung in Nordamerika. Mit dieser Verbindung konnten bis zu vier *Game Boy Advance* (GBA) als erweiternde Bildschirme für den GameCube verwendet werden. In Titeln wie *The Legend of Zelda: Four Sword Adventure* und *Final Fantasy: Crystal Chronicles* konnten die Spieler ihr Inventar einsehen und verwenden ohne die Mitspieler durch ein Pausfenster am Hauptbildschirm zu stören. Darüber hinaus war es den Spielern, unter anderem in *Pac-Man Vs.*, möglich auf ihren jeweiligen GBAs komplett unterschiedliche Ausschnitte der Spielweltkarte zu sehen.

Nintendo DS

2004 schockierte *Nintendo* die Spielepresse mit der Veröffentlichung des *Nintendo DS*, welcher wiederum ein Handheld war, der wie in den 1980er Jahren zwei Bildschirme besaß. Doch die hohe Skepsis, die aufkam, legte sich als bekannt wurde, dass es sich beim zweiten Bildschirm um einen Touchscreen handelte, der neue Interaktionsmöglichkeiten bot. Der Nintendo DS besitzt eine große Softwarebibliothek mit zahlreichen Second Screen Funktionen und wurde weltweit über 150 Millionen mal verkauft.

PlayStation Cross-play

Noch bevor die *next generation* Konsole *PlayStation 4* (PS4) 2013 auf den Markt kam wurde der Handheld *PlayStation Vita* (PS Vita) von *Sony* veröffentlicht. Dadurch wurden die Funktionalitäten von *cross-buy* und *cross-play* zwischen dem Handheld und der *PlayStation 3* (PS3) populär. Weiters gab es zuvor wenige Second Screen Anwendungen der PS3 mit der *PlayStation Portable* (PSP), dem Vorgänger der PS Vita. Nach der Veröffentlichung stockte *Sony* das Repertoire der Funktionen zwischen PS3 und ihren Handhelds auf.

Wenn ein Spiel im PSN⁶ mit der Kennzeichnung **cross-buy** gekauft wurde, können Verbraucher das Spiel für jedes geeignete *Sony* Gerät installieren. Zum Beispiel kann *Spelunky* für die PS Vita gekauft werden und eine Kopie für die PS3 kann ohne weitere Kosten heruntergeladen werden. Andererseits können Spieldaten mit **cross-play** zwischen verschiedenen Plattformen kabellos ausgetauscht werden. Das ermöglicht unter anderem den Fortschritt im Spiel *Hotline Miami* von der PS Vita auf die PS3 (und vice versa) über das Internet zu übertragen um das Spiel auf dem jeweils anderen Gerät fortzusetzen.

Xbox 360 SmartGlass

Die erste Nutzung von Smartphones und Tablets als Second Screen wurde 2012 durch *Microsoft* ermöglicht. SmartGlass von *Microsoft* war eine mobile App ausschließlich für die Xbox 360. Sie bot einer Anwendung erweiternde Bildschirmfunktionen und die Möglichkeit die Konsole zu steuern. Nur wenige Spiele, darunter *Halo 4*, nutzten die entwickelten Second Screen Funktionen vollständig aus während *Netflix*⁷ und *Hulu Plus* SmartGlass als reinen Remote Controller verwendeten.

⁶Das *PlayStation Network* (PSN) ist der Onlinemarkt von *Sony* in dem Spiele für die Konsolen und Handhelds erworben und heruntergeladen werden können.

⁷ *Netflix* ist ein online Filmstreaming-Dienst welcher auf zahlreichen Plattformen installiert werden kann. <http://netflix.com>

2.3.2 Integriertes Second Screen Gaming

Die Verbreitung des Begriffes *Second Screen Gaming* entstand mit der Vermarktung der heutigen *next-gen* Konsolen *PlayStation 4* und *Xbox One* 2013, bei denen mit Second Screen Funktionen geworben wurde. Darunter litt teilweise die Vermarktung der *Wii U* von *Nintendo*, welche zuvor im November 2012 mit einem Second Screen Gamepad auf den Markt gebracht worden war.

Nachfolgend wird auf diese Spielkonsolen, welche eine Second Screen Gaming Erweiterung von Haus aus integrieren, näher eingegangen.

Wii U

Wie bereits erwähnt besitzt die *Wii U* von *Nintendo* ein Gamepad mit integriertem Second Screen. Dieser Bildschirm ist zwischen zwei Joysticks eingelassen und wird mit Touchgesten gesteuert. Spielmechaniken, des *Nintendo DS* können auf Grund der selben Hardwarekonstellation übernommen werden. Weiters eröffnet das Gamepad der *Wii U* neue Funktionen im Mehrspielermodus, da die Bildschirme in zwei separaten Geräten zur Verfügung stehen und diese an verschiedene Personen verteilt werden können. Zum Beispiel kann ein Spieler beim Plattformspiel *Super Mario Bros. U*. (s. Abb. 2.4 (a)) seine Kollegen tatkräftig mit dem Gamepad unterstützen. Dabei platziert der Spieler Hilfsplattformen in der Spielwelt, auf welche seine Mitspieler springen können, um nicht in den Abgrund zu stürzen.

Im Falle einer Einzelspieler-Kampagne, wie sie vom Spiel *Deus Ex Human Revolution* angeboten wird, erhält der Spieler eine Weltkartenansicht am Gamepad (s. Abb. 2.4 (b)), auf welche er mit Touchgesten personalisierte Hinweise zeichnen kann. Diese Zeichnungen erscheinen anschließend am Hauptbildschirm in der aktuellen Minikarte. Weiters ist es mit dem Gamepad möglich auf das Inventar zuzugreifen um schneller Gegenstände zu organisieren und zu verwenden. Aktuelle Aufträge und Spielstatistiken erscheinen am Gamepad und helfen gesetzte Ziele effektiver zu erreichen.

Xbox One SmartGlass

Der Funktionsumfang der *Xbox 360 SmartGlass* wurde 2013 von *Microsoft* für die *Xbox One* ausgebaut. Auf der *Xbox One* installierte Spiele werden in einem Menü in der *SmartGlass* App angezeigt und können von dort geöffnet werden. Wird ein Film über die *Xbox One* angesehen, so können zusätzliche Inhalte dafür empfangen werden [44]. Second Screen Erweiterungen für Spiele der *Xbox One* müssen vom Anwender nicht eigens gesucht werden, sondern scheinen automatisch auf, wenn *SmartGlass* mit der Konsole verbunden ist [43].

Der Exklusivtitel *Dead Rising 3* für die *Xbox One* nutzt zahlreiche Second Screen Funktionen der *SmartGlass*. Die *SmartGlass* Anwendung für das



Abbildung 2.4: Das Wii U Gamepad wird als Touchscreen zum Platzieren von Plattformen in *Super Mario Bros. U.* verwendet (a). Als erweiternder Bildschirm zeigt das Gamepad eine Karte aus dem Spiel *Deus Ex Human Revolution* (b).

Spiel simuliert das Smartphone des Protagonisten und kann vom Spieler gesteuert werden. Die Basisfunktionen beinhalten das Anzeigen der Weltkarte und die Auflistung der Aufgaben. Empfängt der Protagonist im Laufe des Spiels einen Telefonanruf wird dieser über das SmartGlass wiedergegeben. Empfangene Text-Nachrichten aus dem Spiel werden in der *Text App* angezeigt.

Wenn die SmartGlass nicht mit der Xbox verbunden ist, kann sie unterwegs als Community-Portal verwendet werden. Dadurch lassen sich Aktivitäten der Freunde abfragen, zum Beispiel welche Spiele sie gerade spielen. Weiters können Nachrichten versendet und empfangen oder Errungenschaften und Fortschrittspunkte eingesehen werden.

PlayStation App

Die *PlayStation App* von Sony ergänzt die *PlayStation 4* mit der Second Screen Gaming Funktionalität. Als Standalone-Anwendung bietet sie dem Anwender den Zugang zur Onlinecommunity und zum PSN. Aktuelle Spiele und DLCs⁸ können vom Verbraucher verfolgt und erworben werden. Um das Smartphone oder Tablet als erweiternden Bildschirm zu nutzen, wird die *PlayStation App* mit der PS4 über ein Bildschirmpasswort verbunden. Beide Geräte müssen sich im selben Netzwerk befinden.

Das DLC *Toy Maker* für *The Playroom* erlaubt dem Spieler eine 2D Grafik am Second Screen mit verschiedenen Farben zu zeichnen. Ist der Künstler mit seinem Werk zufrieden, kann er es mit einer Wischgeste zur PS4 schicken. Diese erstellt, basierend auf der Grafik, ein dreidimensionales Objekt, welches in der Augmented Reality Welt dargestellt wird.

⁸ Im Spielesektor ist DLC die englische Abkürzung für *Downloadable Content* und steht für Software-Erweiterungen die über das Internet gekauft und heruntergeladen werden können.

2.3.3 Standalone Second Screen Apps

Ein aktueller Trend zeigt, dass für fast jeden neu veröffentlichten Triple-A-Titel eine eigenständige Second Screen App entwickelt wird anstatt für die SmartGlass oder PlayStation App Anbindungen zu implementieren.

Second Screen Apps werden über Onlineserver mit den Konsolen- oder PC-Spielen verbunden und synchronisiert. Das ermöglicht unter anderem eine plattformunabhängige Implementierung der Kommunikation und kann eine kosteneffizientere Methode darstellen, da nicht für jede Plattform unterschiedliche Anbindungen entwickelt werden müssen.

Nachfolgend werden ausgewählte Triple-A-Titel besprochen, die eine eigenständige Second Screen App beinhalten.

Assassin's Creed Black Flag

Ubisoft entwickelte eine standalone Second Screen App für *Assassin's Creed Black Flag* [29], den sechsten Teil ihrer *Assassin's-Creed-Reihe*. Die Kommunikation zwischen dem Hauptspiel und der App geschieht über Onlineserver von Ubisoft. Der Benutzer muss dafür einen UPlay⁹ Account verwenden über den die Daten synchronisiert werden können.

Die App bietet unter anderem eine globale und lokale Kartenansicht der Spielwelt mit der aktuellen Spielerposition. Den Spielstatus kann der Benutzer jederzeit unterwegs abrufen, wenn er mit seinem Onlineaccount verbunden ist. Weiters sind der Soundtrack und Zusatzinhalte zur Geschichte, die im Laufe des Spiels frei gespielt werden können, in der App verfügbar. Mit dem Spielen der *Schiffsmission* kann *Cargo*¹⁰ verdient werden. Diese erspielten Punkte werden online gespeichert und automatisch in das Hauptspiel übertragen wenn es mit dem Server verbunden ist.

Watch Dogs

ctOS Mobile ist eine mobile Second Screen App für den aktuellen Hypetitel *Watch Dogs* von Ubisoft [30]. Diese App fokussiert sich auf die Mehrspielerintegration im Spiel. Watch Dogs versetzt den Spieler nach Chicago in eine Open-World in der jederzeit ein Onlinespieler in seine Welt eindringen kann um dessen Smartphone zu hacken.

Die Mehrspielererweiterung für den Second Screen stellt ein Online-Rennen dar, welches zwischen einem Konsolen-/PC-Spieler und einem Tablet- oder Smartphone-Spieler ausgetragen werden kann. Darin fährt der Konsolen- oder PC-Spieler ein Rennen um die Zeit und wird von einem Polizeihaupt-

⁹ UPlay bezeichnet den Onlineaccount eines Spielers bei Ubisoft. Unter anderem wird der Account benötigt um mit der Community zu kommunizieren und online im Mehrspieler-Modus zu spielen.

¹⁰ Cargo ist eine Währung im Spiel *Assassin's Creed Black Flag* mit der Spielgegenstände gekauft werden können.

kommissar daran gehindert, welcher u. a. einen Polizeihubschrauber über das mobile Gerät steuert. Um eine Strecke zu fahren, kann ctOS Mobile verwendet werden um einen Parkour abzustecken, der von dem Rennfahrer bestritten wird. Alternativ kann aus bereits erstellten Strecken eine ausgewählt werden. Anschließend wird mit ctOS Mobile aus der Vogelperspektive ein Polizeihubschrauber gesteuert und Straßenblockaden können errichtet werden um den Rennfahrer bestmöglich zu blockieren. Der Rennfahrer hat die Aufgabe den definierten Parkour in kürzester Zeit abzufahren, während er einerseits dem Helikopter entkommen muss, um nicht abgeschossen zu werden, und andererseits den Blockaden, welche ihm den Weg versperren, ausweichen muss. Verliert der Rennfahrer, weil er es in der vorgegebenen Zeit nicht bis ans Ende des Parkours geschafft hat, bekommt der ctOS Spieler Onlineerfahrungspunkte. Gewinnt er jedoch, bekommt *er* die Onlineerfahrungspunkte gutgeschrieben. Onlineerfahrungspunkte schalten Sonderfähigkeiten des Protagonisten in Watch Dogs frei.

Battlefield 4

Der bekannte *Commander Modus* für den Multiplayermodus der *Battlefield-Serie* wurde in *Battlefield 4* von DICE und EA als Second Screen App realisiert [22]. Im Multiplayermodus spielen zwei Mannschaften in einer Kriegswelt gegeneinander. Dabei bietet der *Commander Modus* eine Übersicht dieser Kriegswelt in der Vogelperspektive und zeigt die Positionen der Soldaten der eigenen Mannschaft an. Über dieses Interface kann der Spieler als Kommandeur seinen Truppen Befehle erteilen und ihnen Hilfe schicken, wenn sie benötigt wird. Die Soldaten spielen währenddessen im First-Person-Shooter auf der Konsole oder am PC.

Kapitel 3

Kategorisierung von Second Screen Gaming

Im vorangegangenen Kapitel wurde auf Apps eingegangen, welche einen Second Screen während des Videospiegels am PC oder auf der Konsole anbieten. Nachfolgend wird versucht das Second Screen Gaming in vier Kategorien einzuordnen und in Beziehung zu setzen. Dabei wird auf die passive und aktive Interaktion mit dem Second Screen eingegangen und der Abhängigkeit vom Hauptspiel gegenüber gestellt.

3.1 Interaktion mit dem Second Screen

Der Second Screen in Videospielen kann passiv oder aktiv vom Spieler verwendet werden. Dazu wird auf die Interaktion des Benutzers mit dem Second Screen eingegangen.

3.1.1 Passiv

Die geringste Interaktion mit dem Second Screen ergibt sich, wenn dieser neben dem Hauptbildschirm stationär positioniert ist und als erweiternde Anzeige genutzt wird. Diese Verwendungsart wird im Weiteren als *passiv* bezeichnet. Auf diese Weise können zum Beispiel eine alternative Kameraperspektive des Spiels oder Informationen zum aktuellen Spielstatus angezeigt werden, welche keine Eingaben des Spielers erfordern. Jedoch muss hierbei beachtet werden, dass der Spieler, wenn er Informationen abrufen möchte, sich vom Hauptspiel abwendet um sich auf den Second Screen konzentrieren zu können. Um möglichst wenig vom Hauptspiel abzulenken sollte die Zeit, welche für das Abrufen der Informationen vom Second Screen benötigt wird, kurz gehalten werden. Dies kann mit einem klaren Interface



Abbildung 3.1: In *ZombiU* wird für das Öffnen von versperrten Türen am Wii U Controller ein Zahlenfeld angezeigt [37].

Design erreicht werden, welches sich an den HCI¹ Gestaltungsprinzipien orientiert. Diese Prinzipien sind vermehrt zu beachten, wenn mehr Interaktion mit dem Second Screen gefordert wird.

3.1.2 Aktiv

Aktive Second Screen Szenarien erfordern eine Interaktion des Benutzers, die abhängig von der Anwendung, das Aufnehmen des Second Screens erfordern. Hierbei tritt der Second Screen in den Mittelpunkt und das Geschehen am Hauptspiel tritt in den Hintergrund.

Angenommen, in einem Geheimagentenspiel kommt ein Spieler zu einer verschlossenen Tür mit einem elektronischen Zahlenschloss. Den Code für das Schloss hat der Spieler bereits auf einem Zettel gefunden und möchte diesen eingeben. Nachdem er seinen Second Screen mit dem Hauptspiel verbunden hat, wird das Zahlenfeld statt am Hauptbildschirm auf seinem Smartphone angezeigt. Die Spielmechanik zum Eingeben des Codes erfordert, dass der Spieler sein Smartphone in die Hand nimmt. Dadurch wird die Interaktion mit dem Second Screen gesteigert und das Gerät muss aktiv genutzt werden.

Spielmechaniken, welche das Aufnehmen des Second Screens erfordern, sind mit Bedacht einzusetzen. Wenn die Eingabe des Codes sehr zeitkritisch ist, wie beispielsweise in *ZombiU* [27] (s. Abb. 3.1), weil eine Horde an Zombies hinter dem Spieler her ist, kann ihn das Wechseln zwischen

¹HCI ist die Abkürzung für den englischen Begriff *Human-Computer Interaction*, zu Deutsch *Mensch-Computer-Interaktion*, dessen Fachbereich sich mit Prinzipien für die benutzergerechte Gestaltung von Computeranwendungen beschäftigt [3].



Abbildung 3.2: Hinweis am Hauptbildschirm für den Spieler, auf das Display des Wii U Gamepads zu sehen [27].

Tastatur/Gamepad und Second Screen überfordern.

In diesem Spiel für die *Wii U* wird das Gamepad sowohl als Eingabegerät als auch als Second Screen verwendet (s. Abschn. 2.3.2). Daher wird kein Wechsel zwischen Geräten benötigt, wodurch bewusst stressige Situationen vermieden werden. Um die Aufmerksamkeit des Spielers auf den Second Screen zu lenken, kann am Hauptbildschirm, wie in Abb. 3.2, ein Hinweis angezeigt werden.

3.2 Second Screens in Abhängigkeit eines Spiels

Eine weitere mögliche Dimension der Kategorisierung ist die Abhängigkeit des Second Screens von einem Hauptspiel. Hierbei wird unterschieden, ob der Second Screen als erweiternder Bildschirm agiert und kontextbezogene Inhalte anzeigt, oder selbstständig, unabhängig von einem Hauptspiel ausgeführt wird.

3.2.1 Abhängig

Ist der Second Screen abhängig von einem Hauptspiel, wird der Austausch an Spieldaten zwischen den Geräten benötigt. Dies erfolgt i. Allg. über eine Lokale-Netzwerk-, Bluetooth- oder Onlineverbindung. Dabei kann unterschieden werden, ob Daten, Befehle oder Ereignisse ausgetauscht werden.

Agiert der Second Screen als erweiternder Bildschirm, können am Second Screen zusätzliche Informationen angezeigt werden.

3.2.2 Unabhängig

Neben der Verwendung des Second Screens als erweiternder Bildschirm, kann dieser auch unabhängig eines Hauptspiels genutzt werden. Das ermöglicht u. a. das Spielen am Second Screen, wenn der PC oder die Konsole nicht zur Verfügung stehen.

3.3 Einordnung von Second Screen Anwendungen

Nach der Definition der Second Screen Gaming Kategorien wird nachfolgend versucht diese anhand einiger Anwendungsfälle zu verdeutlichen. Zudem werden Besonderheiten des Second Screen Gamings aufgezeigt.

3.3.1 Aufgabenliste

Kategorie: *Aktiv, Abhängig* – Eine übersichtliche Aufgabenliste bietet einen schnellen Überblick über aktive Missionen und Ziele, die im Spiel zu erledigen sind. Aufgabenlisten sind besonders hilfreich, wenn es mehrere Nebenmissionen gibt, welche gleichzeitig erledigt werden können. Zudem hilft das Anzeigen des Missionsfortschritts dem Spieler beim Planen seines weiteren Vorgehens.

Um dem Spieler eine anpassbare Aufgabenliste anzubieten, könnte es ermöglicht werden, eigene ToDos der Liste hinzuzufügen und selbstständig abzuhaken. Würden diese ToDos durch einen Text repräsentiert werden, kann dieser auf dem Second Screen am mobilen Gerät durch die virtuelle Tastatur schneller eingegeben werden, als mit einem klassischen Gamepad² bei einem Konsolenspiel.

3.3.2 Spiele für Zwischendurch

Kategorie: *Aktiv, Unabhängig* – Damit sich ein Spieler auch abseits des PCs oder der Konsole mit einem bestimmten Spiel befassen kann, können zum Titel dazugehörige Minispiele für den Second Screen entwickelt werden. Diese kann der Spieler zwischendurch auf seinem mobilen Endgerät spielen und sind meist weniger zeitintensiv als das Hauptspiel. Dabei ist es hilfreich, wenn es für den Benutzer Belohnungen gibt, welche sich auf das Hauptspiel auswirken, um ihn zum Spielen zu motivieren. In [15] werden acht Belohnungsarten aufgezeigt, welche Menschen animieren können ein Spiel zu spielen. Darunter befindet sich die Möglichkeit, der Person Punkte erkämpfen zu lassen, um sich mit anderen KollegInnen messen und vergleichen zu können. Für Rollenspielen eignet es sich gut, den Protagonisten neue Fähigkeiten erlernen zu lassen, wenn am Second Screen gespielt wird. Weiters kann sich

²Ausgenommen ist unter anderem das *Wii U* Gamepad, welches über ein Touchdisplay verfügt, auf welchem eine Touchtastatur angezeigt werden kann.

der Spieler auch freischaltbare Inhalte, wie z.B. neue Levels, Zugang zu speziellen Orten im Hauptspiel oder digitale Gegenstände zum Sammeln, erarbeiten. Diese können wiederum am Second Screen verfügbar gemacht werden (s. Abschnitt 3.3.6).

3.3.3 Online-Mehrspieler

Kategorie: *Aktiv, Unabhängig* – Mit den Second Screen Apps wurde es möglich, sich im Mehrspieler, am Smartphone oder Tablet, mit Videokonsolen- und PC-Spielern zu duellieren. Generell wurde ein Mehrspieler, bei welchem jeder dieselbe Aufgabe zu erledigen hat, zwischen unterschiedlichen Plattformen vermieden. Ein Grund dafür ist, dass es für je eine Spielmechanik je Gerät unterschiedlich gut zu nutzende Eingabemethoden gibt. Zum Beispiel kann ein PC-Spieler mit Maus und Tastatur schneller einen Punkt anvisieren als mit dem Gamepad [8]. Dadurch erhält er einen klaren Vorteil in einem First-Person-Shooter (FPS) auf Gegner zu zielen und zu treffen im Gegensatz zu Konsolenspielern mit einem Gamepad.

Der Ansatz von Second Screen Mehrspielern ist die Trennung zwischen den zur Verfügung gestellten Spielmechaniken am Hauptspiel und jenen am Second Screen. Das ermöglicht das Teilnehmen an derselben Mehrspieler Runde mit unterschiedlichen Geräten. Die Second Screen App wird dabei zur Hauptanwendung und verliert den *Erweiternden-Bildschirm-Charakter* (s. Abschnitt 3.2.1), nachdem nicht mehr mit dem Hauptspiel interagiert wird. Eine solche Trennung findet sich zum Beispiel im *Commander*, der Second Screen Erweiterung für das Spiel *Battlefield 4*, oder in der *ctOS Mobile* App für *Watch Dogs* wieder (s. auch Abschnitt 2.3.3). Ein weiteres Beispiel dafür ist die Steuerung einer Drohne per Tablet im Mehrspieler von *Tom Clancy's The Division*, während Konsolen- und PC-Spieler im FPS spielen [38].

3.3.4 Kartenvisualisierung

Kategorie: *Passiv, Abhängig* – Um dem Spieler eine Übersicht der Spielwelt zu gewähren, gibt es in zahlreichen Spielen eine Weltkarte. Diese ist eine abstrakte Repräsentation der Spielwelt aus der Vogelperspektive. Die Positionen der Spieler, NPCs³, interessanter Orte und Missionsziele können darauf markiert sein [22, 25, 28, 30]. Für eine stetige Übersicht kann während dem Spielen ein Ausschnitt der Gesamtkarte als sogenannte Minikarte am Bildschirm angezeigt werden. Als Mittelpunkt des Ausschnittes dient die Position des Spielers von welcher aus ein gewisser Radius angezeigt wird. Der Radius verändert sich unter anderem in *Watch Dogs* in Abhängigkeit zur Bewegungsgeschwindigkeit des Spielers. Je schneller der Spieler unterwegs

³Non-player character (NPC) ist eine Figur in Videospielen, welche vom Computer gesteuert wird.

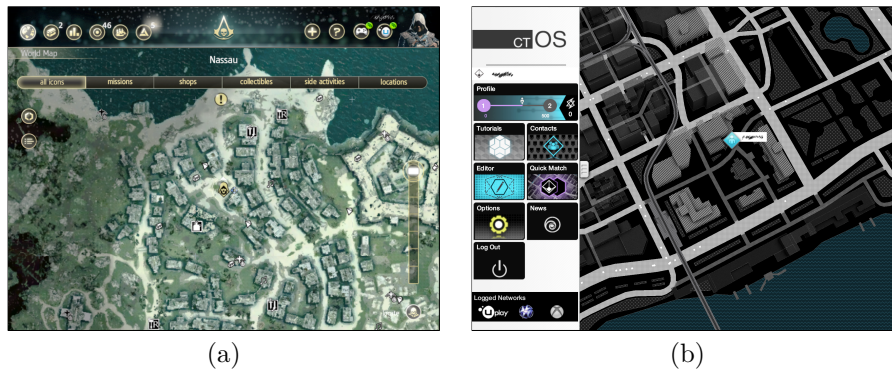


Abbildung 3.3: Kartenansicht von Nassau in der Second Screen App von *Assassin's Creed Black Flag* [24]. Die Zahlen in der links oberen Ecke leuchten abwechselnd auf um den Spieler darauf aufmerksam zu machen (a). Kartenansicht von *The Loop* am Second Screen für *Watch Dogs* (b) [30].

ist, desto größer wird der Ausschnitt der Welt, welcher in der Minikarte dargestellt wird. Eine Gesamtübersicht der Weltkarte nimmt den ganzen Bildschirm ein und verdeckt die Spielwelt. In zahlreichen Spielen kann der Spieler auf der Karte eine Markierung setzen [22, 25, 28, 30]. Darunter gibt es Spiele, welche zum markierten Punkt automatisch einen Weg berechnen [25, 30] und darstellen. Ansonsten wird ausschließlich die Himmelsrichtung zum markierten Punkt in der Minikarte angezeigt [22, 28]. Dies variiert je nach Genre und den gebotenen Spielmechaniken. Zudem ist es wichtig, dass dem Spieler das aktuelle Spielgeschehen vermittelt wird [11], wofür die Kartenvisualisierung unterstützend genutzt werden kann.

Visualisierung am Second Screen

Das Verlagern der Kartenvisualisierung und/oder Minikarte vom Hauptbildschirm auf den Second Screen kann mehrere Vorteile bringen. In [48, 12] wird versucht die Menge an HUD⁴ Elementen auf null zu reduzieren um die Immersion zu verstärken. Dafür bietet sich das Verlagern der Minimap vom Hauptbildschirm auf den Second Screen gut an. Dadurch wird der Bildschirm freier und der Spieler kann mehr von der Spielwelt erfassen. Zum anderen wird der Second Screen für den Spieler zu einem physisch angreifbaren Navigationsgerät der Spielwelt. Durch dieses haptische Interface kann die Immersion in das Videospiel zusätzlich verstärkt werden [52].

Zu beachten ist, dass je nach Größe der Karte und der darauf befindlichen Markierungen die gesamte Welt nicht als Ganzes am Second Screen abgebildet werden kann (s. Abb. 3.3). Der Spieler würde leicht den Über-

⁴Head-up-Display (HUD) ist eine grafische Anzeige vom Spielstatus, welche im Bild immer sichtbar ist und sich somit mit der Spielkamera mitbewegt.

blick verlieren und Spielereignisse könnten übersehen werden, welche den Spieler frustrieren [11]. Folglich wird eine Visualisierung benötigt, wenn außerhalb des gezeigten Kartenausschnittes wesentliche Ereignisse auftreten. Im nachfolgenden Zitat aus [10] wird diese Problematik anhand eines Beispiels aufgezeigt.

Consider a strategy game in which the player has a number of units scattered all over a large map. The map is so large that only a small portion of it can fit on the screen at once. If a group of the player's units happen to be off-screen and are attacked but the player is not made aware of it by the game, the player will become irritated. [...] Why should the player have to guess at such game-critical information?

Darin wird ein Strategiespiel angenommen, indem eine große Anzahl an Einheiten vom Spieler in der Welt verstreut ist. Die Karte umfasst eine Größe, die auf dem Bildschirm nicht als Ganzes angezeigt werden kann. Der Spieler wird verärgert, wenn eine Gruppe seiner Einheiten außerhalb der Anzeige angegriffen wird, aber er vom Spiel darüber nicht informiert wird. Der Angriff könnte auch uninformiert bleiben, wenn am Second Screen statt der Kartenvisualisierung eine andere Anwendung, wie z.B. die Aufgabenliste, angezeigt wird. Hier könnte der Menüpunkt am Second Screen, welcher zur Kartenansicht führt, mit einem blinkenden Element versehen werden, um die Aufmerksamkeit des Benutzers zur Karte zu lenken, wie in Abbildung 3.3 (a) veranschaulicht. Ein Öffnen der Karte über diesen Menüpunkt kann anschließend direkt den Ort des Angriffes zeigen.

