

OBACHT

DOKUMENTATION



INHALT

VORWORT

3

TEAM

4

Betreuer

4

Teammitglieder

5

VORGABEN

6

SPIELKONZEPT

7

Anforderungen an das Spiel

7

Ideenfindung

8

Multiplayer Konzept

9

Finales Spielekonzept

10

DESIGN

11

Corporate Design

11

Menü Design

12

Anleitung

13

Spiel Design

14

Character Entwicklung

15

Character Animation

16

Welten Farbkonzept

17

Welten Aufbau

18

Welten Design

19

Hindernisse

20

Spritesheets

21

Sound

22

Website

23

Ident-Video Konzept

24

Ident-Video Umsetzung

25

PROGRAMMIERUNG

26

Versionskontrolle

26

Entwicklungsumgebungen

27

Codingstandards

28

Wahl der Plattform

29

Wahl der Engine

30

Codearchitektur

31

Eventdiagramme

32

Menü Aufbau

36

Multiplayer-Service

39

Game (und Unterkomponenten)

40

Kollisionserkennung

41

Server

42

Website

45

PROJEKTMANAGEMENT

46

Agile Entwicklung

46

Aufgabenverwaltung

47

Meilensteine

48

Teammotivation

49

Marketing

50

Finanzierung

51

EVALUATION

52

Usability

52

Plattform & Game-Engine

54

Performance

55

NACHWORT

57

Fazit

57

Ausblick

58

Schlusswort

59

OBACHT

BAYERISCH FÜR ACHTUNG!

VORWORT

Und darauf kommt es bei diesem Multiplayer-Spiel für Smartphones auch an. Schnelle Reaktionen beim Ausweichen der Hindernisse sowie beim Einsammeln der Boni sind nötig, um seinen Gegner zu schlagen. Das Spiel funktioniert plattformunabhängig auf Android und iOS und

wurde in HTML5 mit der Spiele-Engine „LimeJS“ umgesetzt. Das Spiel entstand im Zuge der Medienprojekte des sechsten Semesters des Studiengangs Interaktive Medien an der Hochschule Augsburg. Bei den Medienprojekten bekommen Studenten in 8- bis 10-köpfigen Gruppen

einen Auftrag, für dessen Umsetzung das Team ein Semester lang Zeit hat. Zu den Aufgaben zählen Ideenfindung, Konzeption, Gestaltung, Umsetzung und Marketing/Präsentation.



Prof. Michael Stoll

zuständig für den gestalterischen Teil des Projektes

Prof. Dr. Wolfgang Kowarschick

zuständig für den Informatik-Teil des Projektes

BETREUER

Die Betreuer standen dem Team mit Rat zur Seite und beobachteten den Fortschritt des Projektteams. Sie bestimmten und formulierten auch die Aufgabenstellung des Projektes.

TEAM



Claudia Kölbl

Matrikelnummer: 924367

Kontakt: info@claudia-koelbl.de

Aufgaben: Welten-Design, Client-Programmierung, Spritesheets, Ident-Video, Marketing, Design-Betreuung

Fabian Bühler

Matrikelnummer: 920065

Kontakt: fabian.b89@gmail.com

Aufgaben: Kollisionserkennung und Spielmechanik, Mobile Website, Komposition des Soundtracks

Eduard Heitz

Matrikelnummer: 924326

Kontakt: eddy63@gmx.de

Aufgaben: Charakterdesign und Animation, Vorarbeit für den Usability-Test

Simon Heimler

Matrikelnummer:

924317

Kontakt:

heimlersimon@gmail.com

Aufgaben: Programmierung und Aufsetzen des Multiplayer Servers, Code-Architektur, Coding Standards, Performance-Optimierung, Bugfixing, Usability-Auswertung, Unterstützung wo gebraucht

Alina Fink

Matrikelnummer:

924364

Kontakt:

alina-fi@web.de

Aufgaben: Anleitung, Credits und Menü (Design und Client-Programmierung), Präsentation

HOGI

Aufgabe: Hauptrolle im Spiel

Martin Hofmann

Matrikelnummer: 924358

Kontakt: martin.hofmann11@gmail.com

Aufgaben:

Client-Programmierung, Kontakt mit Entwicklern der Game-Engine, Performance-Optimierung, Bugfixing, Website, Erstellung des Ident-Videos

Lukas Jaborsky

Matrikelnummer: 924325

Kontakt: info@lukas-jaborsky.de

Aufgaben: Projektmanagement, Konzeption, Marketing, Usability, Präsentation, Client-Programmierung

Sebastian Huber

Matrikelnummer: 924349

Kontakt: sebastianhuber.mod@web.de

Aufgaben: Git-Betreuung, Client-Server-Kommunikation, Realisierung der Menü-Logik, Code-Architektur, Performance-Optimierung, Bugfixing, Usability-Auswertung, Unterstützung wo gebraucht

” MULTIPLAYER-GAME FÜR SMARTPHONES “

PROJEKTVORGABEN

Die Aufgabenstellungen der Projekte sind kurz und einfach gehalten, sie bieten den Teams in Rücksprache mit den Betreuern viel Spielraum für eigene Interpretationen und Ideen.

Anforderungen

- Entwicklung eines Multiplayer-Games für Smartphones
- Spielbar von 2 oder mehr Spielern
- Kommunikation über Bluetooth und/oder WLAN

Aufgaben

- Entwicklung einer Spielidee
- Technische Umsetzung
- Testbetrieb und Evaluation

**INNOVATIV
SIMPEL
KOMMUNIKATIV**

ANFORDERUNGEN AN DAS SPIEL

In der Findung der Spielidee war das Team frei. Jedoch sollte das Augenmerk auf einer innovativen und kommunikativen