3.3.5 Inventar

Kategorie: *Passiv, Abhängig* – Das Inventar, ein wesentlicher Bestandteil in Abenteuerspielen [14], kann mit dem Second Screen leichter zugänglich gemacht werden. So wäre es möglich den Inhalt des Inventars während dem Erkunden der Spielwelt offenzuhalten. Das könnte verhindern, dass der Spieler unnötig versucht in der Gegend Objekte zu finden, welche jedoch keinen Platz im Inventar hätten.

In Abenteuerspielen wird dem Spieler ermöglicht aus Gegenständen des Inventars einen neuen, kombinierten Gegenstand zu fertigen [14]. Dies kann dazu führen, dass sich der Spieler während des Erstellens des neuen Gegenstands für eine längere Zeit im Inventar befindet. Würde nur am Hauptbildschirm gespielt werden, würde das geöffnete Inventarmenü, während des Kombinierens, einen Teil der Spielwelt verdecken. Das bringt Nachteile im Mehrspieler, wenn bei geöffnetem Inventar das Spiel nicht unterbrochen wird⁵: Durch die begrenzte Sicht auf das Spielgeschehen kann der Spieler zu einem leichten Ziel für Gegner werden (vgl. [23]). Eine Auslagerung vom

⁵In Netzwerkspielen, wie zum Beispiel im Mehrspieler von *The Last of Us* kann nicht



Abbildung 3.4: Ansicht eines freigeschalteten Inhalts in der Second Screen App von *Assassin's Creed Black Flag* [24].

Inventar auf den Second Screen würde dem Spieler die Möglichkeit bieten, Gefahren in der Spielwelt am Hauptbildschirm weiterhin wahrnehmen zu können.

Wie in Abschnitt 3.3.4 bereits erwähnt, kann die Immersion mittels haptischer Interaktion gesteigert werden. In diesem Fall könnte dies dadurch erfolgen, dass die Second Screen Applikation einen Rucksack mit digitalen Gegenständen imitiert.

3.3.6 Sammelbare Zusatzinhalte

Kategorie: *Passiv, Unabhängig* – Als Belohnung für das Sammeln von schwer auffindbaren Gegenständen und folglich dem Erkunden der gesamten Spielwelt, können dem Spieler der Zugang zu Audiokommentaren, Making-ofs oder Hintergrundgeschichten ermöglicht werden. Das motiviert den Spieler neben den Hauptmissionen Zeit in das Suchen im Spiel zu investieren, welches die gesamte Spiel länge eines Titels erhöht. Wenn keine Second Screen Anwendung vorhanden ist, werden solche freischaltbaren Inhalte, falls vorhanden, ausschließlich im Hauptspiel dargestellt. Würde der Zugang zu diesen Inhalten am Second Screen gewährt werden, könnte der Spieler fernab vom Spiel beispielsweise Audiokommentaren oder Making-ofs beziehen. Leser von E-Books könnten es bevorzugen freigeschaltete Hintergrundgeschichten am Sofa auf ihren Tablets oder Smartphones zu lesen, statt am Computer oder Fernseher. *Ubisoft* stellt für jeden seiner Titel aus der *Assassin's*

pausiert werden, da ein Anhalten des Spiels bei allen Mitspielern vorgenommen werden müsste, was den Spielfluss hemmen würde.

Creed Reihe zahlreiche Hintergrundinformationen zu Gebäuden, Charakteren und deren Geschichten zur Verfügung [28]. Wird *Assassin's Creed* auf einem Bildschirm gespielt, welcher mehr als zwei Meter entfernt ist, können solche langen Texte teilweise nur erschwert gelesen werden. Das ist mitunter auf die relativ kleine Schriftgröße zurückzuführen. Hierfür würde sich der Second Screen anbieten (s. Abb. 3.4).

3.4 Gegenüberstellung

In diesem Kapitel wurde auf die verschiedenen Nutzungsvarianten von Second Screens eingegangen. Diese können über die Interaktion (*aktiv, passiv*), sowie über ihren Bezug zum Hauptspiel (*abhängig, unabhängig*) in vier verschiedene Gruppen eingeteilt werden. Zur Verdeutlichung werden in Tabelle 3.1 einige Beispiele in dieses Schema eingeordnet.

Tabelle 3.1: Diese Tabelle ordnet einige Second Screen Anwendungsbeispiele den vier aufgestellten Kategorien zu.

	<i>Abhängig</i>	<i>Unabhängig</i>
<i>Aktiv</i>	Aufgabenliste, Lokaler-Mehrspieler	Spiele für Unterwegs, Online-Mehrspieler
<i>Passiv</i>	Kartenvisualisierung, Inventar	Zugang zu gesammelten Zusatzinhalten

Kapitel 4

Eigener Ansatz

Das Hauptziel dieser Arbeit bestand darin, ein Framework zu entwickeln, welches Spieleentwicklern ermöglicht einen Second Screen in ihren Spielen mittels Smartphone oder Tablet zu integrieren. Dieses Kapitel stellt dafür zwei unterschiedliche Konzepte für die Steuerung des Second Screens auf.

4.1 Abgrenzung

Aus Abschnitt 2.3 geht hervor, dass für das Second Screen Gaming bereits zahlreiche Apps implementiert wurden. *Sony* und *Microsoft* haben für ihre jeweilige Konsole eine App konzipiert, welche von Spieleentwicklern als Second Screen für deren Spiele genutzt werden kann (s. Abschn. 2.3.2). W hingegen u. a. *Ubisoft* und *EA* für ihre Triple-A-Spiele je eine eigenständige Second Screen App erstellten (s. Abschn. 2.3.3). Im ersten Fall müssen Second Screen Anwendung sowohl für PlayStation als auch für Xbox SmartGlass entwickelt werden, welche schwer für weitere Plattformen nutzbar sind. Dafür wird der Entwicklungsaufwand der Netzwerkkommunikation zwischen den Geräten seitens der Konsolenhersteller übernommen. Die zweite Herangehensweise erfordert eine vollständige Implementierung einer App je mobiler Plattform. Hierbei muss die Netzwerkkommunikation selbst entwickelt werden. Dafür kann die Codebasis für unterschiedliche Plattformen wiederverwendet werden und ist nicht auf eine Konsole beschränkt.

Das vorliegende Framework versucht die Vorteile aus diesen Herangehensweisen zu kombinieren. Dabei wird der Gedanke verfolgt, eine mobile App, wie die Playstation App oder Xbox SmartGlass zu entwickeln, welche für unterschiedliche Spiele einen Second Screen zur Verfügung stellt. Jedoch wird dabei die Plattform auf welchem das Spiel ausgeführt nicht auf eine Konsole eingeschränkt. Eine gemeinsame Codebasis für den Second Screen wird mit der Verwendung einer Scriptsprache ermöglicht, welche auf unterschiedlichen Plattformen verwendet werden kann.

4.2 Szenarien anhand eines Jump and Run

Anhand der nachfolgenden Szenarien wird versucht den benötigten Funktionsumfang des Frameworks für eine vielseitig einsetzbare Second Screen App darzulegen, welcher anschließend in den Anforderungen an das Framework konkretisiert wird. Je ein Szenario beschäftigt sich mit einer Kategorie aus Kapitel 3.

Für diese Szenarien wird eine Person namens *Jana* angenommen, die ein Tablet und einen Laptop besitzt und regelmäßig Videospiele spielt. Als Spielbeispiel wird ein Jump 'n' Run namens *Elements* verwendet, welches einen Second Screen anbietet. Am Tablet ist die dazugehörige Second Screen App installiert. Beide Geräte befinden sich im selben Netzwerk. *Nico*, eine weitere Person, ist der Freund von Jana. Er ist ebenfalls ein erfahrener Videospieler.

4.2.1 Zusatzinformationen abrufen

Zu Beginn ist der Second Screen noch nicht mit dem Hauptspiel verbunden. Jana möchte im Laufe des Spiels gesammelte Objekte am Second Screen angezeigt bekommen um stets den Überblick über ihr Inventar zu bewahren (s. auch Abschn. 3.3.5). Dafür öffnet Jana die Second Screen App für *Elements* am Tablet und stellt das Gerät neben dem Computerbildschirm auf. Am Computer ist *Elements* bereits geöffnet und Jana navigiert zum Menü, welches für das Verbinden zum Second Screen benötigt wird. Sie verbindet *Elements* mit der App durch Eingabe eines Passworts. In der App ist für Jana nun ersichtlich, dass die Anwendung mit *Elements* verbunden ist. Sie startet das erste Level und beginnt mit der Spielfigur Obst, Federn und Diamanten einzusammeln. Am Second Screen ist das Inventar geöffnet, welches zum Zeitpunkt des Einsammelns eines dieser Gegenstände, diesen sofort mit einer Grafik darstellt. Gleiche Objekte werden gruppiert und mit einer Mengenangabe versehen (s. Abb. 4.1). Weiters wird in der App die Anzahl an gesammelten Münzen angezeigt.

4.2.2 Rätsel lösen

Bonusinhalte und geheimnisvolle Wege können in *Elements* nur mit einem Second Screen freigeschaltet werden. Einen solchen Weg möchte Jana beschreiten, jedoch wird dieser durch einen magischen Stein versperrt. Jana navigiert ihre Spielfigur vor diesen Stein. Dadurch öffnet sich in der verbundenen Second Screen Anwendung über dem angezeigten Inventar eine Farbmatrix (s. Abb. 4.2). Jana wird aufgefordert einen passenden Farbcode einzugeben um den Stein zu entfernen. In der Spielwelt von *Elements* sind Hinweise für die richtige Konstellation zu finden. Nachdem Jana das Rätsel der Farben durchschaut hat, nimmt sie das Tablet mit der Farbmatrix in

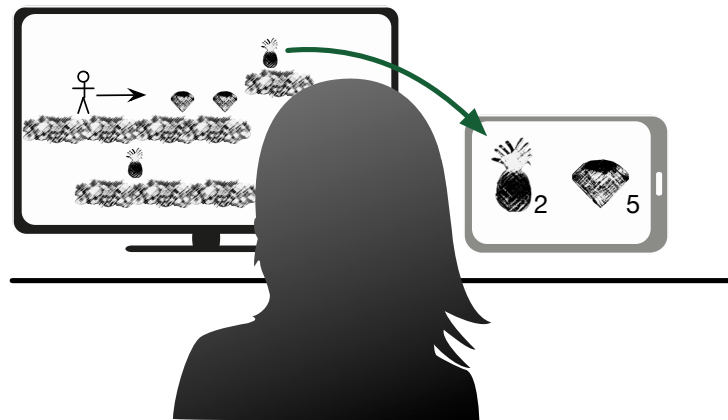


Abbildung 4.1: *Kategorie: Passiv, Abhängig* – Am Second Screen wird ein Inventar von Gegenständen angezeigt, die im Spiel Elements gesammelt wurden. Die Person interagiert mit dem Hauptspiel.

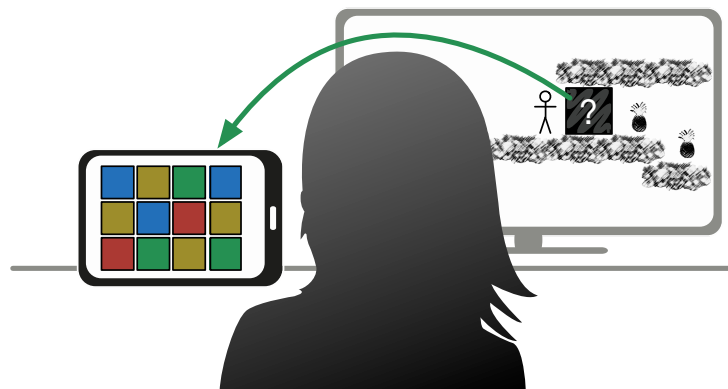


Abbildung 4.2: *Kategorie: Aktiv, Abhängig* – Am Second Screen wird eine Farbmatrix angezeigt, weil die Spielfigur in Elements vor dem magischen Stein steht. Die Person interagiert mit dem Second Screen.

die Hand und gibt den richtigen Code ein (s. auch Abschn. 3.1.2). Der Stein verschwindet, Jana stellt das Tablet ab und die Spielfigur betritt den geheimnisvollen Weg. Jana bekommt für das Lösen des Rätsels Bonusinhalte und Spielerfahrungspunkte.

4.2.3 Unterwegs Spielerfahrungspunkte sammeln

Jana beendet das Spiel und die Second Screen Anwendung. Sie fährt mit der U-Bahn zu Nico um mit ihm Elements im Mehrspieler zu spielen. Dabei hat sie ihr Tablet und ihren Laptop dabei. Während der Fahrt möchte Jana die Fähigkeiten ihrer Spielfigur erweitern, indem sie Erfahrungspunkte mit

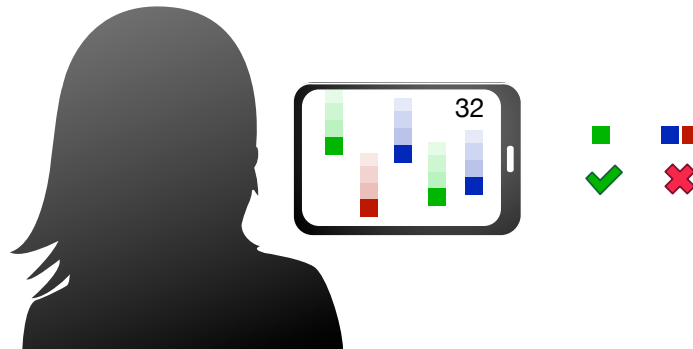


Abbildung 4.3: *Kategorie: Aktiv, Unabhängig* – Während der unabhängigen Nutzung des Second Screens von Elements kann ein Minispiel gestartet werden, in welchem durch Tippen auf herabfallende, grüne Quadrate Punkte erzielt werden. Dabei interagiert die Person mit dem Second Screen.

der Second Screen App sammelt. Dazu öffnet sie die Anwendung am Tablet, welche auch unabhängig vom Hauptspiel genutzt werden kann (s. auch Abschn. 3.3.2). Sie erspielt Erfahrungspunkte in einem integrierten Minispiel. In diesem Spiel muss sie grüne, herabfallende Quadrate einsammeln und darf anders eingefärbte Objekte nicht berühren (s. Abb. 4.3). Die Anzahl an richtig eingesammelten Quadraten wird in einer Ecke dargestellt und am Ende des Spiels als Erfahrungspunkte gutgeschrieben. Das Einsammeln einer falschen Farbe führt sofort zu einem *Game over*. Dieses Spielprinzip ist sehr einfach gehalten und ermöglicht eine kurze Spieldauer.

4.2.4 Ein Making-of ansehen

Bevor die Fahrt mit der U-Bahn zu Ende ist, ergreift Jana noch die Möglichkeit einen freigeschalteten Inhalt von Elements anzusehen. Dafür kehrt sie vom Minispiel, mit welchem sie neue Erfahrungspunkte für ihre Spielfigur erkämpft hat, zum Hauptmenü des Second Screens zurück. Anschließend navigiert sie zu einer Liste von bereits freigeschalteten Inhalten. Dort wählt sie ein Making-of Video aus, welches nach einem kurzen Ladezyklus startet (s. Abb. 4.4).

4.2.5 In einem Mehrspieler mitspielen

Jana ist bei Nico angekommen und verbindet wieder die Second Screen App mit Elements. Dabei werden ihre erzielte Punkte aus dem Minispiel in das Hauptspiel übertragen und neue Fähigkeiten ihrer Spielfigur werden freigeschaltet. Die ersten Mehrspieler-Runden werden zwischen Jana und Nico über ihre Laptops ausgetragen. Dabei helfen Jana die neuen Fähigkeiten, sodass sie öfter als Nico gewinnt.



Abbildung 4.4: *Kategorie: Passiv, Unabhängig* – Die Person sieht sich am Second Screen ein freigeschaltetes Making-of Video eines Spiels an.

Abschließend entschließt sich Jana dazu einen Mehrspieler mit ihrem Tablet gegen Nico zu spielen (s. auch Abschn. 3.3.3). Dafür verbindet Nico sein laufendes Spiel mit Janas Second Screen App. Anschließend eröffnet er eine Mehrspieler-Runde an welcher Second Screen Spieler teilnehmen können. In der App von Jana erscheint daraufhin ein Hinweis, dass Nico eine Runde eröffnet hat, an welcher sie teilnehmen kann. Jana tritt dem Spiel bei. Auf dem Laptop von Nico startet die gewohnte Jump 'n' Run Welt von Elements in der Nico die Spielfigur steuern kann. Am Second Screen wird eine vereinfachte Darstellung der Welt angezeigt. Jana besitzt keine eigene Spielfigur, jedoch sieht sie die Position von Nico und hat im Gegensatz zu ihm die Möglichkeit den Ausschnitt der Welt in Größe und Position zu verändern. Während Nico versucht mit seiner Figur springend und laufend ans Ziel zu gelangen, hat Jana die Aufgabe seine Welt zu manipulieren. Sie kann Hindernisse über ihr Tablet mit Touchgesten aktivieren (s. Abb. 4.5). Wird Nicos Spielfigur durch ein Hindernis von Jana aufgehalten erhält sie dafür Punkte.

Jana berührt unbeabsichtigt den Standby-Knopf ihres Tablets. Der Second Screen wird daraufhin schwarz. Nicos Spiel erkennt, dass die Verbindung zum Second Screen unterbrochen ist und pausiert. Nachdem Jana die Standby-Taste erneut betätigt hat, wird am Tablet der Second Screen wieder angezeigt. Daraufhin wird bei Nico ein Countdown heruntergezählt und eine Information angezeigt. Der Text besagt, dass Elements mit dem Second Screen wieder automatisch verbunden wurde und das Spiel nach Ablauf dieses Countdowns fortgesetzt wird.

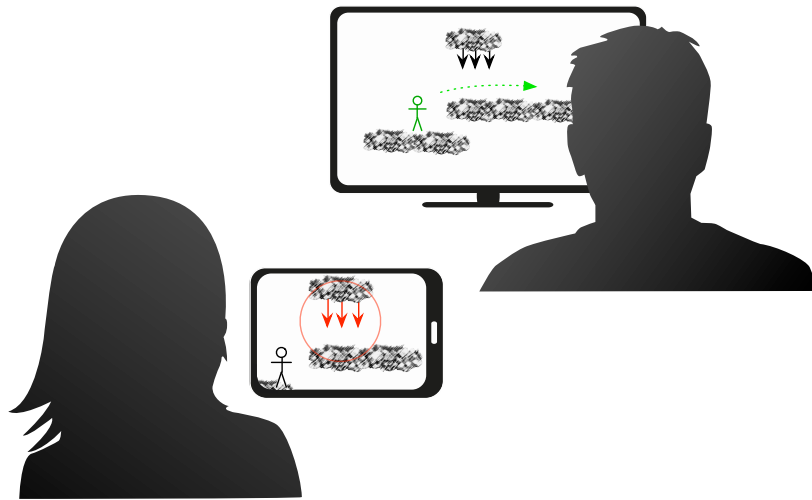


Abbildung 4.5: *Kategorie: Mehrspieler* – Eine Person steuert die Spielfigur eines Jump 'n' Run am Bildschirm während eine zweite Person am verbundenen Second Screen versucht die Spielfigur durch Auslösen von Hindernissen aufzuhalten.

4.2.6 Zusammenfassung

Die Szenarien haben die vier Kategorien aus Kapitel 3 der Verwendung von Second Screens anhand eines Jump 'n' Runs verdeutlicht. Dabei sind drei Sonderfälle aufgetreten, welche in der Umsetzung von Second Screen Gaming zu beachten sind.

Passwortschutz: Für die Verbindung des Spiels mit der Second Screen App wurde ein Passwort benötigt. Dadurch kann verhindert werden, dass Unbefugte sich mit dem Second Screen anderer Personen im selben Netzwerk verbinden können.

Erfahrungspunkte übertragen: Als Jana ihre Second Screen App nach dem Minispiel wieder mit Elements verbunden hat, wurden ihre gesammelten Erfahrungspunkte übertragen. Hätte sie die Verbindung vor dem Match gegen Nico nicht aufgebaut, hätte ihre Spielfigur nicht über mehr Erfahrungspunkte verfügt.

Verbindungsverlust: Die Verbindung der Geräte wurde unterbrochen, als am Tablet der Standbymodus aktiviert wurde. Nachdem das Tablet wieder aktiv wurde, konnten sich beide Geräte automatisch verbinden.

Daraus ergeben sich nachfolgende Anforderungen unter der Berücksichtigung der oben erwähnten Sonderfälle.

4.3 Anforderungen

Wie aus Abschnitt 4.1 hervorgeht gilt es ein Framework zu entwickeln, welches es einem Spieleentwickler ermöglicht einen Second Screen in seinem Hauptspiel anzubieten. Dabei wurde besonders auf die Plattformunabhängigkeit und auf eine zeiteffiziente Integrationsmöglichkeit des Tools in ein konkretes Spiel Wert gelegt.

4.3.1 Funktionale Anforderungen

Unter Berücksichtigung der vorangegangenen Wertlegung und der Szenarien aus Abschnitt 4.2 wurden folgende funktionale Anforderungen aufgestellt:

Universelle Second Screen App: Es sollte eine von sich aus allein stehende App entwickelt werden, welche für jedes Hauptspiel als Second Screen fungieren kann. Diese sollte unabhängig von der im Hauptspiel verwendeten Game-Engine und dem Genre des Spiels sein. Dies verhindert, dass in weiterer Folge für jedes Spiel pro Plattform eine eigene vollständige App implementiert werden muss.

App kann vom Hauptspiel gesteuert werden: Ein Hauptspiel sollte die Anzeige- und Interaktionsmöglichkeiten der App individuell beeinflussen können. Dadurch kann ein Spiel die App, wie im Szenario von Abschnitt 4.2.1, als erweiternden Bildschirm oder als Eingabegerät, wie in Abschnitt 4.2.2, nutzen.

Unabhängig verwendbar: Wie aus den Szenarien in Abschnitt 4.2.3 und 4.2.4 hervorgeht, sollte es ohne eine bestehende Verbindung zum Hauptspiel möglich sein eine Second Screen Anwendung zu nutzen. Nachdem es galt, eine universelle App nicht konkretisiert für ein Spiel zu entwickeln, muss dem Hauptspiel gewährt werden, Individualisierungen auf dem Second Screen vorzunehmen, welche gespeichert bleiben. Dadurch ist ein späteres Verwenden des individualisierten Second Screens ohne aktiver Steuerung durch das Hauptspiel möglich.

4.3.2 Nicht funktionale Anforderungen

Zusätzlich wurden nicht funktionale Anforderungen definiert, welche zu erfüllen waren:

Plattformunabhängigkeit: Das Framework der Second Screen App sollte plattformunabhängig gestaltet werden. Das heißt, dass Spieleproduzenten in der Entwicklung der Kommunikation und Gestaltung ihres Second Screens nicht auf eine konkrete Plattform eingeschränkt werden.

Sicherheit: Es sollte gewährleistet werden, dass der Second Screen nicht unbefugt gesteuert werden kann und das Hauptspiel keinen unerlaubten Zugriff auf persönliche Daten am Second Screen Gerät erhält.

Entwicklerfreiheit: Es sollten dem Entwickler nur wenige Grenzen gesetzt werden, welche ihn in der Individualisierung der App einschränken. Dadurch sollte das Kreieren von neuen Spielmechaniken und Anwendungsarten für den Second Screen ermöglicht werden, auf welche im Vorfeld nicht eingegangen werden kann.

Entwicklerfreundlichkeit: Die Zeit des Einlernens des Frameworks sollte möglichst kurz gehalten werden, d. h. es sollte auf bekannte Technologien und Schnittstellen zurückgegriffen werden um die finanziellen, zeitlichen oder mentalen Hürden zu minimieren.

Benutzerfreundlichkeit: Neben der Entwicklerfreundlichkeit sollte der Second Screen auch benutzerfreundlich von Endkunden verwendet werden können und für diese attraktiv gestaltet sein, soweit dies von der zu entwickelnden App beeinflussbar ist. Dies sollte die Verbreitung der App fördern, wodurch der Anreiz der Entwickler erhöht wird, das Framework zu verwenden. Die Bedienbarkeit und Benutzerfreundlichkeit eines konkreten Second Screen Inhalts muss von den Entwicklern selbst gewährleistet werden.

4.4 Entwurf des Frameworks

Um den definierten Anforderungen unter Berücksichtigung des Szenarios nachzukommen, wurde entschieden, neben der Implementierung der Second Screen App zusätzlich ein SDK für das Hauptspiel mitzuentwickeln. Dieses SDK soll für den Entwickler die Kommunikation mit der Second Screen App abstrahieren und vereinfachen (s. Abb. 4.6). Während der Entwicklung des Frameworks entstanden zwei unterschiedliche Ansätze, auf welche Weise die Second Screen App gesteuert werden kann. Der erste Ansatz, das *Slave-Konzept*, beschreibt eine Steuerung mit vordefinierten Befehlen, während der finale Ansatz, das *Subapp-Konzept*, erlaubt am Second Screen sogenannte *Subapps* zu installieren. In Kapitel 5 wird auf Kommunikation der finalen Lösung eingegangen. Dieses beinhaltet auch den Aufbau der Kommunikation, welcher in beiden Ansätzen äquivalent ist und daher in diesem Kapitel nicht weiter behandelt wird.

4.4.1 Slave-Konzept

Die erste Überlegung war, die universelle Second Screen App mit vordefinierten Befehlen zu steuern, während Ereignisse, welche am Second Screen



Abbildung 4.6: *Entwurf des Frameworks* – Das SDK ist in einem Hauptspiel integriert und kommuniziert mit der Second Screen App über eine drahtlose Verbindung. Dabei wird die App auf einem mobilen Endgerät ausgeführt.

auftreten, an das Hauptspiel zurückgesendet werden. Nachdem das Spiel im Bezug auf die Entwicklung des Frameworks unbekannt ist, können keine Befehle für konkretisierte Second Screen Anwendungen implementiert werden.

Der daraus resultierende Lösungsansatz war eine grafische Benutzeroberfläche in der mobilen App zu verwenden, welche mit Befehlen manipuliert werden kann. Abbildung 4.7 zeigt die dafür entwickelte Architektur. Dabei wird die gesamte Logik des Second Screens vom Hauptspiel berechnet.

Befehle

Die Logik am Hauptspiel kann über das SDK folgende Befehle zum Second Screen senden:

Seite darstellen: Das Initialisieren der Benutzeroberfläche erfolgt mit dem ersten Empfangen von **Seite darstellen**. Dabei wird die Ausgangsstruktur der anzuzeigenden Objekte konfiguriert. Jedes Objekt besitzt Informationen über dessen Aussehen und kann einer eindeutige ID zugewiesen werden.

Element hinzufügen: Zu einer aktiven Seite wird ein neues, anzuzeigendes Objekt hinzugefügt indem dessen Aussehen und ID angegeben wird.

Element entfernen: Ein Objekt wird mit Hilfe seiner ID aus der aktuell angezeigten Seite entfernt. Dabei ist es irrelevant, ob das Objekt durch den Befehl **Element hinzufügen** oder initial durch **Seite darstellen** erstellt wurde.

Element verändern: Ein vorhandenes Objekt wird in seinem Aussehen verändert.

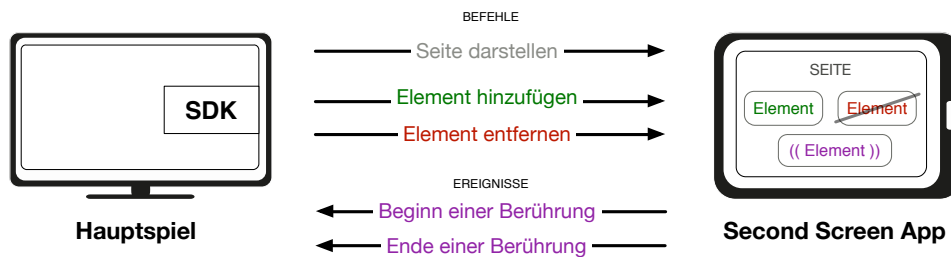


Abbildung 4.7: *Slave-Konzept* – Das SDK steuert mit vordefinierten Befehlen eine grafische Benutzeroberfläche der Second Screen App während Interaktionen mit dieser Benutzeroberfläche zurückgemeldet werden.

Ereignisse

Damit die Interaktionen der Benutzer mit der Benutzeroberfläche am Hauptspiel verarbeitet werden können, werden folgende Ereignisse benötigt:

Beginn einer Berührung: Wird mit einem Finger am Second Screen ein Objekt berührt, wird ein Ereignis ausgelöst, welches die ID dieses Objekts an das Hauptspiel sendet.

Ende einer Berührung: Um festzustellen, dass ein Objekt von einem Finger nicht mehr berührt wird, wird das Ende jeder Berührung mitgeteilt. Würde das Hauptspiel ein Objekt als einen Button ansehen, könnte anhand der Ereignisse festgestellt werden, dass dieser Button betätigt wurde.

Bewegung der Berührung: Wird der Finger während des Berührens des Touchdisplays bewegt, wird dessen Bildschirmposition zurückgegeben. Das ermöglicht z. B. einem Hauptspiel das unter dem Finger liegende Objekt mit dem Befehl **Element verändern** mitzubewegen.

Fazit

Die Funktionsweise des vorgestellten Konzepts wurde in dem Prototyp von Abschnitt 7.2 umgesetzt. Dabei stellte sich heraus, dass mit diesem Konzept universelle Anzeigen erstellt werden können. Jedoch muss das Hauptspiel stets mit dem Second Screen verbunden sein um Anweisungen erteilen zu können. Eine Nutzung der App ohne Verbindung zum Hauptspiel ist daher nicht möglich. Weiters gab es keine Möglichkeit Bilder am Second Screen darzustellen, nachdem benötigte Ressourcen nicht im Vorfeld in der App zur Verfügung standen. Eine weitere Schwachstelle der Lösung war die Limitierung auf vordefinierte Befehle und Ereignisse, welche im Voraus implementiert werden mussten.

Die Erweiterung der Funktionsweise der Second Screen App ist nur bedingt möglich. Die gesamte App muss aktualisiert und anschließend von Endkunden heruntergeladen werden. Dadurch ist schwer abzuschätzen welche Version der App ein Endkunde installiert hat. Ein Hauptspiel müsste unterschiedlichste Versionen der App unterstützen, oder dem Endkunden eine Version aufzwingen. Deswegen wurde nach einer neuen Lösung gesucht.

4.4.2 Subapp-Konzept

Für den finalen Ansatz wurde zu Beginn auf die unabhängige Nutzung der Second Screen App in Bezug auf das Hauptspiel eingegangen. Erst im zweiten Schritt wurden Möglichkeiten der direkten Steuerung entwickelt. Zusätzlich wurde das Ziel des Frameworks aus der Sicht des Spielers wie folgt formuliert:

Ein Kernelement in der Entwicklung des Frameworks ist, dass Spieler die mobile App als individualisierten Second Screen für ein beliebiges Hauptspiel, auch ohne dessen Steuerung, verwenden können, ungeachtet dessen, welches Spiel zuletzt die App gesteuert hat.

Grundlegende Idee

Das finale Konzept baut darauf auf, dass eine mobile App entwickelt wird, in welcher für jedes Hauptspiel eine eigenständige *Second Screen Logik* installiert werden kann. Diese Logik kommuniziert über Events mit dem dazugehörigen Hauptspiel und steuert die individuelle Darstellung der mobilen App und ermöglicht auf Eingaben des Benutzers zu reagieren. Für Konfigurationen der Logik, der Darstellung von Bildern und das Abspielen von Audio werden Ressourcen benötigt, welche im Zuge der Installation in der App gespeichert werden. Die Logik und dessen benötigte Ressourcen werden im Weiteren unter dem technischen Begriff *Subapp* zusammengefasst und stellen eine in sich abgeschlossene Applikation innerhalb der mobilen App dar, auf die in Abschnitt 4.5 näher eingegangen wird.

Daraus folgt, dass eine Subapp in die mobile App installiert werden kann und diese um einen neuen Second Screen erweitert. Die Anzahl der gleichzeitig installierten Subapps ist seitens der mobilen App nicht begrenzt, jedoch ist die maximale Anzahl von der Speicherauslastung der installierten Subapps und der vorhandenen Speicherkapazität des Geräts abhängig. Eine Subapp kann nach ihrer Installation vom Spieler ausgeführt, verwendet, wieder beendet und deinstalliert werden und agiert unabhängig vom Hauptspiel. Dabei sollen Endkunden die Installation der Subapp nur vornehmen können, wenn sie das dazugehörige Hauptspiel besitzen, nachdem der Sinn der Nutzung einer Subapp einzig in der Abhängigkeit zu einem Hauptspiel besteht – Endkunden könnten nicht den vollen Funktionsumfang der Subapp nutzen und wären eventuell von den gebotenen Inhalten ohne Bezug zum

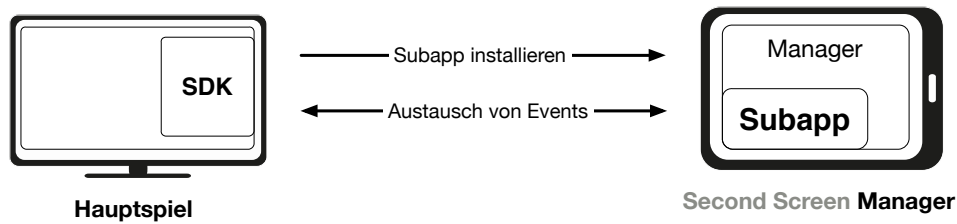


Abbildung 4.8: *Subapp-Konzept* – Das Hauptspiel installiert mithilfe des SDKs eine Subapp in den Second Screen Manager. Anschließend können Events zwischen Hauptspiel und Subapp ausgetauscht werden.

Hauptspiel irritiert. Entwickelt werden die Subapps von den Spieleherstellern, welche mithilfe dieses Frameworks einen Second Screen in ihr Hauptspiel integrieren möchten. Im Weiteren wird für die mobile App die Bezeichnung *Second Screen Manager* verwendet. Abbildung 4.8 veranschaulicht das Zusammenspiel zwischen Manager und Hauptspiel.

Second Screen Manager (SSM)

Aus der vorangegangenen Idee folgt, dass eine Second Screen mit einer Subapp ermöglicht wird und von einer grafischen Steuerung der mobilen App abgesehen werden kann. Daher werden die vordefinierten Befehle zur Anzeigenänderung des Second Screens aus dem Slave-Konzept (s. Abschn. 4.4.1) nicht weiter in Betracht gezogen. Dennoch soll eine Kommunikation mit der mobilen App ermöglicht werden. Dabei ist wesentlich, dass ein Hauptspiel nur mit seiner eigenen Subapp kommunizieren und Daten austauschen kann. Das verhindert eine Fremdsteuerung der Subapps und gewährleistet die Sicherheit der Spieldaten und Ressourcen. Die Kommunikation wird ermöglicht, indem Events zwischen Manager und Subapp ausgetauscht werden, welche von Spielehersteller selbst definiert wurden (s. auch Abschn. 5.3).

Weiters folgt aus der Idee, dass mehrere Subapps zur selben Zeit installiert sein können und diese aufgrund ihrer benötigten Ressourcen (s. Abschn. 4.5.4) Speicherkapazität am Gerät verbrauchen. Für die Endkunden soll daher in der App ersichtlich sein, welche Subapps installiert sind und wie viel Speicherplatz diese benötigen. Es können Subapps einzeln deinstalliert (entfernt) werden, damit wird vermieden, dass die mobile App als Ganzes von den Benutzern gelöscht werden muss, wenn diese Speicher für andere Anwendungen benötigen. Die Anzeige des benötigten Speichervolumens einer Subapp hilft ihnen dabei die Entscheidung zu treffen, welche Subapps sie deinstallieren könnten um eine gewünschte Speicherkapazität wieder zu erhalten. Um während der Deinstallation Kommunikationsfehler und Spieldatenverlust vorzubeugen, darf die Subapp während der Deinstallation nicht ausgeführt werden oder mit dem Hauptspiel verbunden sein.

Subapp-Menü: Das Menü im Second Screen Manager stellt alle installierten Subapps mit deren Icons dar. Durch das Klicken auf dessen Icon kann eine Subapp gestartet werden. Eine Subapp startet nicht automatisch, wenn sich ein Hauptspiel mit dem Second Screen Manager verbindet. Das gibt den Benutzern mehr Kontrolle über den Second Screen Manager. Wird jedoch eine gestartete Subapp ohne bewusstes Vorgehen von Benutzern aus technischen Gründen geschlossen, wird versucht die Subapp erneut zu starten und gegebenenfalls mit dem Hauptspiel neu zu verbinden. Dies kann auftreten, wenn der Manager von der mobilen Plattform geschlossen wird weil das Gerät den Standby-Modus aktiviert, Endkunden in das Hauptmenü ihres mobilen Gerätes wechseln oder eine andere mobile App gestartet wird. Kehrt das mobile Gerät zum Second Screen Manager zurück könnte die Subapp nicht mehr ausgeführt werden, da die App vom Gerät neu initialisiert wurde.

Subapps, welche mit dem Hauptspiel verbunden sind werden aus dem Menü hervorgehoben. Diese visuelle Repräsentation mittels Icons und einer Verbindungsanzeige soll Spielern ermöglichen die gewünschte Subapp schneller zu finden und starten.

4.4.3 Begriffsübersicht

Die nachfolgenden Begriffe werden anhand ihrer Verwendung im Rahmen des entwickelten Frameworks definiert. Daher ist zu beachten, dass unter anderem die Bezeichnung *Second Screen*, wie aus Kapitel 2 hervorgeht, vielfältig benutzt werden kann und die Bedeutung abhängig vom Kontext ist.

Second Screen: Erweiterung des Videospielerlebnisses mithilfe eines zusätzlichen Ein- und Ausgabegeräts.

Hauptspiel: Ein Spiel, welches einen Second Screen zur Verfügung stellt, ist dessen Hauptspiel.

Second Screen Manager: Der Second Screen Manager (SSM) ist die mobile Applikation, in welche Subapps installiert werden können.

Subapp: Die Subapp ist die technische Umsetzung eines Second Screens. Diese wird von Spieleherstellern entwickelt und im Second Screen Manager installiert.

Ressourcenbündel: Das Ressourcenbündel bezeichnet die Zusammenfassung aller von einer Subapp benötigten Ressourcen einschließlich deren Logik.

SDK: Die universelle Bezeichnung *SDK* (Software Development Kit) bezieht sich auf die entwickelte Bibliothek, welche von Spieleherstellern in das Hauptspiel integriert werden kann um die Kommunikation mit dem Second Screen Manager und der installierten Subapp zu vereinfachen.

4.5 Subapp

Wie aus dem finalen Konzept hervorgeht, stellt eine Subapp einen Second Screen für ein Hauptspiel dar und beinhaltet Informationen für Logik, Darstellung und Ressourcen. Abbildung 4.9 bietet einen Überblick über die Architektur. Implementiert wird eine Subapp von Spieleentwicklern, welche den Second Screen Manager verwenden möchten. Nachfolgend wird auf die Struktur und Entwicklung einer Subapp näher eingegangen.

4.5.1 Logik

Das Entwickeln von Spielen benötigt Zeit und Geld. Möchten Spieleentwickler das Second Screen Framework in ihr Spiel einbinden, entstehen dafür zusätzliche Entwicklungskosten. Vor allem ist dabei zu beachten, dass die Erstellung der Subapps zeitaufwendig ist, da Programmierer erst die eigenständige Logik entwickeln müssen. Um diese Hürde der Integration des Frameworks zu minimieren, sollte die Subapp nur einmalig entwickelt werden müssen und kann auf jedem Second Screen Manager, unabhängig auf welcher mobilen Plattform dieser ausgeführt wird, verwendet werden. D. h. die Spieleproduzenten müssen z. B. nicht jeweils eine eigene Subapp für iOS, Android und Windows Phone implementieren, um auf diesen Plattformen einen Second Screen anzubieten.

Dies wird ermöglicht indem die Logik der Subapp mit einer interpretierten Programmiersprache entwickelt werden kann. Eine mögliche plattformunabhängige Programmierung in C++ wurde ausgeschlossen, da für jede Plattform der C++ Code individuell kompiliert werden müsste. Das hätte den Aufwand der Programmierung gleichermaßen reduziert, jedoch wäre dadurch für jede Plattform wiederum eine eigene Subapp notwendig gewesen. Ein weiterer Vorteil der Skriptsprache gegenüber C++ ergibt sich, wenn der Second Screen Manager für eine neue mobile Plattform implementiert wird: Bereits entwickelte Subapps können auf der neuen App ohne Anpassungen der Logik installiert und verwendet werden.

4.5.2 Szene

Eine Subapp unterteilt sich in verschiedene Szenen, von welchen zur Laufzeit jeweils eine aktiv sein kann. Dies ermöglicht Entwicklern für eine Offline-Nutzung eine andere Logik und Darstellung beim Starten der Subapp zu

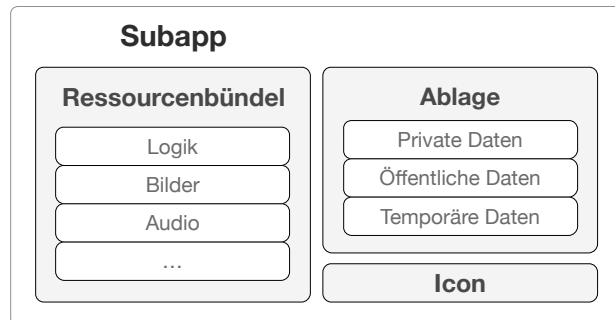


Abbildung 4.9: Die Subapp teilt sich in drei Basiselemente. Im Gegensatz zu den Daten der Ablage können das Ressourcenbündel und das Icon nach der Installation nicht verändert werden.

verwenden, als wenn die Subapp vom Hauptspiel gesteuert wird. Eine Szene beinhaltet folgende Elemente:

Szenen-ID: Jede Szene in der Subapp hat eine eindeutige Identifikationsbezeichnung, mit der die Szene gestartet werden kann. Wird bereits eine Szene ausgeführt, kann mit der ID zu einer neuen Szene gewechselt werden.

Hauptskript: Das Hauptskript wird nach dem Laden der benötigten Ressourcen ausgeführt. Das Wechseln zwischen den Szenen wird dem Hauptskript der zuletzt aktiven und der neuen Szene mitgeteilt. Während eine Szene aktiv ist wird die Logik durch kontinuierliches Aufrufen der **update**-Funktion aktualisiert. Folgende Schnittstellen müssen von Entwicklern im Hauptskript implementiert werden:

setup() wird einmalig, wenn die Szene aktiviert wird, aufgerufen.

update() wird kontinuierlich nach **setup** aufgerufen.

finish() wird einmalig beim Verlassen der Szene, ausgeführt.

Ressourcenliste: Die Liste der Ressourcen-IDs definiert Ressourcen die vor dem Start der Szene geladen werden sollen. Während des Ladens wird eine vollflächige Grafik dargestellt, welche den Spielern mitteilt, dass der Second Screen Manager bis zur Fertigstellung des Ladevorgangs nicht benutzt werden kann.

Szenengraphen: Am Szenengraphen können Grafikelemente hinzugefügt werden, welche am Bildschirm dargestellt werden sollen.

application.script

Wenn eine Subapp gestartet wird, entscheidet das Konfigurationsskript der Subapp (`application.script`), welche Szene zu Beginn geladen und angezeigt werden soll. Dazu wird in diesem Skript die Schnittstelle

```
configure(r : Registry, p : PrimaryScreen) : Config
```

mit zwei vom Second Screen Manager bereitgestellten Objekten aufgerufen. Diese Schnittstelle muss von Subappentwicklern im Konfigurationsskript implementiert werden, bei der ein `Config`-Objekt zurückzugeben ist.

Config: Im `Config`-Objekt werden die Szenen-ID und die zu verwendende Ausrichtung des Gerätes spezifiziert. Die Szenen-ID entscheidet über die Szene, welche geladen werden soll. Mobile Geräte können im Hochformat (Portrait) oder Querformat (Landscape) bedient werden. Dafür richtet sich der Bildschirm anhand der Orientierung des Geräts aus. Nicht jede Ausrichtung kann für eine App-Anwendung sinnvoll sein. Zum Beispiel ist in einem nach rechts laufenden *Jump 'n' Run* von der zukünftigen Spielwelt im Querformat mehr zu sehen, als wenn die Sicht durch das Hochformat eingeschränkt wird. Hingegen wird ein *Tower Jump*, bei welchem die Spielrichtung i. Allg. nach oben verläuft, im Hochformat dargestellt, um mehr weiter oben liegende Plattformen, auf welche man als nächstes Springen könnte, anzuzeigen.

Ein weiterer Aspekt den es beim Anbieten von zwei verschiedenen Ausrichtungen zu beachten gilt ist, dass die Benutzeroberfläche für Quer- und Hochformat designt werden muss. Um in diesem Fall sowohl diesen Entwicklungsaufwand einzuschränken als auch Game Designern die Möglichkeit zu geben ihren Spielern ein vordefiniertes Spielerlebnis zu bieten, kann in der `Config` spezifiziert werden, welche Ausrichtungen die Subapp unterstützt.

Registry: Die `Registry` besteht aus einer Liste von Schlüsselwort-Wert-Paaren, wobei je einem Schlüsselwort ein Wert zugeordnet ist. Jede Subapp verfügt über eine eigene Liste und kann auf die `Registry` anderer Subapps nicht zugreifen.

Beenden Spieler die Subapp, erhalten Entwickler während des Schließvorgangs die Möglichkeit in der `Registry` den Zustand der Subapp zu speichern. Damit kann beim erneuten Starten der zuletzt gespeicherte Zustand wiederhergestellt werden, indem in `configure` die `Registry` ausgelesen wird und anhand dieser eine Szenen-ID definiert wird.

PrimaryScreen: Das Objekt `PrimaryScreen` (Hauptbildschirm) stellt das Hauptspiel für die Subapp dar. Darin sind für die Kommunikation relevante Informationen enthalten; u. a. ob das Hauptspiel verbunden ist und welche Version das Hauptspiel hat. Aufgrund dieser Daten kann auch die zu

startende Szene ausgewählt werden. Zum Beispiel könnte eine Szene implementiert sein, welche nur in Verbindung mit dem Hauptspiel relevant ist (s. Abschn. 3.2.1) und eine Minispielszene, welche unabhängig ausgeführt werden kann (s. Abschn. 3.2.2).

4.5.3 Versionsverwaltung

Ein Hauptspiel kann auf unterschiedlichen Geräten verschiedene Versionen aufweisen. Spiele werden von Spieleherstellern aktualisiert, um Spielinhalte zu erweitern oder neu entdeckte Fehler im Spiel auszubessern. Dabei kann es vorkommen, dass Programmcode, welcher für die Kommunikation mit der Subapp zuständig ist, verändert werden muss und damit verbunden eine neue Logik der Subapp benötigt werden könnte. Daher kann eine Subapp anhand einer eigenen Versionsnummer identifiziert werden. Ist die gewünschte Version nicht vorhanden wird das Hauptspiel darauf hingewiesen, dass eine Installation notwendig ist. Dabei ist zu erwähnen, dass die Installation einer neueren oder älteren Version, eine bereits installierte Subapp ersetzt, die Daten der Ablage jedoch vorhanden bleiben. Damit wird gewährleistet, dass der Spielfortschritt eines Spielers erhalten bleibt, nachdem dessen Second Screen mit einem Hauptspiel einer neueren Version verbunden wurde und eine Aktualisierung der Subapp stattgefunden hat. Die Installation einer älteren Version wird ermöglicht, damit ein Hauptspiel immer mit der dazugehörigen Subappversion kommunizieren kann und keine Verständnisprobleme von Befehlen und Ereignissen auftreten können.

4.5.4 Ressourcenbündel

Für die Darstellung von Bildern und das Abspielen von Audiodateien im Second Screen Manager werden digitale Daten am mobilen Gerät benötigt, welche von der Subapp-Logik geladen und verwendet werden können. Weiters ist die App für das Ausführen der Logik auf deren Skriptcode angewiesen. Diese Daten werden im Zuge der Installation am Gerät gespeichert und stehen bis zur Deinstallation der Subapp zur Verfügung.

Nachdem Endkunden für die Verwendung der Subapp das dazugehörige Hauptspiel bereits besitzen müssen, können die Ressourcen vom Spiel selbst bereitgestellt werden. Die benötigten Daten können dadurch schneller verfügbar gemacht werden, da sie nur vom Hauptspiel zum Second Screen Manager übertragen werden müssen. Das Herunterladen der Ressourcen aus dem Internet wäre in Bezug auf die direkte Übertragung zeitaufwendiger und für Endkunden könnten möglicherweise Kosten für das verbrauchte Datenvolumen anfallen. Um die Übertragung der Ressourcen zu vereinfachen wurde entschieden, dass alle Ressourcen als ein komprimiertes Archiv, dem sogenannten Ressourcenbündel, vom Hauptspiel zur Second Screen Manager geladen werden. Dadurch können Spieleentwickler mit einem Aufruf alle

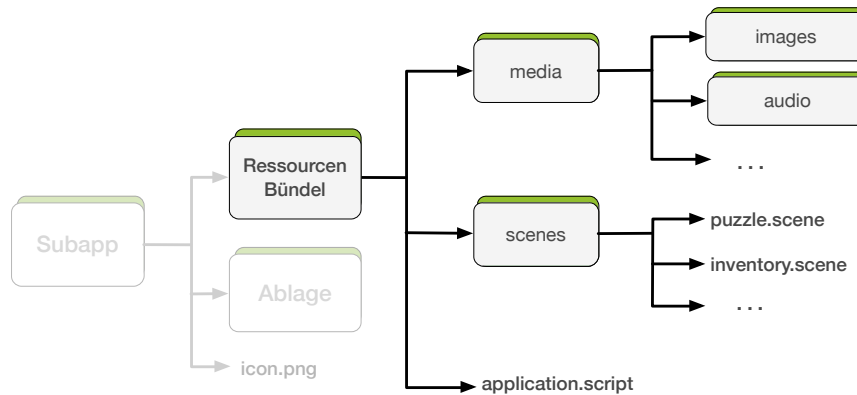


Abbildung 4.10: Verzeichnisstruktur des Ressourcenbündels einer Subapp.

Daten zur App liefern und müssen nicht für jede Datei einen eigenen Befehl ausführen. Um zu gewährleisten, dass der Second Screen Manager das Hauptskript und die Szenen findet, muss das für die Installation benötigte Archiv wie in [Abbildung 4.10](#) aufgebaut werden. `application.script` darf sich in keinem Unterverzeichnis befinden und die Szenen müssen im Ordner `scenes/` abgelegt sein. Weitere für die Subapp relevanten Ressourcen, wie Bilder oder Audiodateien, können in einer von Entwicklern definierten Ordnerstruktur, z. B. im `media` Verzeichnis, abgelegt werden.

4.5.5 Ablage

Für Daten, welche zur Laufzeit von der Subapp gespeichert werden müssen, werden den Entwicklern drei verschiedenen Verzeichnisse angeboten, welche die sogenannte *Ablage* darstellen (s. [Abb. 4.9](#)).

Privat: Im privaten Verzeichnis sind die erzeugten Daten vor Zugriffen anderer Subapps und mobiler Apps geschützt. Weiters haben auch Endkunden keinen direkten Zugang zu diesen Daten. Dies bietet sich unter anderem sehr gut für das Speichern von Spielständen an, welche nicht manipuliert werden dürfen.

Öffentlich: Um erzeugte Grafiken oder von Benutzern erstellte Inhalte abzuspeichern kann das öffentliche Verzeichnis verwendet werden. Dadurch können Endkunden z. B. diese Daten über andere mobile Apps erreichen oder auf einem, mit USB verbundenen Computer, auf diese zugreifen.

Temporär: Flüchtige und nur zur Laufzeit benötigte Daten können im Verzeichnis für temporäre Dateien abgelegt werden. Wenn eine Subapp ein erneutes Mal startet, ist nicht garantiert, dass diese Daten noch vorhanden

sind, da sie von der Plattform oder von Endkunden zu jeder Zeit gelöscht werden können um Speicherkapazität am Gerät wieder zu gewinnen. Dennoch sollten Entwickler, wenn möglich, ihre nicht mehr benötigten, temporären Dateien, selbstständig löschen. Temporäre Dateien bleiben wie die privaten Dateien vor den Benutzern und anderen mobilen Apps verborgen.

Abbildung der Verzeichnisse auf konkreten Plattformen

Die oben beschriebene Verzeichnisstruktur findet sich unter anderem bei Apps für Android, iOS und Windows Phone 8 – den derzeit drei verbreitetsten mobilen Plattformen [42] – wieder. Jedoch spezifiziert jede Plattform andere Speicherorte für diese Struktur. Um die Subapp dennoch plattformunabhängig zu gestalten, werden die aufgezeigten Verzeichnisse vom Second Screen Manager auf die Speicherorte, wie Tabelle 4.1 zeigt, abgebildet. Daraus folgen u. a. zwei Vorteile:

1. Hersteller der drei mobilen Plattformen ermöglichen ihren Benutzern den Zugang zu den öffentlichen Daten einer jeweiligen App (s. Tabelle 4.2). Obwohl die Speicherorte der Daten plattformabhängig definiert werden, können Benutzer aufgrund der Abbildung der Verzeichnisse auf öffentliche Daten der Subapp zugreifen.
2. Je Plattform steht ein Speicherort zur Verfügung, welcher für temporäre Daten genutzt werden kann. Durch die Abbildung des Verzeichnisses für diese flüchtigen Dateien der Subapp auf den, von der Plattform spezifizierten Speicherort, kann sich die Plattform um die Löschung dieser Daten kümmern, wenn Speicherkapazität benötigt wird.

Tabelle 4.1: Abbildung der Verzeichnisse von Subapps auf die darunterliegende Speicherverwaltung der drei verbreitetsten mobilen Plattformen [42].

	<i>iOS 7</i>	<i>Android 4</i>	<i>Windows Phone 8</i>
<i>Privat</i>	Library	Internal Storage	LocalFolder
<i>Öffentlich</i>	Documents	External Storage	RoamingFolder
<i>Temporär</i>	tmp	Cache Directory	TemporaryFolder

Sicherheit der privaten Daten

Um die Sicherheit der Daten im privaten Verzeichnis zu gewährleisten, müssen die Daten verschlüsselt abgelegt werden. Die Tabelle 4.2 zeigt, dass eine Sicherheit auf unverschlüsselte Daten im privaten Verzeichnis den Entwicklern nicht ermöglicht werden kann, da nicht jede Plattform einen Speicherort zur Verfügung stellt, auf welchen nur der Second Screen Manager Zugriff hat.

Tabelle 4.2: Darstellung der Leserechte auf das jeweilige Verzeichnis aus der Sicht anderer Subapps und mobiler Apps welche am selben Gerät installiert sind, sowie aus der Sicht der Eigentümer der Geräte. Ausgenommen sind Eigentümer, welche aufgrund eines Jailbreaks vollen Zugriff auf ihre Geräte erlangt haben.

	<i>Verzeichnis</i>	<i>Subapps</i>	<i>Apps</i>	<i>Eigentümer</i>
<i>iOS 7</i>	Library	—	—	—
	Documents	—	—	+
	tmp	—	—	—
<i>Android 4</i>	Internal Storage	—	+	+
	External Storage	—	+	+
	Cache Directory	—	+	+
<i>WP 8</i>	LocalFolder	—	—	+
	RoamingFolder	—	—	+
	TemporaryFolder	—	—	+

In der Tabelle ist zu beachten, dass vom entwickelten Framework definiert wird, dass eine Subapp keinen Zugriff auf Daten anderer Subapps erhält.

Eine Alternative zur Verschlüsselung der Daten wäre die Speicherung der Daten auf einem Onlineserver, zu welchem nur die Subapp Zugriff hätte. Dieses Konzept findet unter anderem Verwendung in den Standalone Second Screen Apps von *Ubisoft* und *EA* aus Abschnitt 2.3.3. Jedoch wurde die Entwicklung eines allgemeinen Datenservers in dieser Arbeit nicht weiter konkretisiert. Es kann angenommen werden, dass Subappentwickler ihre Daten selbst verwalten möchten und wenn für diese die Sicherheit der Daten relevant ist, eigene Datenserver errichten würden oder bereits welche besitzen.

Eine Speicherung der Daten am Hauptspiel ist nur bedingt möglich, da in der unabhängigen Nutzung diese Daten nicht zur Verfügung stehen würden.

4.5.6 Sandbox

Ein wesentlicher Punkt der Speicherung der Ressourcen und Dateien im Second Screen Manager ist, dass jede Subapp in einer eigenen Sandbox ausgeführt wird, die den Zugriff auf eigenen Ressourcen und Dateien beschränkt (s. Abb. 4.11). Eine Subapp kann daher keine Ressourcen und Spieldaten von anderen manipulieren und verwenden. Das Ermöglichen eines Austauschs von Daten zwischen Subapps ist nicht erforderlich, da es in keinem Zusammenhang mit der Verwendung von Second Screen Gaming steht. Ein Second Screen ist nur einem Hauptspiel zugeordnet und laut den Nachforschungen

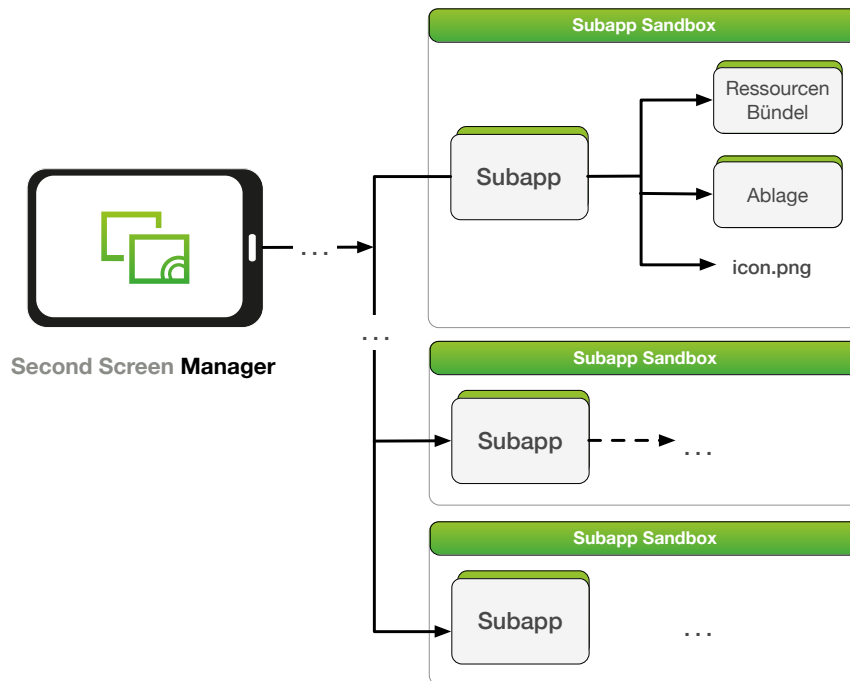


Abbildung 4.11: Darstellung der Subapps im Second Screen Manager. Eine Subapp hat nur Zugriff auf das eigene Ressourcenbündel, die eigenen Daten der Ablage und auf das Icon.

wurde bei Spielen i. Allg. kein Anwendungsfall gefunden, bei welchem zwischen zwei verschiedenen Titeln Daten ausgetauscht wurden.

Kapitel 5

Kommunikation

Das nachfolgende Kapitel beschreibt die Kommunikation zwischen Subapp und Hauptspiel – welche Schritte notwendig sind, um eine Verbindung zwischen Hauptspiel und dem Second Screen zu etablieren, welche Elemente für die Installation einer Subapp benötigt werden und wie Befehle und Ereignisse übertragen werden können.

5.1 Wahl des Übertragungsweges

Aktuelle mobile Endgeräte und Computer besitzen die Möglichkeit über ein *Wireless Local Area Network* (WLAN) miteinander zu kommunizieren. Dabei verbinden sich mobile Endgeräte mit ihren Wireless LAN Adaptern, und Computer über Kabel und/oder ebenfalls drahtlos, mit dem selben LAN. Diese gegenseitige Erreichbarkeit ist eine technische Voraussetzung für die Kommunikation zwischen den Subapps und deren Hauptspielen.

Eine Bluetoothverbindung wird nicht per se ausgeschlossen, obgleich deren Bandbreite aktuell geringer ist als eine WLAN-Verbindung [49]. Bei der Wahl des Kommunikationskanals ist die Latenz zu beachten. Die Latenz spielt eine wichtige Rolle bei der Implementierung eines Mehrspieler-Spiels, welcher in Echtzeit und nicht rundenbasiert, zwischen Subapp und Hauptspiel, ausgetragen wird. Dabei müssen Informationen zwischen den Geräten – im Idealfall – schneller ausgetauscht werden können, als das Spiel voranschreitet [2]:

Angenommen, Nico, aus dem Szenario in Abschnitt 4.2.5, befindet sich vor einem Hindernis. Seine Position wird an Janas Second Screen übertragen und dort mit einer Figur angezeigt. Jana aktiviert das Hindernis um Nico zu behindern und wählt das Hindernis aus. Die Auswahl des Hindernisses wird an das Hauptspiel zurückgesendet. Das Hauptspiel löst das Hindernis nach Erhalt dieses Befehls aus.

Würde die Übertragung der Positionsdaten der Spielfigur länger dauern als sich Nico fortbewegen kann, – weil die Latenz zu hoch ist – würde Jana

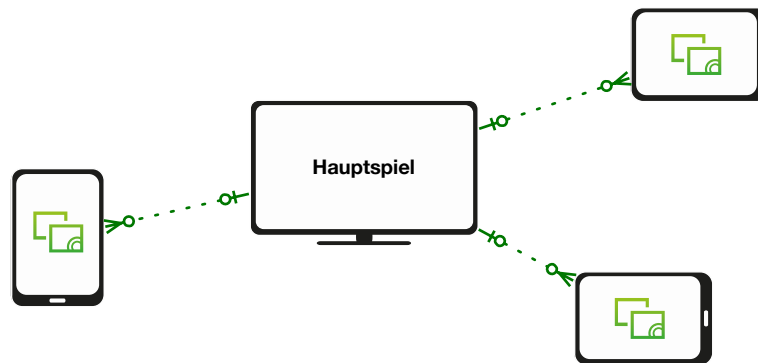


Abbildung 5.1: Ein Hauptspiel kann mehrere Second Screen Manager steuern. Jedoch kann sich ein Second Screen Manager nur mit einem Hauptspiel verbinden.

Nico noch vor dem Hindernis sehen, obwohl er in seinem Spiel schon darüber hinweg ist und einem neuen Hindernis entgegenläuft. Das Auslösen des Hindernisses würde keinen Effekt erzielen. Das Auslösen einer Falle würde Nico auch nicht beeinflussen, wenn der benötigte Befehl nicht rechtzeitig am Hauptspiel einlangen kann.

5.2 Hauptverbindung

Für die Kommunikation zwischen Hauptspiel und Second Screen Manager wird eine TCP Verbindung aufgebaut, über welche Befehle und Ereignisse ausgetauscht werden können und welche als *Hauptverbindung* bezeichnet wird. Neben der Hauptverbindung können Dateien über eine separate Datenverbindung übertragen werden, mit welcher auch das Ressourcenbündel der Subapp zum SSM übertragen wird. Besteht eine Hauptverbindung zwischen dem Hauptspiel und dem Manager, kann sich kein weiteres Hauptspiel mit dem Manager verbinden. Das verhindert ein mögliches Duellieren unter Befehlen des selben Hauptspiels. Des Weiteren kann am Manager nur eine Subapp zur selben Zeit ausgeführt werden, mit welchem einzig ein zugehöriges Hauptspiel kommunizieren kann. Jedoch kann ein Hauptspiel mehrere Hauptverbindungen zu unterschiedlichen Managern aufnehmen (s. Abb. 5.1). Zum Beispiel könnte eine Subapp am Smartphone eine aktuelle Aufgabenliste darstellen, während am Tablet die Spielweltkarte geöffnet ist. Beide Geräte sind mit dem selben Hauptspiel verbunden. Für den Austausch der Befehle und Ereignisse wird ein Eventprotokoll verwendet (s. Abschn. 5.3).

Nachfolgend wird eine grobe Übersicht über den Ablauf der Kommunikation über die Hauptverbindung gegeben.

Aufbau

Der Aufbau der Hauptverbindung wird in drei Schritte unterteilt:

- Erkennung,
- Verbindungsanfrage,
- Annahme der Verbindung.

Dabei agiert der Second Screen Manager als Server zu welchem sich das Hauptspiel als Client verbinden kann (s. Abb. 5.2 (a)). Die Erkennung gibt Aufschluss darüber, zu welcher IP und zu welchem Port die Verbindungsanfrage zu richten ist (s. Abschn. 5.4.1). Anschließend stellt das Hauptspiel die Anfrage und erhält vom Manager die Annahme oder Ablehnung der Verbindung.

Installation

Wurde eine Verbindung erfolgreich vom Second Screen Manager angenommen, wird eine Installation, wie folgt, vom Hauptspiel vorgenommen:

- initiieren der Installation,
- senden des Subapp Icons,
- senden des Ressourcenbündels,
- Installation abschließen.

Dabei wird die Installation über die Hauptverbindung initiiert und das Icon und das Ressourcenbündel im Anschluss über eine separate Datenverbindung übertragen (s. Abb. 5.2 (b)). Abschließend wird die Installation über die Hauptverbindung beendet.

Austausch von Events

Wird die Subapp ausgeführt, werden Events über die Hauptverbindung an die Subapp weitergegeben. Die nachfolgenden Übertragungen können solange die Hauptverbindung steht in beliebiger Reihenfolge und Anzahl erfolgen:

- senden eines Events vom Spiel zur Subapp,
- senden eines Events von der Subapp zum Spiel,
- konkret auf einen Event antworten.

Dabei ist das Client-Server-Modell vom Aufbau der Verbindung nicht mehr ersichtlich, nachdem Subapp und Hauptspiel asynchron miteinander Events austauschen können (s. Abb. 5.2 (c)).

Abbau

Ein geordneter Abbau der Hauptverbindung kann von beiden Teilnehmern vorgenommen werden (s. Abschn. 5.4.5 und Abb. 5.2 (d)).

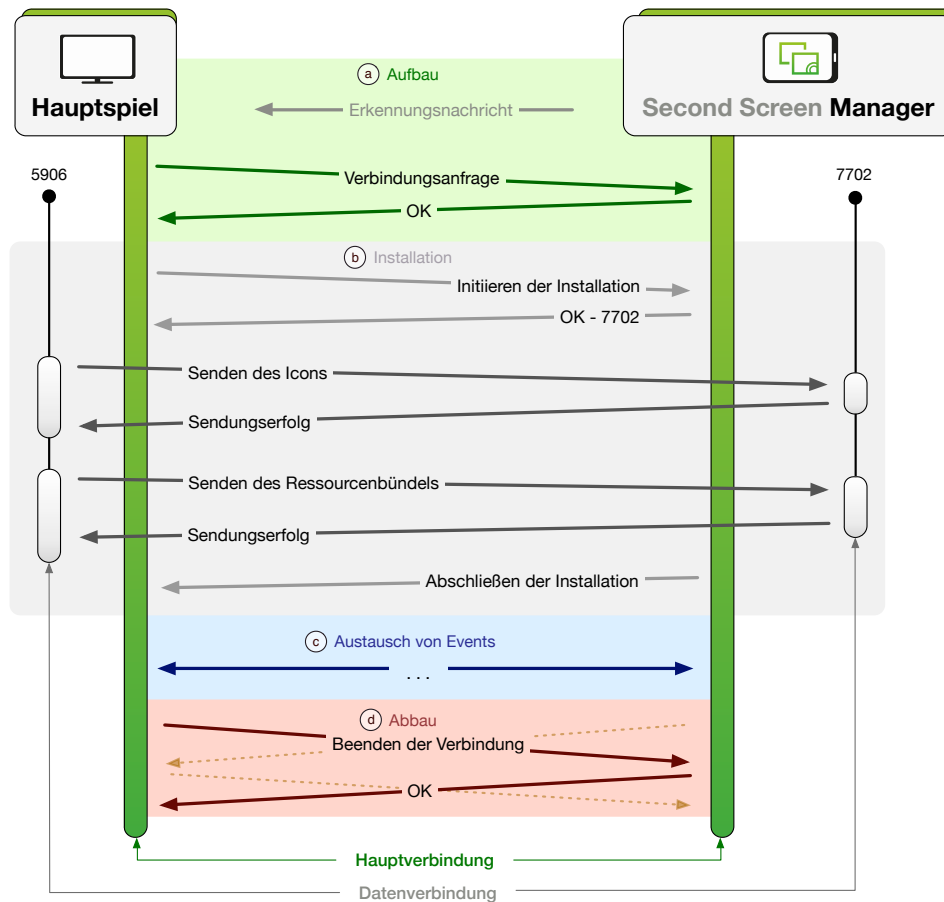


Abbildung 5.2: Kommunikationsablauf der Hauptverbindung mit der Installation einer Subapp über die Datenverbindung.

5.3 Eventprotokoll

Mit dem Eventprotokoll werden asynchron Events über eine TCP Verbindung bidirektional übertragen. Dabei kann sich ein Client zum Eventserver verbinden. Dem Eventserver wird kein spezifischer Port zugewiesen damit jede Anwendung einen Eventserver zur Verfügung stellen kann.

5.3.1 Event

Ein Event besitzt einen Typ und einen Datenblock. Mit dem Typ kann festgestellt werden wie der Datenblock zu interpretieren ist. Der Datenblock besteht aus einer Liste von Schlüsselwort-Wert-Paaren welche in Subblöcke gegliedert werden können. Ein Subblock besitzt einen Namen und kann mit einer ID versehen werden.

Übertragen wird der Event als eine Zeichenkette, welche folgende Struktur aufweist:

```
event <type> {
  <key> = <value>
  <key> = <value>

  <block_name> <id> {
    <key> = <value>
    <key> = <value>
  }
}
```

Werden für einen Event keine Daten benötigt, müssen dennoch die geschwungenen Klammern nachgestellt werden:

```
event <type> {}
```

5.3.2 Übertragung

Jede Übertragung von Events besteht aus einer Kopfzeile und einem Inhalt. Die Kopfzeile beinhaltet vier Zeichen, welche die Länge des Inhalts als String definiert. Der Inhalt besteht aus einem oder mehreren Events. Dabei wird jeder Event durch einen Zeilenumbruch getrennt. Eine Übertragung von zwei Events könnte folgendermaßen abgebildet werden:

```
0082
event·collected-item·{
··item·=·Blaubeere
··position·=·10,·10
}
event·show-map·{}
```

Die Zahl 82 beschreibt die Anzahl der Zeichen des Bodys inkl. der Zeilenumbrüche und Leerzeichen, welche grau hervorgehoben wurden. Wegen der besseren Lesbarkeit des Beispiels wurde nach dem Header bewusst ein Zeilenumbruch gesetzt, welcher nicht benötigt werden würde.

5.3.3 Sicherstellung der Verbindung

In der Übertragung von Events kann es zu groben Verspätungen der Abarbeitung von Events kommen wenn der Server oder Client in den Standby-Modus versetzt wird. Dabei bleibt u. a. unter iOS die TCP Verbindung erhalten und kann weiterhin Events empfangen. Jedoch werden diese durch den schlafen gelegten Haupt-Thread nicht mehr zum Zeitpunkt des Empfangs abgearbeitet.

Das kann vor allem zu Problemen führen, wenn Befehle versendet werden, welche umgehend ausgeführt werden müssen. Befindet sich beispielsweise der Second Screen im Standby-Modus und das Hauptspiel versendet in jedem Game-Loop-Schritt einen Event mit der Position des Spielers um eine Markierung auf einer Karte am Second Screen zu aktualisieren, werden diese Positionen angehäuft und können nicht verarbeitet werden. Erwacht der Second Screen aus dem Standby-Modus werden sogleich alle Positionen abgearbeitet. Um zu verhindern, dass solche Events angehäuft werden, wird

```
event isAlive {  
    timestamp = <value>  
}
```

von jedem Peer versendet, sofern kein anderer Event innerhalb einer bestimmten Zeit verschickt wurde.

Wurde innerhalb der definierten Zeit kein Event empfangen, wird davon ausgegangen, dass die Verbindung tot ist und es werden keine weiteren Events mehr verschickt. Dadurch wird die Anhäufung von Events vermieden.

Verglichen werden kann dieses Konzept mit TCP Keepalive, welches durch Zusenden eines leeren Pakets feststellt ob noch eine valide Verbindung besteht [35].

5.4 Konkreter Ablauf

Im Folgenden werden die benötigten Schritte beschrieben, welche ein Hauptspiel durchführen muss um sich mit dem Second Screen Manager zu verbinden und um eine Subapp zu installieren.

5.4.1 Erkennung

Wird die Second Screen Manager am mobilen Endgerät gestartet, macht sich dieser mit einem Broadcast im Netzwerk bemerkbar. Dabei wird dessen IP-Adresse bekannt und ein Hauptspiel kann eine Hauptverbindung aufbauen. Zusätzlich enthält die Nachricht des Broadcasts folgende Daten:

Identifikationstoken: Um den Broadcast dem Second Screen Manager zuordnen zu können, wird der Erkennungsnachricht ein Identifikationstoken beigelegt. Damit kann sichergestellt werden, dass der Broadcast nicht einer anderen Anwendung entstammt.

Name des Second Screen Managers: Benutzern wird ermöglicht dem Second Screen Manager einen Namen zuzuteilen. Dadurch können Second Screens anhand ihrer Namen am Hauptspiel von Spielern differenziert werden, wenn sich mehrere Geräte im selben Netzwerk befinden.

Passwortschutz: Wie im anschließenden Abschnitt definiert wird, muss für den Verbindungsaufbau ein Passwort übertragen werden. Dieses Passwort wird vom Benutzer definiert und ist weder im Manager noch im Hauptspiel vorgegeben. Darum muss vor dem Verbindungsaufbau das Hauptspiel vom Benutzer das Kennwort abfragen. Entschließt sich ein Benutzer dazu kein Passwort am Second Screen zu definieren, wird eine Eingabe nicht zwingend benötigt. Um den Benutzer nicht unnötig zu einer Eingabe eines leeren Passworts zu verpflichten, wird in der Erkennungsnachricht vermerkt ob ein Kennwort erforderlich ist.

Geräteinformation: Obwohl die Frameworks plattformunabhängig verwendbar sind, können Unterschiede der Hardware vorliegen. Um vom Hauptspiel auf diese Unterschiede im Vorfeld eingehen zu können wird in der Erkennung mitgeschickt, auf welcher Plattform und auf welchem Gerät der Second Screen Manager ausgeführt wird. Nachdem sich ein Hauptspiel zu mehr als nur einem Second Screen Manager verbinden kann, könnte z. B. das Spiel einen Second Screen speziell für Smartphones und einen speziell für Tablets zur Verfügung stellen.

Version: Ein weiterer wesentlicher Punkt ist am Hauptspiel zu erfahren, welche Version der Manager aufweist. Anhand dieser Information kann entschieden werden, wie die Events für die Verbindung und Installation aufgebaut werden müssen. Dadurch können in neueren Versionen des Managers Verbesserungen in der Netzwerkkommunikation vorgenommen werden, wenn sich die Strukturen der Events dafür verändern müssten.

Portnummer zum Aufbauen der Verbindung: Anhand der übergebenen Portnummer kann eruiert werden, an welchen Port die Verbindungsanfrage geschickt werden muss.

5.4.2 Verbindungsanfrage

Wird der Second Screen Manager im Netzwerk erkannt, kann eine Hauptverbindung aufgebaut werden. Um den Zugriff Fremder auf einen Second Screen Manager zu verhindern, ist zu Beginn nur der Befehl/Event für die Verbindungsanfrage zulässig:

```
event Connect {  
    password      = <value>  
    app           = <id>,<name>,<version>  
    company       = <id>,<name>  
    license_key   = <value>  
    subapp_version = <major>.<minor>.<patch>  
}
```

Erst wenn die Anfrage vom Manager bestätigt wurde, können weitere Events verarbeitet werden.

Benutzerdefiniertes Passwort: Das zu übermittelnde Passwort kann von einem Benutzer am Second Screen definiert werden.

Identifikation des Spiels: Die Identifikation beinhaltet eindeutig maschinell differenzierbare IDs für Spiel und Hersteller, sowie deren für Menschen lesbare Bezeichnungen. Anhand der ID des Spiels und Herstellers kann eine installierte Subapp eindeutig identifiziert werden. Die lesbare Bezeichnung kann am Second Screen verwendet werden, um fehlgeschlagene Verbindungsversuche zu loggen bzw. nach erfolgreicher Verbindung am Display anzuzeigen, zu welchem Hauptspiel eine Verbindung besteht.

Lizenzschlüssel: Die Richtigkeit der Identifikation wird mit dem Lizenzschlüssel am Second Screen festgestellt. Das garantiert, dass sich keine Software als eine andere ausgeben kann.

Versionsnummer der Subapp: Im Zuge der Verbindungsanfrage wird die benötigte Versionsnummer der Subapp der Anfrage beigelegt, damit der Second Screen Manager feststellen kann, ob eine Installation vorgenommen werden muss.

5.4.3 Annahme der Verbindung

Nach einer Verbindungsanfrage von einem Hauptspiel wird das übermittelte Passwort und die Identifikation des Spiel überprüft. Stimmt das Passwort mit dem vom Benutzer definierten Kennwort überein und kann die Identifikation mit dem Lizenzschlüssel verifiziert werden, wird die Verbindung akzeptiert. Würde eine der beiden Überprüfungen fehlschlagen wird die Verbindung verweigert. In beiden genannten Fällen wird der Event

```
event Accept {
    accepted          = <true/false>
    [session-id       = <value>]
    [resources-required = <true/false>]
}
```

mit dem Verbindungserfolg an das Hauptspiel zurückgesendet.

Verbindungserfolg: Mit dem Verbindungserfolg (*accepted*) wird mitgeteilt ob die Verbindung von der Second Screen Manager angenommen oder verweigert wurde.

Session-ID: Wird die Verbindung akzeptiert wird ein eindeutig identifizierbarer Schlüssel generiert, welcher dem **Accept**-Event mitgegeben wird. Mit diesem Schlüssel können vom Hauptspiel weitere TCP Verbindungen hergestellt oder UDP Pakete gesendet werden, ohne dass eine erneute Verbindungsanfrage nötig wäre.

Installationsnachweis: Dem Hauptspiel wird im Installationsnachweis (*resources-required*) mitgeteilt, ob die benötigte Version der Subapp vorhanden ist, oder installiert werden muss.

5.4.4 Installation

Eine Installation der Subapp speichert alle für die Subapp benötigten Ressourcen auf dem mobilen Gerät ab. Dazu initiiert das Spiel eine Installation über die Hauptverbindung und sendet das Subapp Icon und dessen Ressourcenbündel über eine Datenverbindung zum Manager. Dies ähnelt der Struktur vom *File Transfer Protocol* (FTP), in welcher in einer Steuerverbindung ein Datenaustausch initiiert wird und über die Datenverbindung nur Daten ausgetauscht werden [17, S. 259–262][7, S. 216–229]. Dadurch bleibt die Hauptverbindung ansprechbar, während über die Datenverbindung nur Daten verschickt werden.

Beginn der Installation

Initiiert wird die Installation mit dem **Install** Event des Hauptspiels. Kann eine Installation vorgenommen werden antwortet der Manager mit dem Event:

```
event InstallOk {  
    port = 7702  
}
```

Dieser beinhaltet den Port, zu dem die Datenverbindung aufgebaut wird. Ist eine Installation nicht möglich, antwortet der Manager mit

```
event InstallDenied {  
    reason = <value>  
}
```

und definiert im Schlüsselwort **reason** den Grund der Ablehnung.

Hochladen der Daten

Wurde die Datenverbindung erfolgreich hergestellt, werden das Icon und das Ressourcenbündel in separaten Dateiübertragungen hochgeladen. Eine Dateiübertragung wird mit dem Zusenden der Identifikation und der Anzahl

an Bytes der hochzuladenden Datei eingeleitet. Anschließend folgen deren Daten nach zwei Zeilenumbrüchen:

```
<Identifikation>
<Byteanzahl>

0x01 0x02 0x03 0x04 ...
```

Identifikation: Die Identifikation beruht auf dem Konzept nach *Uniform Resource Identifier* (URI) [45] und ermöglicht u. a. zwischen dem Ressourcenbündel und dem Icon zu unterscheiden. Einheitlich wird für das Hochladen beider Dateien das Schema `resource://` definiert. Darauf folgt die Session-ID der Hauptverbindung für die Authentifizierung (s. Abschn. 5.4.3) und eine Pfadangabe. Ein Hashwert der Datei kann im Abfrageteil mit dem Namen `hash` hinzugefügt werden. Der Second Screen Manager erkennt das Ressourcenbündel anhand des Pfades `install` und das Icon anhand des Pfades `icon`. Daraus ergeben sich folgende URIs:

```
resource://<session-id>/install?hash=<value>
resource://<session-id>/icon?hash=<value>
```

Hashwert: Von der Archivdatei wird vom Hauptspiel ein Hashwert erstellt und dem Second Screen Manager mitgeteilt. Anhand dieses Werts kann nach der Übertragung festgestellt werden, ob die Daten richtig empfangen und nicht durch Dritte manipuliert wurden. Überprüft wird dies, indem die App einen Hashwert von den empfangenen Daten erstellt und diesen mit dem ursprünglichen Hashwert abgleicht. Dennoch ist hier anzumerken, dass nur eine verschlüsselte Verbindung durch Dritte nicht manipuliert werden kann, nachdem nicht herausgefunden werden kann, ob der Hashwert selbst manipuliert wurde.

Empfangen der Daten

Wurde das Ressourcenbündel erfolgreich vom Second Screen Manager empfangen, wird dessen Inhalt validiert. Dabei wird überprüft, ob mindestens eine Szene und die in Abschnitt 4.5.2 beschriebene `application.script`-Datei vorhanden sind. Sind diese Ressourcen vorhanden, entpackt der Manager die Dateien in ein eigenes, für die Subapp erstelltes, Verzeichnis. Ist bereits eine andere Version dieser Subapp installiert, werden deren installierte Ressourcen zuvor gelöscht. Von der Subapp erstellte Daten der Ablage bleiben jedoch erhalten. Die Versionsnummer, welche in der Verbindungsanfrage übermittelt wurde, wird für die Installation herangezogen und kann im Zuge der Installation nicht verändert werden. Ist dieser Prozess abgeschlossen wird ein Erfolgsnachweis an das Hauptspiel zurückgesandt. Damit gilt

die Installation als abgeschlossen. Wurden die Daten nicht korrekt übertragen oder konnten nicht validiert werden, ist die Installation fehlgeschlagen und ein Fehlercode inklusive Beschreibung wird an das Hauptspiel übermittelt.

Subapp Icon

Wird die Subapp zum ersten Mal installiert wird ein Eintrag im SSM-Menü erstellt. Das Icon, welches im Menü die Subapp repräsentiert, wird vor dem Ressourcenbündel übertragen. Das hat den Vorteil, dass während der Installation das Icon bereits im Menü ersichtlich ist und mittels eines Ladeindikators den Benutzer über den Fortschritt informieren kann. Der Zweck dieser Vorschau dient rein dem Design des Second Screen Managers und bedient keine funktionale Wichtigkeit.

5.4.5 Beenden der Verbindung

Das Ende der Hauptverbindung kann vom Manager oder vom Hauptspiel eingeleitet werden. Statt die TCP Verbindung ohne Ankündigung zu trennen, kann davor

```
event End {
    reason = <value>
}
```

gesendet werden um sicherzustellen, dass kein Prozess unterbrochen wird. Darauf reagiert werden kann mit `EndOk` oder `EndWait`.

```
event EndOk {}
```

beschreibt, dass die TCP Verbindung geschlossen werden kann. Falls die Verbindung aufrecht erhalten bleiben sollte, um aktive Vorgänge abschließen zu können, muss mit

```
event EndWait {
    reason = <value>
}
```

geantwortet werden. Das verhindert, dass die Verbindung geschlossen wird. Anschließend wird wieder mit einem `End` versucht die Verbindung zu beenden. Trifft keine Antwort innerhalb eines Time-Outs ein, oder soll ein `EndWait` nicht möglich sein, wird die Verbindung mit

```
event ForceEnd {
    reason = <value>
}
```

geschlossen.

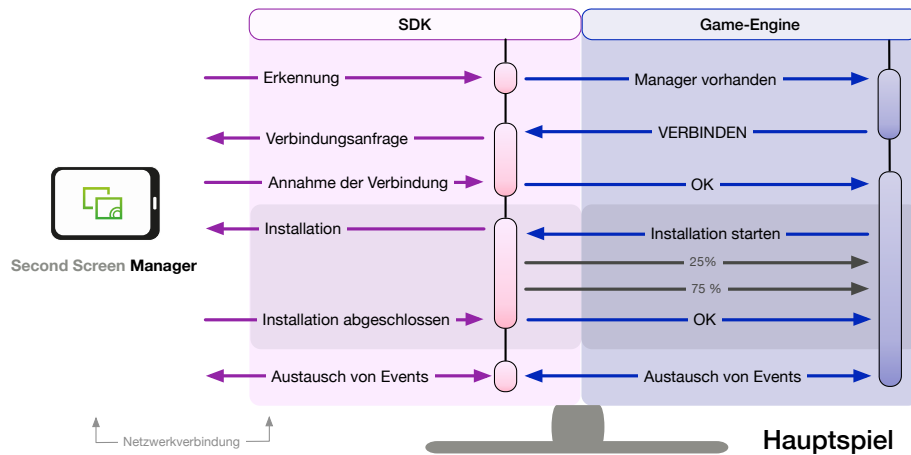


Abbildung 5.3: Das SDK abstrahiert die Netzwerkkommunikation zum Second Screen Manager und stellt Schnittstellen für das Hauptspiel bereit.

5.5 SDK

Als Gegenstück zum Second Screen Manager wurde ein SDK entworfen, welches in ein Hauptspiel integriert werden kann. Das SDK abstrahiert die Netzwerkkommunikation mit dem Second Screen Manager, indem Schnittstellen angeboten werden, welche sich um die Hauptverbindung und Installation kümmern (s. Abb. 5.3). Dabei wurde, wie bereits erwähnt, darauf geachtet, dass die Integration, unabhängig von der Game Engine des Hauptspiels, möglich ist.

5.5.1 Hauptverbindung und Installation der Subapp

Wie oben hervorgeht, wird über die Hauptverbindung mit einem Eventprotokoll kommuniziert. Für den Aufbau und Abbau der Hauptverbindung sowie der Installation stehen vordefinierte Events zur Verfügung (s. Abschn. 5.4). Diese werden im SDK verankert und zum Aufbau der Hauptverbindung verwendet. D. h. diese Events werden für Spielehersteller über Schnittstellen des SDKs aufbereitet. Des Weiteren übernimmt das SDK das Empfangen der Erkennungsnachricht und reagiert darauf. Nachdem sich das SDK nicht selbstständig mit dem Second Screen Manager verbinden kann, da das Passwort für die Verbindungsanfrage unbekannt ist (s. Abschn. 5.4.2), wird dem Hauptspiel mitgeteilt, dass ein Manager im eigenen Netzwerk gefunden wurde. Anschließend kann sich das Hauptspiel entscheiden, sich mit dem Second Screen Manager zu verbinden. Wird eine Installation der Subapp benötigt, kümmert sich das SDK um die Übertragung des Ressourcenbündels und des Icons. Der Fortschritt der Installation wird dem Hauptspiel mitgeteilt und kann über einen Ladebalken am Bildschirm ersichtlich gemacht werden.

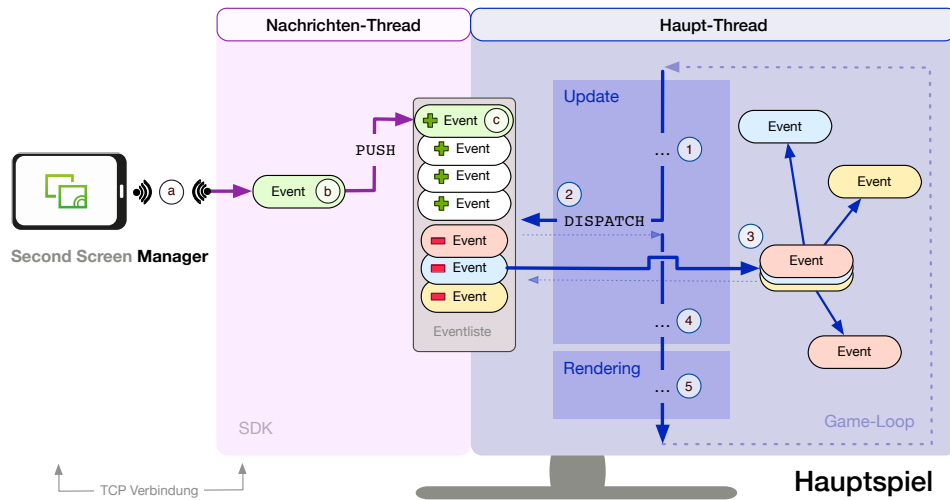


Abbildung 5.4: Events werden im Nachrichten-Thread vom SDK empfangen und zur Eventliste hinzugefügt. Anschließend werden im Rahmen des Game-Loops alle zuletzt hinzugefügten Events aus der Liste wieder entfernt und im Spiel versendet. Im Detail: Der Second Screen Manager versendet Events zum Hauptspiel (a). Der Nachrichten-Thread des SDKs empfängt diese Events (b) und fügt sie mit PUSH der Eventliste hinzu (c). Nach einem beliebigen Code im Update (1) wird die Schnittstelle DISPATCH der Eventliste aufgerufen (2). Diese verteilt die Events im Spiel (3). Anschließend wird wiederum beliebiger Code ausgeführt (4). Zeichenbefehle erzeugen abschließend das Bild für den Monitor (5).

5.5.2 Vermeidung einer Blockierung

Wesentlich in der Integration des SDKs ist die Vermeidung einer Blockierung des Hauptspiels selbst. Das Übertragen des Ressourcenbündels benötigt, abhängig von dessen Größe, Zeit um von der Festplatte geladen und über das Netzwerk verschickt zu werden. Würde dieser Vorgang im Thread des Hauptspiels stattfinden, würde das Spiel angehalten werden bis die Daten übertragen wurden. Nachdem diese Übertragung länger als eine Sekunde andauern kann, entsteht ein ruckeliges Spielverhalten. Das gilt es zu vermeiden. D.h. die Übertragung und Kommunikation mit dem Second Screen Manager muss in einem eigenen Thread stattfinden. Nur so können Spieler den Titel ohne störende, Sekunden anhaltende, Blockierungen genießen. Dennoch müssen die Schnittstellen des SDKs vom Spiel-Thread aufgerufen werden. Das hat den Vorteil, dass sich Spieleentwickler nicht um den internen SDK-Thread kümmern müssen. Um dabei einen sicheren Austausch der Daten zwischen den Threads zu gewährleisten wurde ein Nachrichtensystem entwickelt.

5.5.3 Nachrichtensystem

Mithilfe des Nachrichtensystems können Events über die Hauptverbindung versendet und empfangen werden. Um das Hauptspiel dabei nicht, wie oben erwähnt, zu blockieren, wird die dafür benötigte Netzwerkkommunikation in einem eigenen Thread ausgeführt.

Empfangen von Events

Vom SDK empfangene Events werden in der *Eventliste* des Nachrichtensystems gespeichert. Bei dieser Liste registriert sich das Hauptspiel als Empfänger um über neue Events informiert zu werden. Dazu wird eine Schnittstelle benötigt, welche für das Zusenden von Events verwendet werden kann. Es ist zu beachten, dass diese Schnittstelle nicht zum Zeitpunkt des Eintreffens eines Events informiert wird. Das Spiel muss dem SDK mitteilen, dass empfangene Events an die Schnittstelle zu versenden sind. Dies ermöglicht das Behandeln der Events in einem eigenen Thread (s. Abb. 5.4).

Nachdem das SDK vorrangig für die Integration in eine Game-Engine gedacht ist, wird davon ausgegangen, dass ein Game-Loop für die Spiellogik des Hauptspiels vorhanden ist. Dieser wird i. Allg. im Hauptthread der Anwendung oder in einem, für die Logik bestimmten, Thread ausgeführt [9]. Dabei ist die grundlegende Idee, dass die Verarbeitung der Events in diesem vorhandenen Game-Loop integriert werden kann und Spielehersteller gezielt den Zeitpunkt bestimmen können.

Versenden der Events

Auf der anderen Seite kümmert sich das Nachrichtensystem um das Versenden der Events vom Hauptspiel zum verbundenen Second Screen Manager. Hierbei wird ein Event im Haupt-Thread erzeugt und im Nachrichtensystem in einer Liste gespeichert. Diese werden anschließend im Nachrichten-Thread zum Second Screen Manager versandt.

Kapitel 6

Implementierung

Dieses Kapitel beschäftigt sich mit der praktischen Umsetzung der vorgestellten Konzepte (s. Abschn. 4.4.1 u. 4.4.2). Vorwiegend wird dabei auf die Struktur des Subapp-Frameworks eingegangen.

6.1 Die Frameworks

Wie bereits in Kapitel 4 erwähnt, wurde zunächst ein Ansatz verfolgt, bei welchem spezifizierte Befehle zum Verändern des auf dem Second Screen vorhandenen GUI¹ Verwendung fanden (s. Abschn. 4.4.1). Die Umsetzung davon wird im Folgenden als *Slave-Framework* bezeichnet.

Da sich diese ursprüngliche Idee jedoch nicht als erfolgreich herausstellte, wurde der Programmteil zur Steuerung des Second Screens verworfen und das *Subapp-Framework* anhand des finalen Konzepts (s. Abschn. 4.4.2) entwickelt. Die wesentliche Idee im Subapp-Framework ist vor allem die Entwicklung von einzelnen Subapps, welche jeweils einen Second Screen für ein Hauptspiel darstellen.

6.2 Verwendete Technologien

Nachfolgend wird auf verwendete Technologien, welche für die Umsetzung herangezogen wurden, sowie auf die Gründe für deren Wahl näher eingegangen. Weiters wird das gewählte Hauptspiel *Elements* vorgestellt, welches für das Entwerfen der Prototypen verwendet wurde.

6.2.1 Game-Engine: xis-engine

Für die Entwicklung des Frameworks wurde eine bestehende Game-Engine als Grundlage für den Second Screen Manager verwendet. Dies ermöglichte es, Spielelemente out of the box für den Second Screen zu implementieren,

¹Graphical User Interface (GUI) bezeichnet die grafische Benutzeroberfläche.

ohne dass ein eigenständiger Game-Loop entwickelt werden musste. Dabei wurde beschlossen, die *xis-engine* zu verwenden. *xis-engine* ist eine plattformunabhängige Game-Engine und ermöglicht das Entwickeln in C++ [20]. Der Vorteil bei dieser Engine war der freie Zugang zum gesamten Quellcode, da es sich hierbei um eine Eigenentwicklung handelt. Benötigte Schnittstellen konnten leicht in das System integriert werden. Im Weiteren ermöglichte die *xis-engine* eine genreunabhängige Spielentwicklung und eignete sich daher besonders für die Erschaffung von vielfältigen Second Screen Anwendungen.

6.2.2 Kommunikationsbibliothek: Boost.Asio

Boost.Asio ist eine plattformunabhängige C++ Netzbibliothek und bietet eine synchrone und asynchrone low-level I/O Programmierung. Sie ist seit 2005 Bestandteil von *Boost*² und konnte seither ausführlich von Benutzern der *Boost*-Bibliothek getestet werden. *Boost.Asio* bietet ein abstrahiertes Input- und Outputsystem an, welches auch für die Kommunikation über COM Serial Ports verwendet werden kann. Auf Basis des Abstraktionslayers können Input und Output synchron und asynchron durchgeführt werden [13]. Die Low-Level Entwicklung ermöglichte es, das geforderte Eventprotokoll (s. Abschn. 5.3) und den Datenserver für die Installation einer Subapp (s. Abschn. 5.4.4) abzubilden. Weiters erlaubte der Abstraktionslayer für I/O eine plattformunabhängige Programmierung. Ein weiterer Vorteil in der Verwendung von *Boost.Asio* war, dass die benötigten Grundbibliotheken der *Boost* bereits im Vorhinein in der *xis-engine* inkludiert waren.

6.2.3 Skriptsprache: Lua

Die verbreitetsten, frei verfügbaren Skriptsprachen in der Entwicklung von Videospielen sind Lua und Python. Diese wurden bereits erfolgreich in der Spieleentwicklung verwendet. U. a. wurde Lua von den Spielen *Grim Fandango*, *Baldur's Gate*, *MDK2*, *Psychonauts* und *Blitzkrieg* verwendet. Python wurde z. B. in den Spielen *Toontown Online*, *Eve Online*, *Blade of Darkness* und *Earth & Beyond* integriert [1, S. 114-115].

Recherchen ergaben, dass Lua den Speicher effizienter nutzt und schneller ist als Python, wohingegen Python eine umfassendere Standardbibliothek an Funktionen bietet und, im Gegensatz zu Lua, für objektorientierte Programmierung ausgelegt ist (s. Tab. 6.1) [40, 41, 50]. Tabelle 6.2 zeigt die jeweilige Syntax der Skriptsprachen. Im Bezug auf diese Arbeit konnten keine ausschlaggebenden Vorteile für die Einbindung einer Skriptsprache gegenüber der anderen festgestellt werden. Nachdem Lua als thread-sicherer eingestuft wurde und das Integrieren der Bibliothek in die *xis-engine* kein Problem darstellte, wurde Lua als Skriptsprache für die Subapps ausgewählt.

² *Boost* ist eine freie C++ Bibliothek, die aus einer Vielzahl von portablen Unterbibliotheken besteht <http://boost.org>.

Tabelle 6.1: Diese Tabelle vergleicht die Skriptsprachen Lua und Python miteinander. [40, 41].

	<i>Lua</i>	<i>Python</i>
Binäre Größe	200 KB (v. 5.1)	2 MB (v 2.6)
Multithreading	Thread-Sicher	schlechte Qualität
Standardbibliothek	klein u. kompakt	umfangreich und mächtig
Objektorientiert	möglich mit <i>metha-methods</i>	konkretes Model vorhanden

Tabelle 6.2: Diese Tabelle stellt die Syntax der Skriptsprachen Lua und Python anhand eines Programmbeispiels gegenüber. Die Programme definieren eine Funktion, welche die Fakultät einer übergebenen Zahl berechnet, fragen den Benutzer nach einer Zahl, und geben dessen Fakultät aus [5, S. 3][51].

<i>Lua</i>	<i>Python</i>
<pre>function fact (n) if n == 0 then return 1 else return n * fact(n-1) end end -- read from console print("Enter a number:") a = io.read("*number") print(fact(a))</pre>	<pre>def fact(n): if n == 0: return 1 else: return n * fact(n-1) # read from console str = input("Enter a number:") a = int(str) print fact(a)</pre>

Kostenpflichtige Skriptsprachen, wie *UnrealScript* oder *Torque Script* wurden in dieser Arbeit nicht in Betracht gezogen.

Lua ist eine dynamisch typisierte Skriptsprache. Der Interpreter von Lua ist in C geschrieben und kann plattformunabhängig für C und C++ Projekte frei verwendet werden. Die aktuelle Bibliothek, Lua 5.2.3³, umfasst nur 572 KB, beinhaltet eine automatische Speicherverwaltung mit einem inkrementellen Garbage Collector⁴, ermöglicht das Aufrufen von C-Funktionen

³Stand 14.09.2014

⁴Der Garbage Collector dient der automatischen Speicherbereinigung von Lua Objekten, welche nicht mehr benötigt werden. Dabei wird das zuletzt angelegte, nicht mehr



Abbildung 6.1: Der Teaser von Elements zeigt die zwei Hauptspielfiguren der Alphaversion, zwischen denen der Spieler kontinuierlich wechseln muss, um die Hindernisse der Maya-Welt zu überwinden.

aus Lua und vice versa, und interpretiert Bytecode. Kommentare starten in Lua mit zwei Bindestriche und lokale Variablen werden mit dem Schlüsselwort `local` definiert. Weiters besitzt Lua eine prozedurale Syntax und verwendet überwiegend assoziative Arrays, welche als *Table* bezeichnet werden [5, 26].

6.2.4 Hauptspiel: Elements

Elements ist ein 2D Jump 'n' Run Spiel, welches für iOS, Android, Windows und MacOS X in der Alphaversion verfügbar ist (s. Abb. 6.1). Die Spielfigur bewegt sich automatisch und der Spieler hat die Aufgabe zu Springen und das Element der Figur zur rechten Zeit zu wechseln um den Gefahren der Spielwelt zu trotzen. Zur Auswahl stehen zwei verschiedene Elemente: Feuer und Stein, welche es zur richtigen Zeit einzusetzen gilt [21].

Entwickelt wurde das Spiel 2013 mit der *xis-engine* in C++ im Rahmen der Lehrveranstaltung *Game Art and Level Design* an der Fachhochschule Hagenberg. Das Team bestand aus vier Mitgliedern: Angelika Bugl, Melanie Freilinger, Christoph Lipphart und Mark Mühlberger.

Für die Entwicklung der Prototypen wurde auf das bestehende Spiel aufgebaut, da die Entwicklung eines eigenständigen Spiels den Rahmen der Arbeit gesprengt hätte. Weiters befand sich *Elements* noch in der Alpha-phase und es war ein freier Zugang zum Quellcode möglich. Das bot ideale Bedingungen für das Entwerfen der Prototypen der Second Screen Anwendungen.

benötigte, Objekt als erstes wieder gelöscht.

6.3 Aufbau der Frameworks

Zur Umsetzung der Frameworks wurde ein Software Development Kit (SDK) für das Hauptspiel in C++ und der Second Screen Manager (SSM) als plattformunabhängige mobile App entwickelt. Um zu gewährleisten, dass die benötigten Events für den Auf- und Abbau der Hauptverbindung zwischen SDK und der mobilen App übereinstimmen, werden diese nur im SDK implementiert. D.h. das SDK wird vom Hauptspiel und vom Second Screen Manager eingebunden.

Für das Eventprotokoll (s. Abschn. 5.3) wurde das `NetEventService` entwickelt, welches das Interface `NetEventConnection` für die Hauptverbindung bereitstellt. Dieses Interface wird sowohl vom SDK als auch vom SSM verwendet.

Weiters entstand im Zuge der Implementierung des Subapp-Frameworks ein *Erweiterungssystem*, welches an die Verwendung des SDKs im Hauptspiel und einer Subapp am Second Screen Manager gebunden ist. Das *Erweiterungssystem* ermöglicht eine modulare Erweiterung (s. Abschn. 6.3.4).

Abbildung 6.2 gibt einen Überblick der vier entwickelten Systeme und bietet zudem eine grobe Übersicht über die Beziehungen zwischen den wesentlichsten Klassen.

6.3.1 NetEventService

Das `NetEventService` implementiert eine Server-Client-Architektur unter Verwendung von *Boost.Asio* für den Austausch der Events über eine asynchrone TCP-Verbindung. Am Second Screen Manager wird der `NetEventServer` ausgeführt, zu welchem sich das SDK mit dem `NetEventClient` verbinden kann.

Von *Boost.Asio* wird hierfür `tcp::socket` und das `io_service` verwendet. `tcp::socket` abstrahiert einen plattformspezifischen Netzwerksocket für eine TCP-Verbindung. Dazu stehen für die Kommunikation u. a. folgende I/O-Befehle zur Verfügung:

```
read(stream,buffer[,extra options])
async_read(stream,buffer[,extra options],handler)
write(stream,buffer[,extra options])
async_write(stream,buffer[,extra options],handler)
```

Ausgeführt werden diese Befehle vom `io_service`. Werden asynchrone Befehle, z.B. `async_read` oder `async_write` verwendet, werden diese im Gegensatz zu synchronen Befehlen im `io_service` gestapelt und nicht sofort ausgeführt. Abgearbeitet wird der Stapel an Befehlen in dem Thread, in welchem `io_service::run` aufgerufen wird. Nachdem `run` eine blockierende Funktion darstellt, wird diese vom SDK in einem eigenen Thread für das `NetEventService` ausgeführt um den Hauptthread nicht zu blockieren.

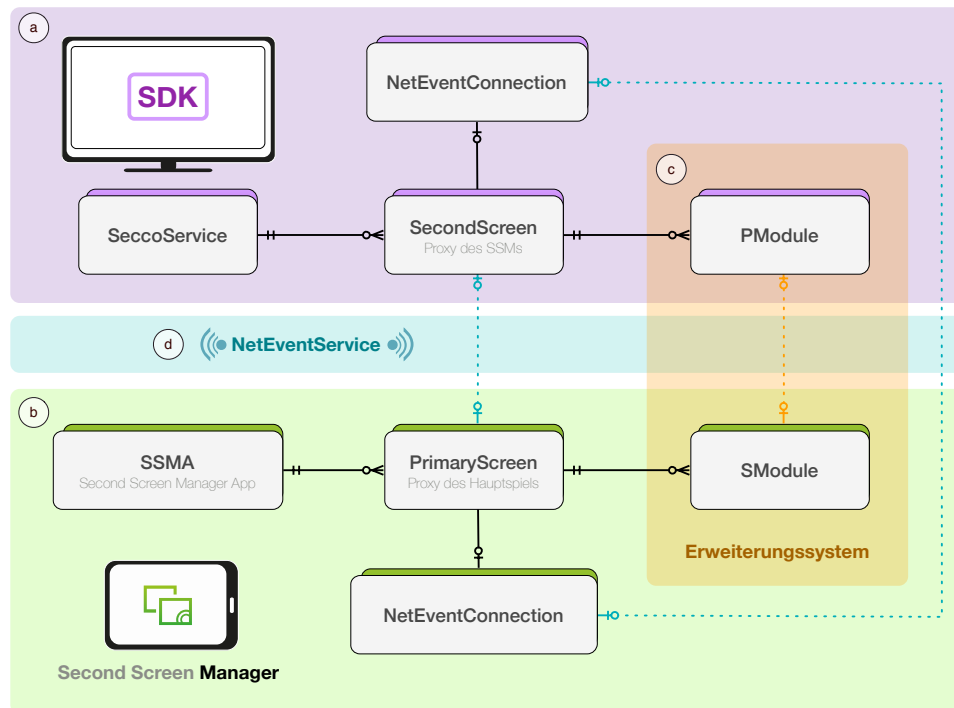


Abbildung 6.2: Die Struktur der Frameworks besteht aus einem SDK für das Hauptspiel (a), dem Second Screen Manager als mobile App (b), einem Modulsystem (c), welches von beiden verwendet wird, und aus einem **NetEventService** (d), welches für den Austausch der Events über das Netzwerk verantwortlich ist.

NetEventConnection

Die **NetEventConnection** ist ein Interface für eine bestehende Verbindung zwischen **NetEventClient** und dem **NetEventServer** und ermöglicht das Austauschen von **NetEvents** (s. Abbildung 6.3). Die Funktion

```
connectTo ( ip, port ) : NetEventConnection
```

des **NetEventClients** eröffnet eine TCP-Verbindung zu einem **NetEvent-Server**. Ist die IP-Adresse und/oder der angegebenen Port unbekannt, kann keine Verbindung erfolgen und **connectTo** gibt **std::nullptr** zurück. Konnte eine Verbindung hergestellt werden, kann mit

```
sendEvent ( event )
```

ein **NetEvent** zum Server versendet werden.

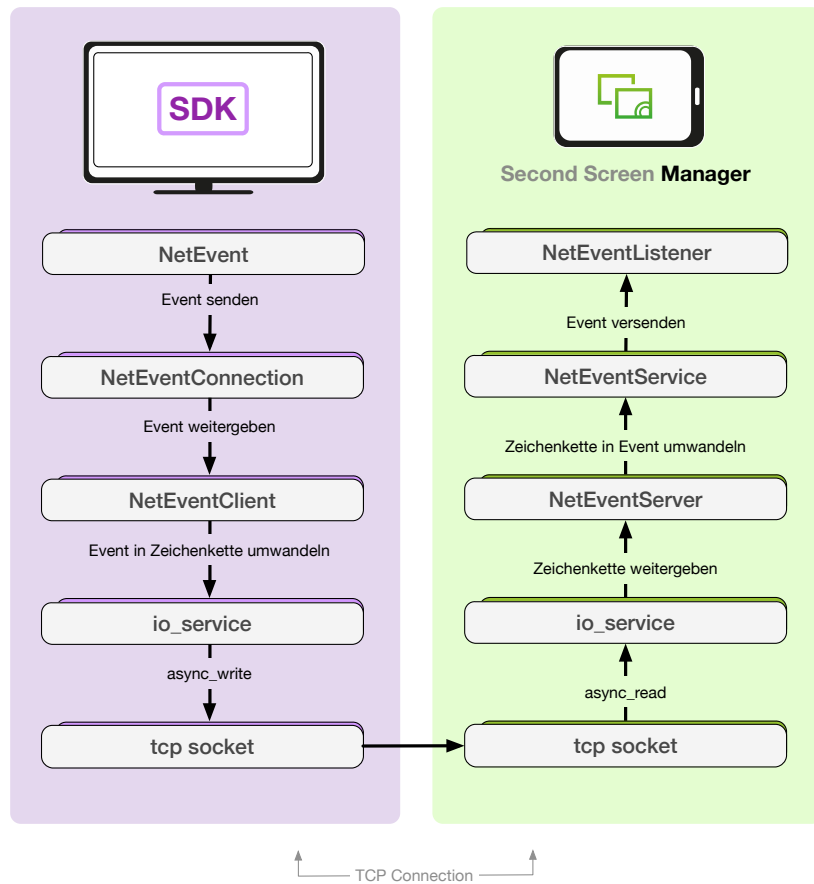


Abbildung 6.3: Die gezeigte Pipeline visualisiert den Vorgang der Übertragung eines **NetEvents** vom SDK zum Second Screen Manager.

NetEvent

Ein konkreter Event wird von der Basisklasse **NetEvent** abgeleitet und, wie oben erwähnt, über eine bestehende *NetEventConnection* versendet. Werden keine zusätzlichen Daten benötigt kann eine Instanz von **NetEvent** mit dem benötigten Typ erzeugt werden ohne davon abzuleiten (s. Abbildung 6.4).

NetEventRestriction

Nachdem jegliche Events versendet werden können und diese auch von der Anwendung durch Callbacks im Programm an unterschiedlichen Orten abgefragt werden können, wird eine zentrale Stelle benötigt, an welcher entschieden wird ob ein Event zulässig ist oder nicht. Daher wurde die Firewall **NetEventRestriction** entwickelt. Diese ermöglicht mit nachfolgenden Schnittstellen Events einzuschränken:

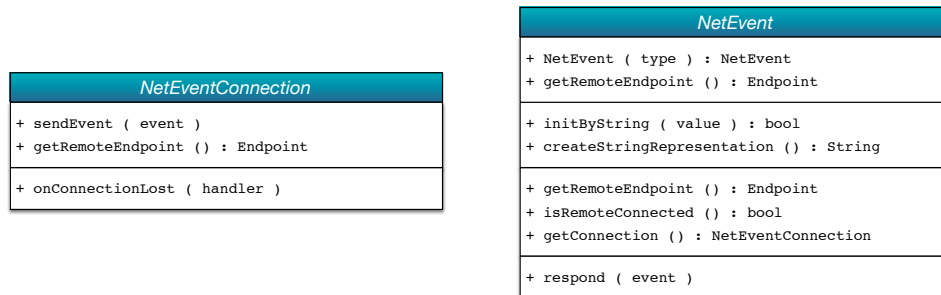


Abbildung 6.4: Die Klassendiagramme zeigen die öffentlichen Schnittstellen, welche am Hauptspiel verwendet werden können.

```

acceptNetEventFromAny ( type )
acceptAnyNetEventFrom ( endpoint )
acceptNetEventFrom ( type, endpoint )
cancelAccept ... ( ... )
clearAllPermissions ( )
  
```

Hierbei wird zwischen dem **Endpoint**, welcher die IP-Adresse und den Port des Senders aufweist, und einem Typ eines Events unterschieden. Mit der Funktion `restrictEvents( true|false )` kann die Firewall aktiviert und deaktiviert werden. Das vermeidet, dass alle erdenklichen Events eingetragen werden müssten, falls keine Einschränkungen benötigt werden.

Verwendung findet **NetEventRestriction** u. a. in der Hauptverbindung zwischen dem SDK und dem SSM (s. auch Abschn. 5.2). Dabei wird die Firewall vom SSM so konfiguriert, dass nur der **Connect**-Event der Hauptverbindung von jedem **Endpoint** zulässig ist.

Verteilung der Events

Das **NetEventService** wurde im SDK implementiert und kann vom Hauptspiel und vom SSM verwendet werden. Jedoch werden die Events für das Hauptspiel anders aufbereitet als sie im Second Screen Manager behandelt werden.

Im SSM: Nachdem es in der *xis-engine* bereits ein globales Eventsystem gibt, wird dieses im SSM herangezogen und empfangene Events können vom **NetEventService**-Thread an den Game-Loop-Thread der *xis-engine* weitergegeben werden. Dafür wird der **NetEvent** von `xis::Event` abgeleitet und mit der globalen Funktion `xis::fireEvent( event )` versendet. Darin besteht der Vorteil, dass im gesamten SSM ein **NetEvent** auf die selbe Weise wie ein Game-Event abgefragt werden kann, während das **NetEventService** von der Spiellogik losgelöst bleibt.

Im Hauptspiel: Auf der Seite des Hauptspiels erfährt das `SeccoService` von neu empfangene Events in dessen innerer Klasse `EventSystem`. Diese wurde dem Nachrichtensystem des Subapp-Konzepts (s. auch Abschn. 5.5.3) nachempfunden. Es stellt eine `push` und eine `dispatch` Funktion bereit um Events hinzuzufügen und zu versenden. Dabei wird `SeccoService` als Fassade für das `EventSystem` verwendet.

Mit der Schnittstelle `SeccoService::onEvent` kann sich das Hauptspiel registrieren, um Events zu erhalten, wenn `SeccoService::dispatchEvents` aufgerufen wird.

6.3.2 Second Screen SDK

Das Second Screen SDK (S3DK) wurde unter Verwendung von plattform-unabhängigen Bibliotheken – u. a. mit *Boost.Asio* und Teilen der *xis-engine* – in C++ entwickelt. Das ermöglicht das Einbinden des S3DKs in die unterschiedlichsten Game-Engines. Eine Game-Engine, welche beispielsweise in Java entwickelt wurde, könnte das S3DK mit dem JNI⁵ einbinden.

Das Interface des S3DKs umfasst nur die wesentlichsten Funktionen welche für die Kommunikation mit dem Second Screen Manager seitens des Hauptspiels notwendig sind. In Abbildung 6.2 (a) sind bereits alle wesentlichen Klassen angeführt. Dabei bildet das `SeccoService` den zentralen Punkt des Systems und kümmert sich primär um das Erkennen und Verbinden von SSMs.

Erkennen des SSMs

Wurde ein Second Screen Manager erkannt, wird umgehend eine Instanz von `SecondScreen`, welche den Proxy⁶ des SSMs im Hauptspiel darstellt, erzeugt und in einer Liste im `SeccoService` gespeichert (s. Abb. 6.5).

Daraufhin wird automatisch eine `NetEventConnection` zum SSM etabliert. Das ermöglicht im Falle des Abbrechens der TCP-Verbindung über den Verlust des SSMs informiert zu werden. Zu diesem Zeitpunkt akzeptiert das `NetEventService` des SSMs nur den `Connect`-Event und reagiert auf jeden anderen Event mit dem `NetEventError`. Dieses Blockieren wird durch die `NetEventRestriction` des `NetEventServices` ermöglicht.

Abschließend wird das Hauptspiel vom `onSecondScreenAvailable`-Callback informiert, dass ein Second Screen zur Verfügung steht.

⁵Mit dem *Java Native Interface* (JNI) kann eine C++ Funktion von Java aus aufgerufen werden und umgekehrt.

⁶Hier gilt Proxy als Stellvertreter für den SSM in der Programmierung des Hauptspiels. In der Netzwerkprogrammierung wird ein Proxy i. Allg. zwischen dem Client und dem Server eingesetzt. Dieser nimmt eine Anfrage eines Clients entgegen, modifiziert diesen ggfs., und sendet ihn an einen Server. Anschließend wird die Antwort des Servers entgegengenommen, ggfs. modifiziert, und an den Client weitergegeben [13, S. 108].

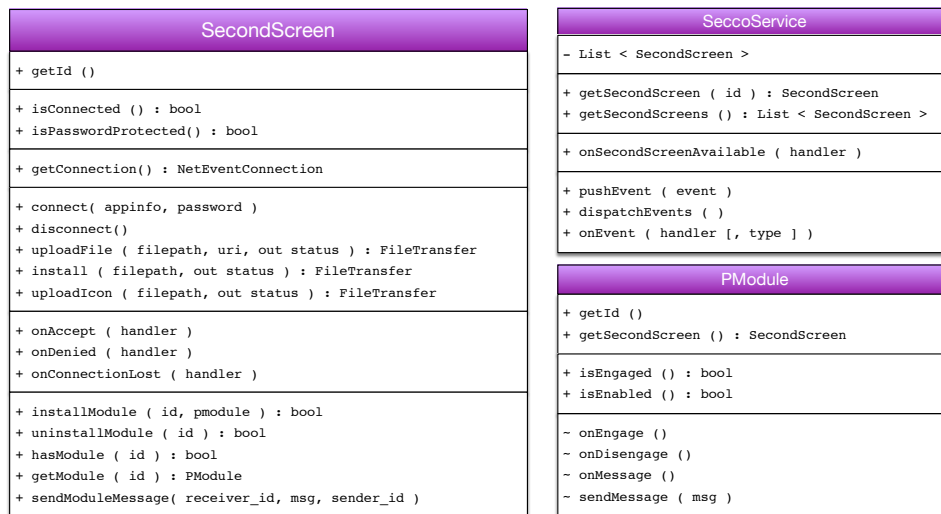


Abbildung 6.5: Diese Grafik beinhaltet die Klassendiagramme der wichtigsten Klassen des SDKs.

Verbinden mit einem SSM

Wurde dem Hauptspiel ein **SecondScreen** bekannt, kann das Hauptspiel Callbacks für das Akzeptieren (`onAccept`) und Ablehnen (`onDenied`) der Verbindungsanfrage registrieren.

Möchte sich das Hauptspiel verbinden, kann eruiert werden, ob dem Benutzer eine Eingabemaske für ein Passwort gezeigt werden muss. Anschließend kann das Hauptspiel die Verbindungsanfrage (s. auch Abschn. 5.4.2) mit der Methode `connect` versenden. Dabei wird ein Event für die Anfrage erzeugt und an die `NetEventConnection` weitergegeben. Daraufhin wird der Event asynchron vom `NetEventService`-Thread verschickt. Nun wartet das SDK auf den `Accept`-Event des Second Screen Managers.

6.3.3 Second Screen Manager

Der Second Screen Manager (SSM) wurde mit der *xis-engine* in C++ entwickelt. Diese ermöglichte es unter der Verwendung des integrierten *GUI-Systems* in kürzester Zeit eine Menüdarstellung der Subapps sowie eine Eingabemaske für einen Namen und ein Passwort zu erstellen. Nachdem die *xis-engine* plattformunabhängig ist, kann der C++ Quellcode des SSMs für unterschiedliche mobile Plattformen genutzt werden. Entwickelt und getestet wurde der SSM anhand einer iOS Version am *iPad Air* und einer macOS X Version am *MacBook Pro*.

SSMA Second Screen Manager App	PrimaryScreen
<pre>- List < SecondScreen ></pre>	<pre>+ getId ()</pre>
<pre>+ getPrimaryScreen (id) : PrimaryScreen + getPrimaryScreens () : List < PrimaryScreen ></pre>	<pre>+ isInstalled (version) : bool + getInstalledVersions () : List < Version > + getCurrentVersion () : Version + getLatestInstalledVersion () : Version</pre>
<pre>+ setName (name) + setPassword (password)</pre>	<pre>+ onConnected (handler) + onDisconnected (handler) + onConnectionLost (handler)</pre>
<pre>+ filesystem () : Filesystem + application () : Application</pre>	<pre>+ sendEvent(event) : bool</pre>
<pre>+ installModule (id, smodule) : bool + uninstallModule (id) : bool + hasModule (id) : bool + getModule (id) : SModule + sendModuleMessage(receiver_id, msg, sender_id)</pre>	

Abbildung 6.6: Die wichtigsten Klassen des Second Screen Managers setzen sich aus der SSMA, dem SModule und dem PrimaryScreen zusammen.

Skriptsprache einbinden

Nachdem in der *xis-engine* noch keine Skriptsprache inkludiert war, musste eine Skriptsprache gefunden werden, welche für die Entwicklung der Subapps herangezogen werden kann. Dazu wurde, die vor allem in der Videospielprogrammierung bekannte Skriptsprache, *Lua* verwendet (s. auch Abschn. 6.2.3). Anfangs funktionierte die Einbindung weitgehend problemlos, zumal die verwendete Bibliothek aus reinem C-Code besteht und für eine plattformunabhängige Verwendung ausgelegt ist. Jedoch war die Erstellung der Schnittstellen zwischen der C++ *xis-engine* und der Skriptsprache Lua aufwendig. Für jede C++ Funktion, welche in Lua zur Verfügung stehen sollte, mussten mindestens drei C-Befehle der Lua-Bibliothek implementiert werden. Dabei ist anzumerken, dass die Programmierung von benötigten Schnittstellen, zwischen nativem Code und Skriptcode, für jede einzubindende Skriptsprache einen Mehraufwand darstellt.

Zum Verknüpfen der beiden Programmiersprachen wurden Teile aus dem *Lua-Wrapper* von *Gameplay*⁷ herangezogen und für die *xis-engine* modifiziert. Dieser erleichterte das Verbinden von C++ Klassen mit *Lua-Tables* und umgekehrt, da Lua nur C-Funktionen aufrufen kann.

SSMA

Die Klasse **SSMA** (Second Screen Manager App) ist die Hauptschnittstelle der mobilen App (s. Abb. 6.6). Sie ist von `xis::Application` abgeleitet

⁷Gameplay ist eine plattformunabhängige Game-Engine. <http://gameplay3d.org>

Programm 6.1: Die Verbindungsanfrage wird am SSM in der Klasse **Connector** entgegengenommen, validiert und beantwortet.

```

1 void Connector::accept(ConnectEvent c)
2 {
3     NetEventConnection* nec = c.getConnection();
4
5     // Überprüfen ob noch eine TCP Verbindung besteht
6     if (nec == NULL) return;
7
8     // Passwortüberprüfung und
9     // Validierung der Identifikation des Hauptspiels
10    if (!(validatePassword(m_password, c.getPassword()) &&
11        validateAppInfo(c.getAppInfo())) )
12    {
13        c.respond( AcceptEvent(false, "unauthorized") );
14    }
15    else // die Hauptverbindung kann akzeptiert werden:
16    {
17        xis_string key = generateSessionKey(c.getAppInfo(), std::rand());
18        PrimaryScreen* ps = new PrimaryScreen(c.getAppInfo(), nec);
19
20        ...
21
22        // Adaptieren der Firewall - NetEventRestriction
23        m_eventService->restriction().acceptNetEventsFrom(c.getEndpoint());
24
25        // Antworten mit dem Accept Event
26        c.respond( AcceptEvent(true, key, ps->isInstalled()) );
27    }
28 }

```

und existiert solange die App lebt. Die **SSMA** stellt einen **Connector** bereit, welcher sich um die Annahme der Verbindungsanfrage des SDKs kümmert.

Connector

Die Klasse **Connector** wartet auf **Connect**-Events des SDKs und evaluiert diese, solange noch keine Hauptverbindung etabliert wurde.

Ist ein **Connect**-Event eingetroffen, wird das darin enthaltene Passwort mit dem, vom Benutzer definierten, Passwort abgeglichen. Stimmt dieses überein und kann die Identifikation des Hauptspiels mit dem Lizenzschlüssel validiert werden (s. Abschn. 5.4.3), wird die Verbindung angenommen.

Daraufhin wird eine Instanz von **Accept**-Event erzeugt und mit der Funktion **respond** des **Connect**-Events geantwortet (s. Prog. 6.1). Wurde die Anfrage akzeptiert gilt die Hauptverbindung als etabliert und das Hauptspiel kann den SSM mit weiteren Events steuern (s. Abschn. 7.2).

hörigen zweiten Teil dar. Beide, nochmals erwähnt, bilden gemeinsam eine *Extension*. Die Kommunikation zwischen den Modulen wurde auf den Austausch von Textnachrichten heruntergebrochen. Jedoch kann eine Kommunikation auch über das **NetEventService** vorgenommen werden. Hierbei gilt zu unterscheiden, dass eine Textnachricht direkt an ein Modul versendet wird, aber ein **NetEvent** an unterschiedlichen Stellen des Programms behandelt werden kann.

Für die Zuordnung eines **PModule** zu einem **SModule** wird eine eindeutige Identifikation vergeben. D. h. jedes Module verfügt über eine ID mit der festgestellt werden kann, zu welcher *Extension* es gehört.

PModule

Das **PModule** wird in C++ für das Hauptspiel entwickelt. Dieses muss für das Versenden und Empfangen von Textnachrichten am **SecondScreen** installiert werden.

SModule

Das **SModule** wird in Lua für die Subapp entwickelt. Dieses muss für das Versenden und Empfangen von Textnachrichten am **PrimaryScreen** installiert werden. Das Besondere am **SModule** ist, dass es sich hierbei um eine abstrakte C++ Klasse handelt. D. h. es kann sowohl in Lua als auch in C++ von **SModule** abgeleitet werden.

Aktivität einer Extension

Eine *Extension* kann sich in einem aktiven (*engaged*) oder inaktiven (*disengaged*) Zustand befinden. Dabei ist der Zustand abhängig von der Verfügbarkeit beider zugehöriger Module. Nur wenn beide verfügbar sind ist die Extension aktiv (s. Tabelle 6.3). Ein Modul ist verfügbar, wenn

- eine Hauptverbindung besteht,
- das Modul am Proxy installiert wurde und
- wenn es aktiviert ist (**isEnabled**).

Anhand des Zustands der *Extension* können Module feststellen, ob sie mit dem Gegenmodul Nachrichten austauschen können. Wenn sich die Verfügbarkeit eines Moduls ändert wird der Zustand der *Extension* automatisch angepasst. Eine Veränderung wird den Modulen mit den Funktionen **onEngage** und **onDisengage** mitgeteilt.

Zuständig für die Überprüfung der Verfügbarkeit ist der jeweilige Proxy, auf welchem das Modul installiert ist. Wurde ein Modul installiert, wird ein **ConfigModule**-Event zum anderen Proxy übermittelt. Dieses beinhaltet die ID und die aktuelle Verfügbarkeit des Moduls. Ändert sich die Verfügbarkeit,

Tabelle 6.3: Eine *Extension* des Erweiterungssystem ist nur aktiv, sofern sowohl das *PModule* als auch das *SModule* verfügbar (+) sind.

<i>PModule</i>	<i>SModule</i>	<i>Extension</i>
+	+	engaged
–	+	disengaged
+	–	disengaged
–	–	disengaged

während dieses am Proxy installiert ist, wird ein erneuter Event mit der dementsprechenden Einstellung übermittelt.

Ein Modul kann jedoch auch verwendet werden, wenn die *Extension* inaktiv ist, eventuell aber nur in eingeschränktem Maße. Dies kann je nach eigener Implementierung variieren. Beispielsweise könnte ein *SModule* für eine Kartenvisualisierung, während die *Extension* inaktiv ist, dem Benutzer mitteilen, dass die Position der Spielfigur zur Zeit nicht angezeigt werden kann, weil keine Verbindung zum Hauptspiel besteht.

Austausch von Textnachrichten

Ist eine *Extension* aktiv können zwischen den Modulen Textnachrichten ausgetauscht werden. Für das Empfangen einer Nachrichten muss die nachfolgende Funktion in einer abgeleiteten Klasse überschrieben werden:

```
Module::onMessage(message, sender_id) : bool
```

Das Ergebnis der Funktion ist ein boolescher Wert welcher angibt ob das Modul die Nachricht annimmt (**true**) oder zurückweist (**false**). Wird die Nachricht zurückgewiesen, übermittelt der Proxy eine Fehlermeldung an den Absender. Versendet werden die Nachrichten mit:

```
Module::sendMessage(msg : String)
```

Der Einfachheit halber muss dieser Funktion nicht ausdrücklich die ID des Senders mitgegeben werden. Diese wird von der Basisklasse bestimmt. Der Proxy ermöglicht es zudem unter Verwendung der Funktion

```
Proxy::sendModuleMessage(receiver_id, msg, sender_id)
```

von jeder Stelle im Programmcode aus eine Nachricht an ein bestimmtes Modul zu übermitteln. Die einzige Bedingung dafür ist, dass die ID des gewünschten Empfängers bekannt sein muss.

Kapitel 7

Umsetzung der Prototypen

Um die Praxistauglichkeit der Implementierung zu testen, wurden für das vorhandene Spiel *Elements* zwei Second Screen Prototypen entwickelt. In Abschnitt 7.2 wird auf die Verwendung des verworfenen Slave-Frameworks eingegangen, welches sich grundlegend von der finalen Version, dem Subapp-Framework, differenziert. Abschließend wird eine Subapp für *Elements* mithilfe des Subapp-Frameworks erstellt und dessen Installation aufgezeigt.

7.1 Begriffsdefinition

Um die umgesetzten Prototypen differenzieren zu können werden folgende Begriffe verwendet:

Karten-Prototyp wurde mit dem Slave-Framework erstellt und stellt eine Kartenvisualisierung am Second Screen zur Verfügung (s. Abschn. 7.2).

Abenteuer-Prototyp ermöglicht eine Inventaransicht und das Spielen eines Puzzles am Second Screen (s. Abschn. 7.3). Umgesetzt wurde dieser Prototyp mit dem Subapp-Framework. Der Begriff *Abenteuer* wurde gewählt, da u. a. die Spielmechaniken Inventar und Puzzle in Abenteuerspielen eingesetzt werden [14].

7.2 Umsetzung des Karten-Prototyps

Um die entwickelten Frameworks, wie erwähnt, zu testen, wurde jeweils ein Prototyp erstellt welcher das Spiel *Elements* um einen Second Screen erweitert.

Im Folgenden wird näher auf die Funktionsweise des ersten Prototyps eingegangen, in welchem für jedes Level am Second Screen eine Karte dargestellt und die aktuelle Position des Spielers markiert wird. Dafür wurde u. a. das *GUI-System* der *xis-engine* verwendet.

7.2.1 GUI-System

Der Karten-Prototyp verwendet das Slave-Framework, in welchem es möglich ist, nach dem Etablieren der Hauptverbindung eine grafische Benutzeroberfläche des SSMs zu manipulieren. Diese Benutzeroberfläche wurde mit dem bestehenden *GUI-System* der *xis-engine* umgesetzt.

Für die Darstellung des GUIs wird eine Instanz der Klasse **Page** benötigt. Dieser können u. a. verschiedene rechteckige Objekte als Instanzen des Basisobjekts **Component** hinzugefügt und wieder entfernt werden. Unter Verwendung der Klasse **Label** kann Text am Bildschirm ausgegeben werden. Weitere Informationen über das GUI-System können der online Dokumentation¹ der *xis-engine* entnommen werden.

7.2.2 Anpassen der Darstellung unter Einsatz von Events

Für das Anpassen der **Page** wurden die Befehle aus Abschnitt 4.4.1 als Events umgesetzt. Dazu wurde definiert, dass dem Hauptspiel eine Instanz der Klasse **Page** zugeteilt wird. Diese wird angezeigt, sobald der Event **ShowPage** zum ersten Mal einlangt. Empfängt der SSM den besagten Event ein weiteres Mal, wird die vorhandene **Page** überschrieben. Das hat zur Folge, dass alle **Components** der vorherigen **Page** zerstört und neue den Daten entsprechend erstellt werden.

Nachdem das **GUI-System** das Konfigurieren der **Page** und das Erstellen der **Components** anhand einer Zeichenkette unter Einsatz der Funktionen

```
GuiSystem::ConfigurePageByString(page, value)
GuiSystem::CreateComponentByString(value)
```

ermöglicht, kann u. a. **ShowPage** mit nur drei Befehlen von der SSMA behandelt werden:

```
1 void SSMA::onShowPage(ShowPage evt)
2 {
3     m_page->deleteAllChildren();
4     GuiSystem::ConfigurePageByString(m_page, evt.getPage());
5     xis::getService<GuiSystem>()->showPage(m_page);
6 }
7 void SSMA::onAddComponent(AddComponent evt)
8 {
9     Component* c = GuiSystem::CreateComponentByString(evt.getComponent());
10    if (c != std::nullptr) {
11        m_page->addChild(c);
12    }
13 }
```

Dabei ist zu beachten, dass die Zeichenkette in der benötigten Syntax vom Hauptspiel zusammengestellt werden muss.

¹<http://docs.xis-engine.com>

7.2.3 Integration in Elements

Um das GUI des Second Screen Managers für die Kartenvisualisierung in Elements ändern zu können, wurde das SDK des Karten-Frameworks im Spiel eingebunden und im Game-Loop verankert.

Im Karten-Prototyp wurde die Hauptverbindung vorerst automatisch hergestellt, sobald beide Geräte miteinander verbunden waren. Weiters wurde in dieser ersten Version auch das benötigte Passwort unveränderlich im Programmcode vordefiniert, da die Programmierung einer Passwortabfrage in Elements vorerst ausgelassen wurde.

Konnte eine Hauptverbindung etabliert werden, so wurde der Titel des Spiels am SSM-Display angezeigt. Wurde ein Level in Elements gestartet, so wurde die Funktion `displayMap` aufgerufen, welche anhand der Leveldaten für jede Plattform ein `Component` erzeugte. Die Anzahl der Plattformen eines Levels war jedoch zu hoch um mit einem einzigen `ShowPage`-Event versendet zu werden, da die Datenlänge eines Events begrenzt ist. Daher erzeugte `ShowPage` nur eine leere Seite mit dem Titel des Levels, während die Plattformen mit separaten `AddComponent`-Events hinzugefügt wurden.

Anschließend wurde für den Spieler ein weiß eingefärbtes `Component` mit der ID „player“ erzeugt. Während des Spiels wurde die Position dieses `Components` mit dem `TransformComponent`-Event in jedem Zyklus des Game-Loops aktualisiert, sodass es immer die aktuelle Position des Spielers innerhalb des Levels visualisierte (s. Abb. 7.1 (b-c)).

Die Verbindung zum Second Screen konnte durch das Beenden des SSMs oder des Hauptspiels unterbrochen werden.

7.2.4 Zurückkehren zum Hauptmenü

Wurde die Hauptverbindung zwischen dem SSM und Elements unterbrochen, wurde die zugeteilte `Page`-Instanz wieder zerstört und das Hauptmenü am SSM angezeigt. Dieses bestand aus einer Eingabemaske für den Namen und das Passwort des SSMs. Dabei wurde jedes Mal, wenn der Name oder das Passwort verändert wurde, die Erkennungsnachricht versendet. Wurde vom SDK dennoch der SSM nicht gefunden, konnte bewusst mit dem `Scan`-Button eine Erkennung versendet werden.

7.3 Umsetzung des Abenteuer-Prototyps

Nachfolgend wird auf die Umsetzung des zweiten Prototyps eingegangen, welcher Spielmechaniken aus dem Abenteuerspielgenre am Second Screen unter Einsatz des Subapp-Frameworks realisiert. Dazu wurde wie oben auf das bestehende Spiel *Elements* aufgebaut. Weiters wird auf die Programmierung der entworfenen Subapp für Elements eingegangen. Diese demonstriert zwei Second Screen Anwendungen: Sie verfügt über die Möglichkeit das In-

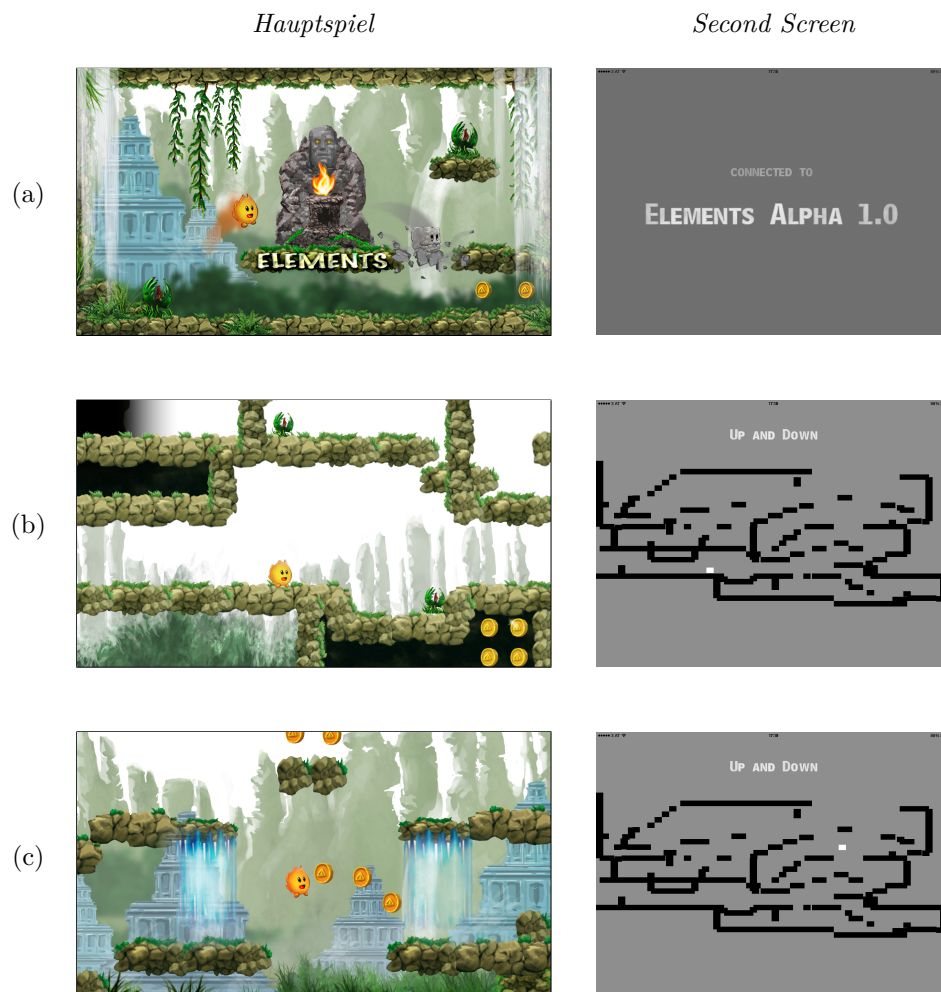


Abbildung 7.1: Mit dem Karten-Prototyp wurde für Elements eine Kartenvisualisierung am Second Screen realisiert. Sobald sich Elements mit dem SSM verbunden hat, wird ein `ShowPage`-Event versendet. Nach Empfangen dieser Nachricht wird auf dem Second Screen bis zum Starten eines Levels der Spieltitel angezeigt (a). Wird ein Level gestartet, zeigt der Second Screen die Übersicht des Levels und markiert mit einem weißen Rechteck die Spielerposition (b). Am Second Screen wird die Spielerposition mit dem `TransformComponent`-Event stetig aktualisiert (c).

ventar eines Spielers darzustellen, sowie über ein Puzzle, bei welchem ein Farbcode in eine Farbmatrix einzugeben ist.

7.3.1 Vorbereitung

In Elements wurde für das Testen der Subapp ein Testlevel zusammengestellt. Darin wurden für das Inventar unterschiedliche Gegenstände zum

Sammeln platziert. Der Weg zu diesen Gegenständen ist durch einen violetten Stein blockiert. Bei diesem Stein gilt es eine richtige Farbkonstellation am Second Screen einzugeben um ihn zu entfernen. Anschließend können die Gegenstände gesammelt werden. Die Abschnitte 4.2.1 und 4.2.2 behandeln die zugehörigen Szenarien, welche mit dieser Subapp nachempfunden wurden.

7.3.2 Aufbau der Szene

Für das Darstellen des Inventars und der Farbmatrix wurde eine Szene in Lua entwickelt. Diese Szene besteht aus einem Verzeichnis im Ressourcenbündel „`scenes/inventory.scene`“, in welchem folgende drei Dateien abgelegt wurden:

main.lua: Das `main.lua` Skript ist das Hauptskript der Szene und wurde in Lua entwickelt. Darin wurden die drei Schnittstellen `setup`, `update` und `finish` implementiert, welche vom Second Screen Manager aufgerufen werden (s. auch Abschn. 4.5.2).

resource.list: Im selben Verzeichnis befindet sich `resource.list`, welche die Liste der Ressourcen beinhaltet, welche vor dem Anzeigen der Szene geladen werden.

page.mdl: Unter der Verwendung des *GUI-Systems* der *xis-engine* wurde der Szene eine `Page` zur Verfügung gestellt, welche in der `page.mdl` vorkonfiguriert wurde. Dazu wurde die Funktion `ConfigurePageByString` benutzt, welche im Karten-Prototyp für das Behandeln des `ShowPage`-Events verwendet wurde (s. Abschn. 7.2.2).

7.3.3 Inventar

Im Inventar des Second Screens werden alle Gegenstände angezeigt, welche vom Spieler im Testlevel gesammelt wurden (s. Abb. 7.2). Dazu wird vom `InorySystem` der *xis-engine* die Klasse `xis::inory::Inventory` verwendet und als Instanz der Spielfigur von `Elements` zugewiesen. Gesammelte Gegenstände des Testlevels werden dieser Instanz hinzugefügt.

Die Darstellung des Inventars am Second Screen wurde als *Extension* entwickelt, in welcher das `PModule` Änderungen des Inventars wahrnimmt und an das `SModule` der Subapp meldet.

PModule

Wird eine Hauptverbindung im Verbindungsmenü hergestellt, installiert `Elements` das `PModule` und setzt dieses mit `setEnabled(false)` inaktiv. Erst wenn ein Level startet, wird das `PModule` aktiviert.

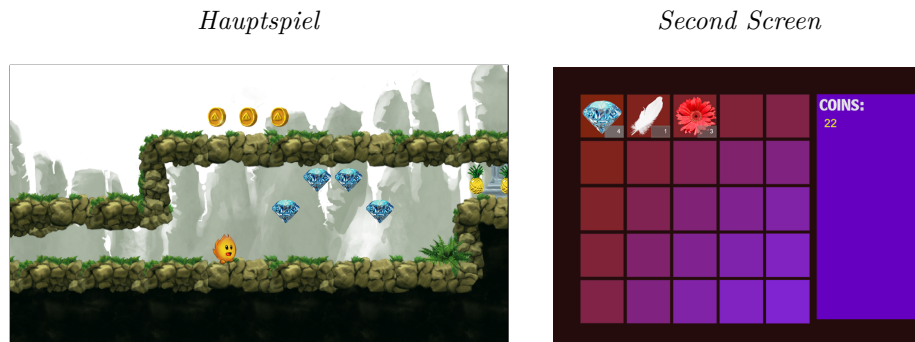


Abbildung 7.2: Gesammelte Gegenstände in Elements werden am Second Screen dargestellt. Die Menge an gesammelten Münzen wird in einem separaten Label rechts angezeigt.

SModule

Das entwickelte SModule wird zum Zeitpunkt von `setup` des Hauptskripts am Proxy PrimaryScreen installiert:

```
-- Erzeugung der Lua-Instanz des SModuls
local inventoryModule = xis.secco.InventoryModule.Secondary()
-- SModule am PrimaryScreen installieren
local primary = App:getPrimaryScreen()
local result = primary:installModule("extension-id", inventoryModule)
```

Während das PModule nicht verfügbar ist, wird am Second Screen ein Text dargestellt, welcher mitteilt, dass sich Elements im Menü befindet und das Inventar erst während des Levels dargestellt wird.

Die Ansicht des Inventars wird vom SModule unter Einsatz des *GUI-Systems* erstellt. Für jeden Gegenstand wird ein **Component** erzeugt und der **Page** der Szene hinzugefügt. Ein **Component** verfügt über ein Bild und eine Zahl für die Mengenangabe.

Aktualisieren der Daten

Verändert sich das Inventar in Elements, sendet das PModule das gesamte Inventar als Text zum SModule. Dieses empfängt die Textnachricht und aktualisiert anhand der neuen Daten die Einträge der **Page** (s. Prog. 7.1).

7.3.4 Puzzle

Wie oben erwähnt, sind im Testlevel violette Steine platziert, welche sich nur durch das Lösen eines Puzzles am Second Screen entfernen lassen. Kommt die Spielfigur vor einem dieser Steine zu stehen, wird eine 4×4 Farbmatrix am Second Screen angezeigt (s. Abb. 7.3).

Programm 7.1: Das SModule der Inventar-Erweiterung empfängt eine Liste der Gegenstände, welche sich im Inventar des Hauptspiels befinden und aktualisiert die Einträge am Second Screen.

```

1 function onMessage(msg, sender)
2   if (sender == "inventory-pmodule") then
3     local inventory = Inventory:fromString(msg)
4     -- getItem liefert alle Gegenstände in einem assoziativen Array
5     local items = inventory:getItems()
6     local i = 1
7     -- Aktualisierung aller Einträge im Verbindungsmenü
8     for item,amount in pairs(items) do
9       local entry = getPage():getItemEntry(i)
10      entry.image:setImage(getImage(item))
11      entry.number:setText(tostring(amount))
12      i = i + 1
13    end
14    return true
15  else
16    return false
17  end
18 end

```

Die Farbe eines Elements der Matrix kann durch Toucheingabe verändert werden. Dabei werden die Farben Rot, Gelb, Grün und Blau durchgewechselt.

Puzzle-Erweiterung

Programmiert wurde das Puzzle, wie auch das Inventar, als *Extension*.

Das PModule empfängt Kollisionen zwischen der Spielfigur und den violetten Steinen der Elements-Spiellogik. Dabei wird zwischen dem *Beginn* und dem *Ende* der Kollision unterschieden. Empfängt das PModule einen *Beginn*, wird der Befehl zum Öffnen der Farbmatrix als Nachricht „open“ an das SModule gesendet. Das SModule fügt daraufhin ein vollflächiges Component der Page der Szene hinzu. In diesem werden in einer Matrix 4×4 Farbelemente angezeigt. Die Farbelemente wurden als Klasse *Piece* in Lua entwickelt und verwenden ein Component für die Darstellung und für das Touchfeedback. Wird ein Farbelement berührt wechselt dessen Farbe und die neue Farbkonstellation wird an das PModule versendet.

Stimmt die eingegebene Konstellation, entfernt das PModule den violetten Stein von der Spielwelt und die Farbmatrix wird wieder geschlossen. Verlässt der Spieler den Stein bevor die richtige Konstellation eingegeben wurde, wird die Nachricht „close“ für das Schließen der Farbmatrix an das SModule versendet.

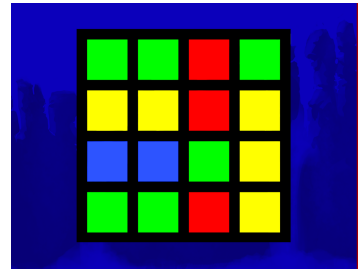
Hauptspiel*Second Screen*

Abbildung 7.3: Befindet sich die Spielfigur in Elements vor dem violetten Stein, öffnet sich am Second Screen eine Farbmatrix. Wird der richtige Farbcode eingegeben, wird der Stein entfernt und die Spielfigur kann passieren.

7.4 Installation der Abenteuer-Subapp

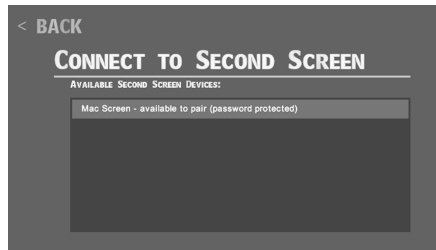
Im Subapp-Framework wurde die Installation von Subapps und deren Ausführen am Second Screen Manager ermöglicht. Anhand des entwickelten Abenteuer-Prototyps wird nachfolgend die umgesetzte Installationslösung demonstriert. Dabei wird auf die Verwendung des SDKs genauer eingegangen.

7.4.1 Vorbereitung

In Elements wurde das neue SDK des Subapp-Frameworks integriert. Für die Installation wurde die entwickelte Subapp für Elements (s. Abschn. 7.3) herangezogen und als Zip-Archiv im Ressourcen-Ordner von Elements abgelegt.

Um sich mit dem Second Screen Manager zu verbinden, wurde in Elements ein eigenes *Verbindungs Menü* entwickelt. Dieses Menü benutzt die Schnittstellen des SDKs und stellt für das Abfragen eines Passworts ein Dialogfenster bereit. Wird das *Verbindungs Menü* geöffnet, wird vom **Secco-Service** mit der Funktion `getSecondScreens` abgefragt, welche Second Screen Manager bereits erkannt wurden, um diese in einer Liste anzuzeigen. Erkennt das SDK einen neuen Second Screen Manager, während das Menü offen ist, wird diese Information dem *Verbindungs Menü* über den Callback `onSecondScreenAvailable` mitgeteilt. Anschließend wird ein neuer Eintrag im *Verbindungs Menü* erstellt. Wurde zum Zeitpunkt des Öffnens des Menüs kein SSM erkannt, wird dem Benutzer mitgeteilt, dass er einen Second Screen Manager auf seinem mobilen Gerät starten möge.

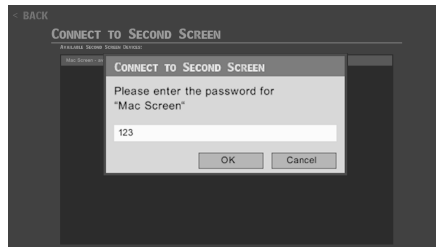
Konnte ein Second Screen Manager gefunden werden, wird ein Eintrag mit dem Namen des Second Screen Managers im *Verbindungs Menü* dargestellt (s. Abb. 7.4 (a)). Mit einem Klick auf diesen Eintrag kann der Benutzer

*Hauptspiel**Second Screen*

(a)



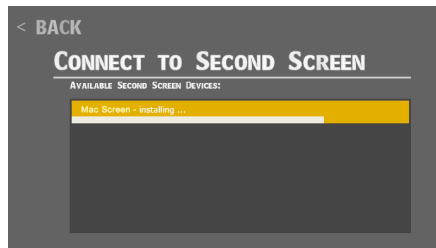
(b)



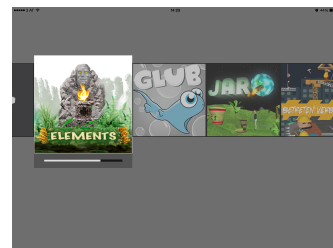
(c)



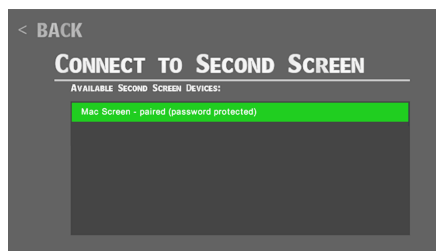
(d)



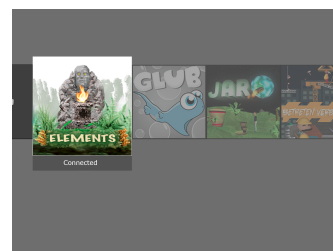
(e)



(f)



(g)



(h)

Abbildung 7.4: Die Grafik zeigt die Installation der Elements-Subapp in vier Schritten. Elements findet einen Second Screen Manager im Netzwerk und zeigt diesen im *Verbindungs*menü an (a). Wird versucht eine Verbindung aufzubauen, muss ein Passwort eingegeben werden (c). Konnte das Passwort vom SSM verifiziert werden, wird die Subapp installiert. Der Fortschritt der Übertragung ist als Ladebalken ersichtlich (e). War die Installation erfolgreich wird abschließend im *Verbindungs*menü der SSM grün hinterlegt (g).

einen Verbindungsversuch zum SSM einleiten. Daraufhin wird ein Feld zur Passworteingabe angezeigt (s. Abb. 7.4 (c)), sofern ein Passwort benötigt wird. Das Passwort zum Verbinden findet der Benutzer auf der Einstellungsseite des SSMs (s. Abb. 7.4 (d)), wo dieses auch verändert werden kann. Die Einstellungen ermöglichen auch das Definieren eines Namens für den Second Screen Manager. Würde der Benutzer diesen am SSM ändern, würde im *Verbindungsmenü* der Name aktualisiert werden.

7.4.2 Durchführung

Konnte das Passwort korrekt vom Benutzer eingegeben werden, startet Elements automatisch den Installationsvorgang mittels SDK, falls dieser notwendig wäre. Prog. 7.2 zeigt, das Initiieren der Installation aus der Sicht von Elements. Die `FileTransfer`-Instanz ermöglicht das Verfolgen des Fortschritts des Installationsvorgangs indem, wie in Zeile 36 des Programms ersichtlich, der Funktion `onProgressChanged` ein Callback mitgegeben wird. Dadurch konnte ein Ladeindikator umgesetzt werden (s. Abb. 7.4 (e)). Das Ende der Installation wird mit `onTransferCompleted` festgestellt. Steht eine Verbindung zum SSM und war die Installation erfolgreich wird der Eintrag des SSMs im *Verbindungsmenü* grün dargestellt (s. Abb. 7.4 (g)).

Programm 7.2: Das Programmbeispiel zeigt einen Ausschnitt aus dem *Verbindungs*menü von Elements, welches für die Installation der Subapp verantwortlich ist.

```
1 void onAccept(AcceptEvent evt)
2 {
3     // Den Eintrag des Second Screens vom Verbindungs Menü holen
4     // und den Text aktualisieren
5     Entry* e = getEntry(evt.getSecondScreen().getId());
6     e->property<Boolean>("connected").value(true);
7     updateEntry(e);
8     if (evt.isInstallationRequired())
9     {
10         e->setBackgroundColor(Color::orange());
11         installSubapp(evt.getSecondScreen());
12     } else {
13         e->setBackgroundColor(Color::green());
14     }
15 }
16
17 void installSubapp(SecondScreen* secco)
18 {
19     Path subapp = "res/subapp.zip";
20
21     FileTransfer::StatusCode status;
22     FileTransfer* ft = secco->uploadResourceFile(
23         SecondScreen::INSTALL_PACKAGE, subapp, status);
24
25     // Überprüfen ob die angegebene Datei übertragen werden kann
26     if ( status == FileTransfer::STATUS_FILE_FAILED_TO_OPEN
27         || status == FileTransfer::STATUS_FILE_NOT_FOUND
28         || status == FileTransfer::STATUS_FILE_FAILED_TO_READ)
29     {
30         logError("Ressourcenbündel konnte nicht geladen werden.");
31     }
32     // Überprüfen ob eine Übertragung möglich ist
33     else if (ft != std::nullptr)
34     {
35         ft->onTransferStarted(uploadResourceStarted);
36         ft->onProgressChanged(uploadResourceProgressChanged);
37         ft->onTransferInterrupted(uploadResourceInterrupted);
38         ft->onTransferCompleted(uploadResourceComplete);
39     }
40     else // Keine Übertragung möglich
41     {
42         logError("Ressourcenbündel konnte nicht versendet werden.");
43     }
44 }
```

Kapitel 8

Evaluierung

Anhand des letzten Kapitels wurde die Praxistauglichkeit der entworfenen Frameworks festgestellt. Zudem wurden in Abschnitt 4.3 Anforderungen gestellt, welche zu erfüllen waren. Diese werden in diesem Kapitel in Bezug auf das finale Subapp-Framework kritisch betrachtet und es wird versucht aufzuzeigen, inwiefern diese erfüllt wurden.

8.1 Universelle Second Screen App

Zu den funktionalen Anforderungen zählte die Entwicklung einer universellen Second Screen App, welche für jedes Hauptspiel als Second Screen fungieren kann (s. auch Abschn. 4.3.1).

Dies wurde durch die Entwicklung des Second Screen Managers (SSM) erreicht. Der SSM ist eine plattformunabhängige mobile App, in welche sogenannte Subapps (s. Abschn. 4.5) installiert werden können. Die Entwicklung einer Subapp ermöglicht das Entwerfen eines Second Screens für ein beliebiges Hauptspiel.

8.1.1 Umsetzung einer Subapp

Das Subapp-Framework wurde mit dem Abenteuer-Prototyp für *Elements* auf seine Praxistauglichkeit getestet. Die Erstellung der benötigten Subapp konnte, dank des flexiblen Frameworks, problemlos innerhalb weniger Stunden bewerkstelligt werden. Das vorhandene Erweiterungssystem bot ideale Bedingungen um mit wenigen Codezeilen am Second Screen ein Inventar und eine Farbmatrix darzustellen. Gerade der Austausch von Textnachrichten zwischen den Modulen vereinfachte die Kommunikation.

Weiters zeigte sich in der Entwicklung der Subapp, dass die Wahl, die Logik mittels interpretierter Skriptsprache auszuführen, vorteilhaft war. Die Subapp konnte mit einem Second Screen Manager, welcher unter MacOS X ausgeführt wurde, laufend getestet werden ohne stetig den Programmcode

neu auf einem mobilen SSM installieren zu müssen. Der anschließend entstandene Luacode konnte eins zu eins auf den mobilen Second Screen Manager installiert werden, ohne dass eine plattformspezifische Kompilierung oder Abänderung des Codes erforderlich gewesen wäre.

Die Subapp beinhaltet zudem alle benötigten Ressourcen für die Darstellung von Bildern sowie für die Audioausgabe. Diese Dateien mussten nicht von einer externen Quelle über das Netzwerk geladen werden und standen dadurch umgehend zur Verfügung.

8.1.2 Verbindungsmenü

Ein zusätzlicher Mehraufwand war, wie erwartet, die Erstellung des *Verbindungsmenüs*. Nachdem die Game-Engine für das Framework unbekannt ist, wurden vom SDK nur Schnittstellen für die Kommunikation mit dem Second Screen Manager angeboten. Die Verwendung des asynchronen Eventprotokolls stellte jedoch einen bei Weitem geringeren Aufwand dar, als die Entwicklung der grafischen Benutzeroberfläche des *Verbindungsmenüs*, welche dem Benutzer das Verbinden mit dem Second Screen ermöglicht. Um diesen Mehraufwand für Spielehersteller zu reduzieren, könnte für häufig benutzte Game-Engines ein *Verbindungsmenü* vorprogrammiert werden, welches sich einfach individualisieren lässt. Anhand der Integration des Frameworks in *Elements* konnten keine fehlenden Schnittstellen des SDKs festgestellt werden. Dennoch sollte das SDK noch mit weiteren Game-Engines getestet werden, um Verbesserungsmöglichkeiten der Schnittstellen zu eruieren. Für das Inventar und die Farbmatrix sind zwei *Extensions* entstanden. Es wäre möglich, noch weitere solche *Extensions* zu entwickeln und diese, beispielsweise in einer Community, frei zugänglich zu machen. Dadurch könnte der Programmcode auch in anderen Spielen wiederverwendet werden.

8.1.3 Spieldaten und Ressourcen der Subapp

Eine Subapp verwaltet ihre abgelegten Dateien selbstständig. Bei der Deinstallation werden diese zerstört. Dadurch können für den Benutzer relevante Spieldaten verloren gehen, welche noch nicht ins Hauptspiel übertragen wurden. Abhilfe könnte eine Abfrage vor der Deinstallation schaffen, in der dem Benutzer die Möglichkeit offengelassen wird, gespeicherte Daten aufzubewahren und nur die Ressourcen der Subapp zu löschen.

Ein weiterer Nachteil könnte das Speichern aller von der Subapp benötigten Ressourcen am mobilen Gerät sein. Nachdem die Speicherkapazität eines mobilen Geräts im Vergleich zu einem PC oder einer Konsole relativ gering ist, können Subapps, welche beispielsweise eine Vielzahl an hochauflösenden Bilddateien beinhalten, verhältnismäßig viel Speicherplatz einnehmen. Dabei wird jeweils nur eine Szene zur selben Zeit verwendet und die restlichen Ressourcen werden nicht gebraucht. Hierbei könnte eine Möglichkeit geschaf-

fen werden, nur die benötigten Ressourcen der aktuellen Szenen am Second Screen zu installieren. Dadurch müssten jedoch Daten unter Umständen des Öffern vom Hauptspiel auf den Second Screen nachgeladen werden.

Mit einer Subapp, welche in einer interpretierten Skriptsprache entwickelt wird, kann die Leistung einer Programmiersprache, wie z. B. C++ oder Objective-C, nicht erreicht werden. Daher sollten rechenintensive Operationen am Hauptspiel durchgeführt werden. Deren Ergebnisse könnten anschließend zur Subapp übertragen werden.

8.2 App kann vom Hauptspiel gesteuert werden

Es galt die Second Screen App so zu gestalten, dass diese von einem Hauptspiel gesteuert werden kann. Dabei sollten die Anzeige- und Interaktionsmöglichkeiten der App beeinflussbar sein. Dies wurde im Subapp-Framework durch die Möglichkeit zur Entwicklung einer eigenen Subapp, welche für die grafische Benutzeroberfläche verantwortlich ist, erreicht. Unter Einsatz des `NetEventServices` ist die Steuerung der Benutzeroberfläche durch ein Hauptspiel nun möglich (s. Abschn. 6.3.1).

Dafür muss das Hauptspiel eine Subapp am Second Screen Manager installieren, mit welcher im Anschluss selbst definierte Befehle ausgetauscht werden können. Der Abenteuer-Prototyp demonstriert dies durch die Darstellung eines Inventars (s. Abschn. 7.3.3). Hierbei wird die Anzahl der Gegenstände des Inventars zur Subapp gesendet und in dessen grafischer Benutzeroberfläche dargestellt.

Weiters zeigt das Puzzle des Abenteuer-Prototyps, dass sich Subapp und Hauptspiel gegenseitig beeinflussen können (s. Abschn. 7.3.4). Dabei veranlasst das Hauptspiel das Öffnen des Puzzles am Second Screen. Wird das Puzzle am Second Screen gelöst, wird im Gegenzug ein Spielobjekt am Hauptspiel zerstört.

8.2.1 NetEventService

Durch die Verwendung des `NetEventService` konnte die Netzwerkkommunikation zwischen SDK und SSM vorteilhaft asynchron abstrahiert werden. Die zugrunde liegende TCP-Verbindung wird von der Klasse `NetEventConnection` verdeckt, sodass von Spieleentwicklern keine Aufrufe der Netzerkbibliothek *Boost.Asio* benötigt werden. Zudem ist das Konzept einer Ereignis-gesteuerten Architektur in der Spieleprogrammierung weit verbreitet [4, S. 773–793][47, 9].

Die Verwendung eines eigenen Threads für das Senden und Empfangen der Events ermöglicht ein ruckelfreies Spielvergnügen. Das demonstriert der Karten-Prototyp, in welchem in jedem Game-Loop-Schritt ein Event asynchron zum Second Screen Manager versendet wurde um die Position der Spielfigur in der Kartenvisualisierung zu aktualisieren (s. Abschn. 7.2).

Das Erstellen neuer Befehle und Ereignisse mittels Events ist ohne großen Aufwand möglich. Dabei reicht es wenn von `NetEvent` abgeleitet wird und die benötigten Daten in eine Zeichenkette umgewandelt werden (s. Abschn. 6.3.1).

8.2.2 Schwierigkeiten einer asynchronen Kommunikation

Das `NetEventService` arbeitet durchgängig asynchron, daher ist keine synchrone Netzwerkkommunikation über die Hauptverbindung möglich. Dies kann das Finden von Fehlern zwischen dem Hauptspiel und der Subapp erschweren, da kein sequenzieller Pfad an Funktionsaufrufen nach dem Versenden eines Events bis zu dessen Ergebnis dargestellt werden kann. Wird eine Verbindungsanfrage versendet, ist der Zeitpunkt des Eintreffens der Antwort nicht definiert und von der Geschwindigkeit des Netzwerkes und des Second Screen Managers abhängig.

Weiters benötigt das Versenden eines Events mehr Bandbreite, als wenn nur ein Schlüsselwort als Befehl übertragen wird. Ein kontinuierliches Versenden von übermäßig vielen Events in einem Game-Loop könnte zur Folge haben, dass der Second Screen Manager mit dem Abarbeiten der empfangenen Events nicht nachkommt. Ein Event setzt sich aus dem Schlüsselwort `event`, seinem Typ und den zu übertragenden Daten zusammen. Letztere werden als Zeichenkette übertragen, was den Nachteil hat, dass die Datenmenge bei der Übertragung von numerischen Datentypen enorm steigt: Ein Integer benötigt je nach Betriebssystem vier oder acht Bytes. Wird ein Integer in eine Zeichenkette umgewandelt, benötigt jede Ziffer der Zahl ein Byte. D. h. wird ein Integer-Wert übertragen, welcher mehr als vier/acht Zeichen benötigt, fallen für die Zeichenkette je zusätzliche Ziffer ein Byte mehr an.

8.3 Unabhängig verwendbar

Durch den Einsatz von Subapps wurde es im Subapp-Framework ermöglicht, eine unabhängige Second Screen Anwendung umzusetzen. Diese Subapps können von Spieleentwicklern in Lua programmiert und vom SSM ausgeführt werden, ohne dass eine Verbindung zu einem Hauptspiel besteht.

Ein konkreter Prototyp wurde dafür noch nicht umgesetzt, jedoch zeigt der Abenteuer-Prototyp (s. Abschn. 7.3) die Umsetzbarkeit einer selbst agierenden Subapp. Diese könnte auch verwendet werden, wenn kein Hauptspiel besteht. Jedoch würden im Inventar des Prototyps keine Gegenstände dargestellt werden.

8.4 Plattformunabhängiges Framework

Der Second Screen Manager konnte für einen plattformunabhängigen Einsatz entwickelt werden. Dabei zeigte sich die Verwendung von C++ als hilfreich, da diese Sprache u. a. von den derzeit drei verbreitetsten mobilen Plattformen Android, iOS und Windows Phone 8 [42], unterstützt wird. Hierbei ist anzumerken, dass die *xis-engine* momentan noch nicht unter Windows Phone 8 verwendet werden kann. Jedoch wird davon ausgegangen, dass wenn Windows Phone 8 von der *xis-engine* in Zukunft unterstützt wird, wenige Adaptionen des bereits entwickelten Programmcodes für den SSM vorgenommen werden müssen. Getestet wurde der SSM erfolgreich mit dem Karten-Prototyp (s. Abschn. 7.2) und dem Abenteuer-Prototyp (s. Abschn. 7.3) unter iOS auf einem iPad Air.

Weiters wurde das SDK plattformunabhängig implementiert. Hierbei wurde auch auf C++ zurückgegriffen. Daher kann hier eine Vielzahl an Plattformen abgedeckt werden: Windows, MacOS X, Linux, PlayStation 3 u. 4 sowie Xbox One u. 360. Im Zuge der Arbeit konnte das SDK noch nicht für Spielkonsolen getestet werden – u. a. wird für das Entwickeln von PlayStation Anwendungen eine Entwicklerlizenz benötigt, welche kostenpflichtig zu erwerben ist.

8.5 Sicherheit des Second Screen Managers

Bezüglich der Sicherheit des Second Screen Managers mussten einige unterschiedliche Dinge beachtet werden, auf welche im Weiteren näher eingegangen wird.

8.5.1 Unbefugte Installationen von Subapps vermeiden

Es wurde am Second Screen Manager gewährleistet, dass Unbefugte ohne Kenntnis des Passworts keine Subapps installieren und verwenden können (s. Abschn. 5.4.2). Jedoch kann nicht ausgeschlossen werden, dass Unbefugte den Zugang zum Passwort erhalten, da die Netzwerkkommunikation zwischen Hauptspiel und SSM aufgrund der unverschlüsselten Übertragung mitgelesen werden kann. Diese müsste mit dem SSL-Protokoll oder einem dem entsprechenden sicheren Protokoll abgesichert werden. Allerdings wäre im Rahmen dieser Arbeit die praktische Umsetzung einer verschlüsselten Kommunikation nicht realistisch gewesen. Dafür hätte eine Bibliothek, welche plattformunabhängig mit *Boost.Asio* am SSM und in der SDK zusammenspielt, gefunden und in das Framework integriert werden müssen. Angeboten hätte sich dafür die freie Bibliothek *OpenSSL*, welche allerdings noch nicht genauer untersucht wurde.

8.5.2 Spieldaten der Subapp schützen

Wie Abschnitt 4.5.5 zeigt, wurde auch ein Konzept entwickelt, welches es vorsieht Spieldaten des *privaten Verzeichnisses* verschlüsselt abzulegen, um diese vor unbefugtem Zugriff zu schützen. Unter Verwendung der zuvor angesprochenen *OpenSSL* Bibliothek könnte dies als einer der nächsten Schritte im Subapp-Framework umgesetzt werden.

8.5.3 Schutz der privaten Daten des mobilen Geräts

Die Sicherheit der persönlichen Daten des mobilen Geräts, auf welchem der Second Screen Manager ausgeführt wird, kann zum jetzigen Zeitpunkt gegeben werden, da keine Schnittstellen in Lua entwickelt wurden, welche das Lesen von z. B. Kontaktdaten, Kalendereinträgen oder Telefonprotokollen ermöglichen. Diese Daten hätten unbefugt von einer Subapp ausgelesen und an ein Hauptspiel übertragen werden können und hätten dadurch ein Sicherheitsproblem dargestellt. Zudem sind Bilder, Dokumente und andere Dateien, welche sich außerhalb des Second Screen Managers befinden unter iOS vor unbefugten Zugriffen sicher, da die iOS App an sich in einer Sandbox ausgeführt wird [33]. Bei Ausführung des SSMs auf einem Gerät, bei welchem dieses Sandbox-Konzept nicht besteht, sind diese Dateien im Subapp-Framework jedoch noch nicht abgesichert.

Zum Beispiel ist es unter dem Betriebssystem Android mit der Berechtigung *USB-Speicherinhalte ändern oder löschen* möglich vom gesamten Gerätespeicher Daten zu lesen und solche zu schreiben. Diese Berechtigung würde u. a. vom SSM benötigt werden um unter Android ausgeführt werden zu können. Dadurch erhält eine Subapp in der Lua-Umgebung implizit Lese- und Schreibrechte am gesamten mobilen Gerät. Dies gilt es zu unterbinden, da, wie das Konzept vorschlägt, die Subapp nur Schreib- und Lesebefehle innerhalb der definierten Sandbox (s. Abschn. 4.5.6) durchführen darf. Ansonsten könnten mit einer Subapp, wie oben erwähnt, Daten ausspioniert und an das Hauptspiel gesendet werden. Diese Absicherung konnten im Zuge der Subapp-Framework-Entwicklung noch nicht durchgeführt werden, da der Test, in welchem alle erdenklichen Variationen wie in Lua eine Datei gelesen oder geschrieben werden kann abgedeckt werden müsste, den Arbeitsumfang gesprengt hätte.

8.6 Entwicklerfreiheit

Die nicht funktionale Anforderung, den Entwickler im Entwerfen von Second Screen Anwendungen nicht einzuschränken, konnten für die folgenden Fälle ermöglicht werden.

8.6.1 Freie Wahl der Game-Engine

Dem Entwickler wurde offengehalten, welche Game-Engine das SDK benutzt. Dadurch wurde verhindert, dass sein Hauptspiel eine bestimmte Game-Engine verwenden muss um das Subapp-Framework verwenden zu können. Eingeschränkt wurde der Entwickler in der Wahl der Programmiersprache, da nur ein SDK in C++ entwickelt wurde. Um weitere Sprachen zu unterstützen müsste für jede eigens ein SDK mit der jeweiligen Sprache programmiert werden.

8.6.2 Ermöglichen neuer Spielmechaniken

Der Implementierung einer Subapp wurden weitestgehend keine Grenzen gesetzt. Die Verwendung einer Skriptsprache ermöglicht zudem eine Vielzahl an Spielmechaniken für den Second Screen umzusetzen, welche zum Zeitpunkt der Konzeption des Subapp-Frameworks noch nicht entworfen wurden. Dies wurde mit dem Abenteuer-Prototyp gezeigt. Es konnte sowohl ein Inventar, als auch ein Puzzle mit Lua umgesetzt werden, ohne dass für deren konkrete Logik vorab im Second Screen Manager Funktionalitäten hinzugefügt werden mussten. Daher ist es denkbar, dass das Minispiel von Abschnitt 4.2.3 als Subapp programmiert werden kann.

8.6.3 Freiheiten in der Darstellung der Subapp

In der Wahl des Grafikdesigns ist der Entwickler zur Zeit noch limitiert auf 2D Grafiken. Im Subapp-Framework wurden für die Subapp keine Schnittstellen in Lua implementiert, welche das Rendern von 3D-Objekten ermöglichen. Dafür wurden dem Spielehersteller in der Entwicklung von 2D Anwendungen unter Verwendung des *GUI-Systems*, dessen Schnittstellen umfangreich in Lua zur Verfügung stehen, fast keine Grenzen gesetzt.

8.6.4 Offenheit der Anwendungsmöglichkeit

In Kapitel 3 wurden unterschiedliche Verwendungsmöglichkeiten eines Second Screens in Kategorien zusammengefasst. Einen Überblick über diese Kategorien bietet die Tabelle 3.1. Nachfolgend wird aufgezeigt, dass Spieleproduzenten mit dem umgesetzten Framework die Möglichkeit erhalten für jede Kategorie eine Second Screen Anwendung zu entwickeln.

Aktive Interaktion

Wie Abschnitt 3.3 aufzeigt, können unterschiedlichste Anwendungen eine aktive Interaktion benötigen. Der Abenteuer-Prototyp zeigt mit dem Entwurf eines Puzzles (s. Abschn. 7.3.4), dass mit dem Subapp-Framework aktive Second Screen Anwendungen umgesetzt werden können.

Weiters verwendet der Second Screen Manager das *GUI-System* um Eingabeereignisse zu erhalten. Diese Ereignisse werden der Subapp unter Einsatz von Callbacks in die Lua-Umgebung weitergereicht. D. h. es kann auf Interaktionen eingegangen werden, welche von Benutzern ausgelöst werden.

Passiv Verwendung

Eine passive Verwendung des Second Screens ist durch die Möglichkeit der Ausgabe von Bild und Ton am Second Screen Manager gewährleistet. Dafür bot das *GUI-System* der *xis-engine*, dessen Schnittstellen in der Lua-Umgebung verfügbar gemacht wurden, ideale Möglichkeiten einer Individualisierung.

Ein Vorteil der sich durch die Installation von Subapps ergab, war dass Ressourcen schneller geladen werden konnten, als wenn diese über das Netzwerk stetig vom Hauptspiel zur Verfügung gestellt wurden.

Jedoch konnte aus Zeitgründen das Abspielen eines Videos am Second Screen Manager noch nicht realisiert werden, wie dies beispielsweise für ein Making-of-Video in Abschnitt 4.2.4 benötigt werden würde. Dennoch ist denkbar ein Animationsvideo unter Verwendung von animierten Grafiken und Transformationen von **Components** des *GUI-Systems* mit der Subapp umsetzen zu können.

8.7 Entwicklerfreundlichkeit

Das Hauptaugenmerk in der Entwicklung des Frameworks richtete sich auf das Erschaffen eines entwicklerfreundlichen Werkzeugs, mit welchem es möglich ist, einen Second Screen ohne großen Aufwand in ein Spiel zu integrieren. Dies konnte mit dem Slave-Framework als auch mit dem Subapp-Framework erreicht werden.

8.7.1 Etablieren der Hauptverbindung

Das Etablieren der Hauptverbindung konnte in überschaubare Schritte gegliedert werden (s. Abschn. 5.2), welche vom Spieleentwickler durchgeführt werden müssen. Das zeigt, dass die Abstrahierung der Netzwerkkommunikation im SDK erforderlich war, da eine direkte Verwendung von *Boost.Asio* weniger überschaubar gewesen wäre. Hierbei hat sich vor allem die Verwendung des Eventprotokolls bewährt (s. Abschn. 5.3).

8.7.2 Direkte Steuerung

Der Karten-Prototyp mit dem Slave-Framework zeigte, dass vom Spieleentwickler nur wenige Funktionsaufrufe nötig waren, um eine Karte am Second Screen visualisieren zu können (s. Abschn. 7.2.3). Wie das Beispiel zeigt,

wurde die Darstellung sehr einfach gehalten, da nur schwarze Rechtecke für Plattformen der Spielwelt benutzt wurden. In einem nächsten Schritt könnten die skizzierten Plattformen durch Grafiken ersetzt werden, um aus dem Prototyp eine vollwertige Second Screen Anwendung zu modellieren.

8.7.3 Subappentwicklung

Das Einbinden von Grafiken wurde im Subapp-Framework mit dem Ressourcenbündel möglich. Bei der Entwicklung des Abenteuer-Prototyps zeigte sich, dass die Zusammenfassung aller benötigten Dateien als Archiv-Datei die Installation der Subapp vereinfachte. Es konnte die Subapp mit einem Befehl installiert werden, ohne dass für jede Datei ein eigener SDK-Aufruf benötigt wurde.

Die Verwendung einer Skriptsprache am SSM ermöglichte eine unabhängige Nutzung, dennoch besteht hier ein Mehraufwand in der Implementierung bevor eine Second Screen Anwendung möglich ist. Hierbei muss unter Einsatz der Skriptsprache ein GUI erzeugt werden. Im Slave-Framework reichte das Senden von Befehlen aus, um eine Second Screen Anwendung darstellen zu können. Im Subapp-Framework müssen die Schnittstellen zum Ändern des GUIs selbst definiert werden.

Es wurde versucht die Programmierung der Subapp plattformunabhängig zu gestalten, um den Entwicklungsaufwand zu reduzieren. D. h. es wurde möglich, dass der entwickelte Programmcode auf jedem Second Screen Manager ausgeführt werden kann, egal auf welchem Betriebssystem der SSM ausgeführt wird. Weiters wurde im Zuge der Implementierung ein Erweiterungssystem erschaffen, welches das Wiederverwenden von vorhandenem Programmcode ermöglicht (s. Abschn. 6.3.4).

Eine Hürde in der Verwendung des Subapp-Frameworks liegt in der Wahl der Skriptsprache Lua. Spieleentwickler, welche der Sprache Lua nicht mächtig sind, oder nicht über die Ressourcen verfügen, sich in Lua einzulernen, könnten Schwierigkeiten haben, einen Second Screen umzusetzen.

Unterschied zum Slave-Framework

Im Slave-Framework wurden Eingabeereignisse an das Hauptspiel gesendet (s. Abschn. 4.4.1). Spieleentwickler können in der Subapp in Lua einen Berührungsevent im Voraus auswerten, um anschließend einen gezielten Befehl an das Hauptspiel zu senden.

Nachdem nur ein gezielter Befehl an das Hauptspiel versendet wird, kann sich die Struktur der Subapp nach Belieben ändern. Dabei muss auf Seiten des Hauptspiels keine Rücksicht genommen werden, welche ID ein `Component` aufweist, nachdem ein konkreter Befehl und nicht die ID des `Components` von der Subapp an das Hauptspiel gesendet wird. Diese bewusste Trennung schafft einen besseren Überblick über den Informations-

fluss zwischen dem Hauptspiel und dem Second Screen und bietet zudem eine gezielte Entkopplung der Schnittstellen. Weiters wird die Rechenleistung des Hauptspiels entlastet, da keine Berechnungen für die Logik des Second Screens durchgeführt werden müssen.

8.8 Benutzerfreundlichkeit

Im Design des Second Screen Managers wurde besonders auf die Benutzerfreundlichkeit Wert gelegt. Hierbei wurde versucht die nötigen Schritte für den Installationsvorgang, das Ausführen und wieder Beenden einer Subapp möglichst gering zu halten.

8.8.1 Interface des Second Screen Managers

Wird der SSM gestartet, wird unmittelbar nach dem Ladevorgang aller benötigten Ressourcen für den SSM, ein übersichtliches Menü mit den installierten Subapps dargestellt. Wenn die benötigte Subapp bereits installiert ist, sind nur zwei Schritte notwendig um diese zu Öffnen:

- starten des Second Screen Managers,
- ausführen der Subapp.

Anschließend kann sich das Hauptspiel zum SSM verbinden.

Die Einstellungen des SSM werden auf einer separaten Ansicht am SSM vorgenommen, obwohl zur Zeit nur zwei Einstellungen möglich sind: Der Name und das Passwort. Zum Ändern des Namens oder Passworts sind drei Schritte nötig:

- starten des Second Screen Managers,
- öffnen der Einstellungsseite,
- Name/Passwort ändern.

Würden die Felder zum Eingeben der Werte auf der Menüseite dargestellt werden, könnte der zweite Schritt eingespart werden. Jedoch wurde davon ausgegangen, dass in den weiteren Entwicklungsschritten des Frameworks mehr Einstellungsmöglichkeiten folgen könnten, welche nicht alle im Menü dargestellt werden können, da dies zu unübersichtlich werden würde. Weiters wurde der Schritt bewusst gesetzt, da das Ändern der Einstellungen in Relation zum Öffnen und Installieren von Subapps sehr selten durchgeführt wird und dadurch nicht stetig angezeigt werden muss, um ein schlichtes, überschaubares Menü zu erhalten.

8.8.2 Installationsvorgang

Besonders wurde im Konzept auf die Benutzerfreundlichkeit der Installation der Subapp eingegangen. Wie der Abenteuer-Prototyp zeigt, kann die Installation einer Subapp, unter Verwendung des SDKs, benutzerfreundlich

in vier Schritten gestaltet werden (s. Abschn. 7.4). Zudem sind nur wenige Schritte bis zur allerersten Verwendung einer Subapp am SSM notwendig:

1. Second Screen Manager herunterladen und installieren.
2. Hauptspiel und Second Screen Manager starten.
3. am Hauptspiel zum *Verbindungsmenü* navigieren und
4. gefundenen Second Screen Manager Eintrag auswählen.
5. ggfs. ein Passwort eingeben.
Installation startet automatisch und wird abgeschlossen.
6. Subapp am Second Screen Manager ausführen und
7. im Hauptspiel ein neues Spiel starten.

Allerdings muss das Installations-GUI von jedem individuellen Spiel selbst entworfen werden.

8.8.3 Deinstallation einer Subapp

In der Umsetzung des Subapp-Frameworks fehlt die Deinstallation einzelner Subapps. Diese ist jedoch notwendig um Benutzern die Möglichkeit zu geben, Subapps einzeln entfernen zu können ohne den gesamten Second Screen Manager deinstallieren zu müssen. Des Weiteren wird in diesem Konzept erwähnt, dass die Größe des benötigten Speichervolumens einer Subapp angezeigt werden sollte. D. h. es wurden Konzepte für eine benutzerfreundlichere Entfernung von Subapps aufgestellt, jedoch noch nicht im Second Screen Manager umgesetzt. Dies könnte daher einer der nächsten wesentlichen Punkte in der Weiterentwicklung des Frameworks sein.

8.9 Zusammenfassung

Unter Berücksichtigung aller oben genannten Punkte kann abschließend zusammengefasst werden, dass das entwickelte Framework im Wesentlichen die Anforderungen erfüllt, allerdings noch nicht alles vollständig umgesetzt werden konnte. Trotzdem ist es schon möglich in kurzer Zeit und mit relativ geringem Aufwand unterschiedlichste Second Screen Anwendung zu entwickeln. Auch wenn das entwickelte Framework derzeit noch nicht veröffentlicht werden kann, bildet es eine vielversprechende Basis, welche für den Einsatz in unterschiedlichsten Projekten und mit verschiedenen Game-Engines weiterentwickelt werden könnte.

Kapitel 9

Schlussbemerkungen

9.1 Zusammenfassung

In der vorliegenden Arbeit wurde versucht ein Framework zu entwickeln, welches Spieleentwicklern die Möglichkeit bietet auf einfachste Weise einen Second Screen in ihre Spiele zu integrieren. Dabei entstanden zwei unterschiedliche Ansätze.

Der erste Ansatz verfolgte eine direkte Steuerung des Second Screens mithilfe von Befehlen, welche eine grafische Benutzeroberfläche manipulierten. Unter Verwendung dieses Konzepts konnten bereits universelle Second Screen Anwendungen umgesetzt werden. Dabei stellte sich heraus, dass eine unabhängige Nutzung des Second Screens nicht möglich war, da zur Benutzung des Second Screens immer das Hauptspiel benötigt wurde.

Jedoch zeigte die Analyse von aktuellen Second Screen Gaming Anwendungen, dass ein, zu einem Hauptspiel zugehöriges Minispiel, auch als Second Screen betrachtet werden kann.

Daher wurde das Grundkonzept der direkten Steuerung verworfen und ein neues Framework entwickelt, in welchem es möglich war unter Einsatz einer Skriptsprache eine Second Screen Anwendung unabhängig vom Hauptspiel auszuführen. Der Skriptcode und die benötigten Ressourcen der Second Screen Anwendung wurden unter dem Begriff *Subapp* zusammengefasst. Je eine Subapp auf der mobilen App stellt einen Second Screen für ein Hauptspiel dar.

Das neue Framework bietet zudem die Möglichkeit einer Entwicklung von sogenannten *Extensions*. Diese bestehen aus einem Hauptspiel-Teil und einem Second Screen-Teil, welche über Textnachrichten kommunizieren können. Der entwickelte Programmcode kann daher wiederverwendet werden, wodurch die Entwicklungszeit für neue Second Screen Anwendungen verkürzt werden kann.

Die Kommunikation zwischen dem Second Screen und dem Hauptspiel wurde mit einer lokalen Netzwerkverbindung gelöst, welche für das Haupt-

spiel mithilfe des entworfenen SDKs abstrahiert werden konnte. Zudem entstand ein Eventprotokoll, bei welchem Befehle zwischen dem SDK und dem Second Screen als Zeichenketten ausgetauscht werden. Dieses zeigte sich als hilfreich, da Spieleentwickler selbst Events definieren können, mit welchen der Second Screen gesteuert werden kann. Für die Netzwerkkommunikation wurde die Bibliothek *Boost.Asio* verwendet, welche Dank einer guten Onlinedokumentation und zahlreichen Beispielen, welche im Web vorhanden waren, zügig integriert werden konnte.

9.2 Ausblick

Mit dem entwickelten Framework ist es möglich verschiedene Second Screens plattformunabhängig und mit geringem Zeitaufwand zu entwickeln. Jedoch ist von einer Veröffentlichung der jetzigen Version abzusehen. Im Folgenden werden die Gründe dafür genauer erläutert.

Nutzertests

Im Rahmen der Arbeit konnte das Framework noch nicht von Entwicklern getestet werden, da ein solcher Test sehr umfangreich gewesen wäre und eine zu lange Laufzeit benötigt hätte, nachdem sich Entwickler in das Framework einarbeiten müssten, Hauptspiele gefunden werden müssten, in welche Second Screen Anwendungen sinnvoll integriert werden können und Subapps programmiert werden müssten.

Um zu gewährleisten, dass der Second Screen Manager auch problemlos von Spielern ohne Vorkenntnisse benutzt werden kann, ist noch ein Benutzertest ausständig. Darin könnte festgestellt werden, ob der Installationsvorgang der Subapp verstanden wird, ob Probleme in der Benutzung von zwei Geräten zur selben Zeit entstehen oder grafische Benutzeroberflächen zum besseren Verständnis angepasst werden müssen.

Veröffentlichung unter iOS

Im Zuge der Arbeit stellte sich ein Problem in der Veröffentlichung der iOS Version des Second Screen Manager heraus. In den Richtlinien von Apple zur Entwicklung von iOS Apps ist seit 2010 Folgendes vermerkt [39]:

3.3.2. An Application may not download or install executable code. Interpreted code may only be used in an Application if all scripts, code and interpreters are packaged in the Application and not downloaded. The only exception to the foregoing is scripts and code downloaded and run by Apple's built-in WebKit framework.

Das bedeutet, dass Subapps nicht auf einen iOS Second Screen Manager installiert werden dürfen, da eine Subapp Skriptcode beinhaltet, welcher nicht nachträglich in eine App heruntergeladen werden darf. Dieses Problem hatte jedoch keinen Einfluss auf die vorliegende Arbeit. Auf der einen Seite ist es durchaus möglich den Second Screen Manager für Android oder Windows Phone zu veröffentlichen und auf der anderen Seite wurde eine Lösung für dieses Problem gefunden: Anstatt Subapps von extern zu installieren, werden diese direkt in den Second Screen Manager integriert. D. h. die Logik des Second Screen Manager wird als SDK extrahiert und kann in eine eigenständige iOS App integriert werden. Dieser eigenständigen App wird anschließend die entwickelte Subapp als Ressource beigelegt. Dadurch ist weiterhin eine voll funktionsfähige Subapp möglich, welche auf der einen Seite in den Android- oder Windows Phone-Second Screen Managern installiert und unter iOS in eine eigene App verpackt wird. Seitens des SDKs des Hauptspiels ist nicht zu unterscheiden, ob es sich beim gefundenen Second Screen Manager um einen universellen Manager handelt, bei welchem Subapps installiert werden können, oder ob es sich dabei um einen konkretisierten Second Screen handelt, in welchem eine einzige Subapp bereits vorinstalliert ist.

Weiterentwicklungen

Wie die Evaluierung aufzeigt, benötigt das finale Framework die Implementierungen der konzipierten Sicherheitsmaßnahmen und der Deinstallation einer Subapp. Diese wurden bereits ausführlich im Konzept ausgearbeitet, jedoch konnten sie noch nicht umgesetzt und getestet werden.

Bevor das Framework veröffentlicht wird, muss jedenfalls sichergestellt werden, dass entwickelte Skripts für den Second Screen, welche am Second Screen Manager ausgeführt werden, nicht unbefugt persönliche Daten auslesen und an ein Hauptspiel zurücksenden können. Weiters ist die Deinstallation von Subapps nötig, da, wie bereits erwähnt, vermieden werden sollte, dass der gesamte Second Screen Manager deinstalliert werden muss, wenn Speicherkapazität am mobilen Gerät benötigt wird.

Das SDK für das Framework wurde in C++ entwickelt. Wie bereits vorgeschlagen, kann dieses SDK mit dem JNI in Java eingebunden werden. Eine weitere Möglichkeit könnte die Implementierung des SDKs für unterschiedliche Programmiersprachen sein. Zum Beispiel könnte ein SDK mit Java, C oder Objective-C entwickelt werden und ähnliche Schnittstellen wie das C++ SDK bereitstellen. Das würde eine native Integration in Android, Windows Phone oder iOS ermöglichen ohne einen C++ Kontext zu benötigen.

Weiters zeigte sich in der Entwicklung der Prototypen, dass ein grafisches *Verbindungs*menü für Elements entwickelt werden musste. *Verbindungs*menüs am Hauptspiel für das Verbinden zum Second Screen könnten für häufig verwendete Game-Engines vorprogrammiert werden. D. h. es

könnte beispielsweise ein eigenes *Verbindungs*menü für *Unity*¹, *Cocos2D-x*² oder *libgdx*³ entwickelt werden.

Die entwickelten Prototypen zum Testen der Frameworks könnten weiter ausgebaut werden und vollwertig in das Hauptspiel *Elements* unter Verwendung des finalen Frameworks integriert werden. Damit könnte ein erstes Vorzeigeprojekt für den Einsatz des Frameworks veröffentlicht werden.

¹<http://unity3d.com>

²<http://www.cocos2d-x.org>

³<http://libgdx.badlogicgames.com>

Anhang A

Inhalt der CD-ROM

Format: CD-ROM, Single Layer, ISO9660-Format

A.1 Masterarbeit

Pfad: /

[Lipphart2014.pdf](#) Masterarbeit

A.2 Anwendungen

Pfad: /

[Elements/bin/](#) Ausführbare MacOS X Version des
Hauptspiels *Elements*

[SSM/bin/](#) Ausführbare MacOS X Version des SSMs

A.3 Quellcode

Pfad: /

[Abenteuer-Subapp/](#) . . Lua Quellcode der Abenteuer-Subapp

[S3DK/](#) Quellcode des S3DKs

[SSM/src/](#) Quellcode des Second Screen Managers

A.4 Sonstiges

Pfad: /

[Online-Quellen/](#) Kopien der Online-Quellen

[Bilder/](#) Verwendete Bilder und Grafiken

Quellenverzeichnis

Literatur

- [1] John Ahlquist, Jr. und Jeannie Novak. *Game Development Essentials – Game Artificial Intelligence*. Canada: Thomson Learning Inc., 2008 (siehe S. 61).
- [2] Mark Claypool und Kajal Claypool. „Latency can kill: precision and deadline in online games“. In: *Proceedings of the first annual ACM SIGMM conference on Multimedia systems*. MMSys '10. Phoenix, Arizona, USA: ACM, 2010, S. 215–222 (siehe S. 46).
- [3] Alan Dix u. a. *Human-Computer Interaction – Third Edition*. 3. Aufl. Edinburgh Gate, England: Wordware Publishing, 2004 (siehe S. 17).
- [4] Jason Gregory. *Game Engine Architecture*. Wellesley, Massachusetts: A K Peters. Ltd., 2009 (siehe S. 88).
- [5] Roberto Ierusalimsky. *Programming in Lua*. 3. Aufl. Rio de Janeiro: Lua.org, 2013 (siehe S. 62, 63).
- [6] Nicholas Katzakis u. a. „Smartphone Game Controller“. In: *74th HIS SigVR Workshop*. Tokyo, Japan, 2011, S. 55–60 (siehe S. 5).
- [7] Gerhard Lienemann und Dirk Larisch. *TCP/IP – Grundlagen und Praxis*. 2. Aufl. Hamburg, DE: Heise, 2014 (siehe S. 54).
- [8] Daniel Natapov, Steven J. Castellucci und I. Scott MacKenzie. „ISO 9241-9 Evaluation of Video Game Controllers“. In: *Proceedings of Graphics Interface 2009*. GI '09. Kelowna, British Columbia, Canada: Canadian Information Processing Society, 2009, S. 223–230 (siehe S. 20).
- [9] Andrew Rollings und Dave Morris. *Game Architecture and Design – A New Edition*. Indiana 46240, USA: New Riders Publishing, 2004 (siehe S. 59, 88).
- [10] Richard Rouse. *Game Design – Theory and Practice*. 2. Aufl. United States: Wordware Publishing, 2005 (siehe S. 22).
- [11] Katie Salen und Eric Zimmerman. *Rules of Play: Game Design Fundamentals*. The MIT Press, 2003 (siehe S. 21, 22).

- [12] Kevin Saunders und Jeannie Novak. *Game development essentials – Game interface design*. 2. Aufl. Clifton Park, USA: Delmar, Cengage Learning, 2012 (siehe S. 21).
- [13] John Torjo. *Boost.Asio C++ Network Programming*. Birmingham, UK: Packt Publishing, 2013 (siehe S. 61, 68).
- [14] Clara Fernández Vara. „The Tribulations of Adventure Games: Integrating Story into Simulation Through Performance“. Dissertation. Georgia: Georgia Institute of Technology, Dez. 2009 (siehe S. 22, 75).
- [15] Hao Wang und Chuen-tsai Sun. „Game reward systems: Gaming experiences and social meanings“. In: *Think Design Play: The fifth international conference of the Digital Research Association (DIGRA)*. Hilversum, Niederlande: DiGRA/Utrecht School of the Arts, Jan. 2011 (siehe S. 19).
- [16] Yonggao Yang und Lin Li. „Turn Smartphones into Computer Remote Controllers“. In: *International Journal of Computer Theory and Engineering* 4.4 (2012), S. 561–564 (siehe S. 3).
- [17] Harald Zisler. *Computer-Netzwerke – Grundlagen, Funktionsweise, Anwendung*. Bonn, DE: Galileo Press, 2012 (siehe S. 54).
- [18] Bernadette Zlabinger. „Interaktive Fernsehszenarien unter Verwendung von SmartTV API und externen Devices“. Diplomarbeit. Hagenberg, Austria: University of Applied Sciences Upper Austria, Digitale Medien, Juni 2013. URL: <http://theses.fh-hagenberg.at/thesis/Zlabinger13> (siehe S. 6, 7).

Software und Programmbibliotheken

- [19] FEO Media AB Grym! AS. *Quizduell – Fordere deine Freunde heraus*. App Store, Play Store, Windows Phone Store. 2012. URL: <http://www.quizduell-game.de> (siehe S. 7).
- [20] Angelika Bugl und Christoph Lipphart. *xis-engine 2*. 2013. URL: <http://www.itengy.com/xis-engine> (siehe S. 61).
- [21] Angelika Bugl u. a. *Elements Alpha*. 2014. URL: <http://www.itengy.com/elements> (siehe S. 63).
- [22] DICE. *Battlefield 4*. 2013. URL: <http://www.battlefield.com/de/battlefield-4> (siehe S. 15, 20, 21).
- [23] Naughty Dog. *The Last of Us*. 2013. URL: <http://www.thelastofus.playstation.com> (siehe S. 22).
- [24] Ubisoft Entertainment. *Assassin's Creed IV® Black Flag Companion*. App Store, Play Store. 2013. URL: <https://itunes.apple.com/de/app/assassins-creed-iv-black-flag/id692766233?mt=8> (siehe S. 21, 23).

- [25] Ghost Games. *Need for Speed: Rivals*. 2013. URL: <http://www.battlefield.com/de/battlefield-4> (siehe S. 20, 21).
- [26] Roberto Ierusalimschy, Waldemar Celes und Luiz Henrique de Figueiredo. *Lua*. 2014. URL: <http://www.lua.org> (siehe S. 63).
- [27] Ubisoft Montpellier. *Watch Dogs*. Nintendo Wii U. 2012. URL: <http://zombiu.ubi.com/zombi/de-de/home/> (siehe S. 17, 18).
- [28] Ubisoft Montreal. *Assassin's Creed*. 2007. URL: <http://assassinscreed.ubi.com/de-de/home/> (siehe S. 20, 21, 24).
- [29] Ubisoft Montreal. *Assassin's Creed: Black Flag*. 2013. URL: <http://www.ubi.com/DE/Games/Info.aspx?pld=11459> (siehe S. 14).
- [30] Ubisoft Montreal. *Watch Dogs*. 2014. URL: <http://watchdogs.ubi.com/> (siehe S. 14, 20, 21).
- [31] ROCCAT. *PowerGrid*. Online Download. 2013. URL: <http://power-grid.roccat.org/de/Home/> (siehe S. 4).
- [32] Steppschuh. *Remote Control Collection*. Google Play Store. 2014. URL: <http://remote-control-collection.com> (siehe S. 4).

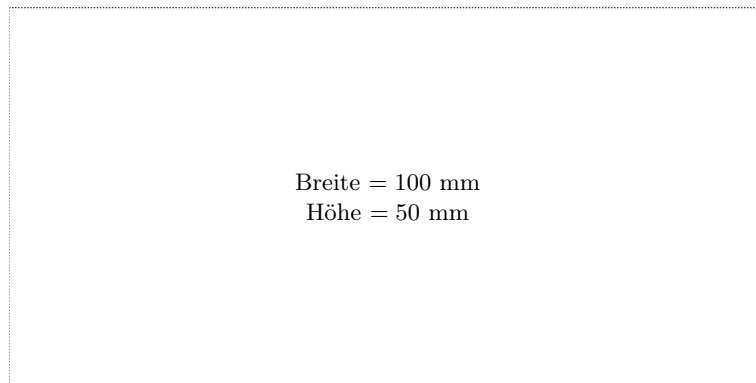
Online-Quellen

- [33] Apple Inc. *iOS App Programming Guide*. 2013. URL: <https://developer.apple.com/library/ios/documentation/iphone/conceptual/iphoneosprogrammingguide/TheiOSEnvironment/TheiOSEnvironment.html> (besucht am 16.09.2014) (siehe S. 91).
- [34] Grant Brunner. *The rise of second screen gaming: Where will the PS4 and Xbox One take us?* URL: <http://www.extremetech.com/gaming/174482-the-rise-of-second-screen-gaming-where-will-the-ps4-and-xbox-one-take-us> (besucht am 21.06.2014) (siehe S. 8, 9).
- [35] Fabio Busatto. *TCP Keepalive HOWTO*. Mai 2007. URL: <http://tldp.org/HOWTO/TCP-Keepalive-HOWTO/index.html> (besucht am 22.09.2014) (siehe S. 51).
- [36] *Connecting the Nintendo GameCube Game Boy Advance Link Cable*. URL: http://www.nintendo.com/consumer/systems/nintendogamecube/gcn_agb_connect.jsp (besucht am 30.07.2014) (siehe S. 10).
- [37] D4GAndy. 2012. URL: <http://www.d4gameplay.com/2012/06/zombiu-just-sold-the-wii-u-controller/> (besucht am 25.09.2014) (siehe S. 17).
- [38] Ubisoft Entertainment. *Tom Clancy – The Division*. URL: <http://tomclancy-thedivision.ubi.com/> (besucht am 30.07.2014) (siehe S. 20).

- [39] Peter Kafka. *Apple Hands App Developers an Olive Branch. What About Adobe and AdMob?* 2010. URL: <http://allthingsd.com/20100909/apple-hands-app-developers-an-olive-branch/> (besucht am 22.09.2014) (siehe S. 98).
- [40] lua-users wiki: *Lua Versus Python*. URL: <http://lua-users.org/wiki/LuaVersusPython> (besucht am 13.09.2014) (siehe S. 61, 62).
- [41] *Lua vs Python*. URL: <http://de.slideshare.net/cybrshin/lua-vs-python> (besucht am 13.09.2014) (siehe S. 61, 62).
- [42] *Marktanteile mobiler Betriebssysteme Q2 2014*. URL: <http://mobilestudien.de/marktanteile-betriebssysteme/marktanteile-mobiler-betriebssysteme-q2-2014/> (besucht am 25.08.2014) (siehe S. 43, 90).
- [43] *Microsoft's emphasizes second screen SmartGlass experience at GDC 2013*. URL: <http://o.canada.com/technology/gaming/microsofts-emphasizes-second-screen-smartglass-experience-at-gdc-2013/> (besucht am 09.01.2014) (siehe S. 12).
- [44] *Mit Xbox SmartGlass ist noch mehr Unterhaltung garantiert*. URL: <http://www.xbox.com/de-AT/smartglass/> (besucht am 09.01.2014) (siehe S. 12).
- [45] Network Working Group. *Uniform Resource Identifier (URI): Generic Syntax*. 2005. URL: <http://tools.ietf.org/pdf/rfc3986.pdf> (besucht am 29.08.2014) (siehe S. 55).
- [46] Norddeutscher Rundfunk. *Quizduell – Das Erste*. URL: <http://daserste.ndr.de/quizduell/> (besucht am 30.07.2014) (siehe S. 7).
- [47] Robert Nystrom. *Event Queue – Decoupling Patterns – Game Programming Patterns*. 2014. URL: <http://gameprogrammingpatterns.com/event-queue.html> (besucht am 16.09.2014) (siehe S. 88).
- [48] *Off with their HUDs!: rethinking the heads-up display in console game design*. URL: http://www.gamasutra.com/features/20060203/wilson_01.shtml (besucht am 24.07.2014) (siehe S. 21).
- [49] Ian Paul. *Wi-Fi Direct vs. Bluetooth 4.0: A Battle for Supremacy*. URL: http://www.pcworld.com/article/208778/Wi-Fi_Direct_vs_Bluetooth_4_0_A_Battle_for_Supremacy.html?page=2 (besucht am 26.08.2014) (siehe S. 46).
- [50] *Programming Languages Benchmarks*. URL: <http://attractivechaos.github.io/plb/> (besucht am 13.09.2014) (siehe S. 61).
- [51] Christian Schneider. *Function for Factorial in Python*. URL: <http://stackoverflow.com/questions/5136447/function-for-factorial-in-python> (besucht am 16.09.2014) (siehe S. 62).
- [52] *Virtuelle Realität und Haptik*. URL: <http://www.zwisler.de/scripts/haptics/haptics.html> (besucht am 24.07.2014) (siehe S. 21).

Messbox zur Druckkontrolle

— Druckgröße kontrollieren! —



— Diese Seite nach dem Druck entfernen! —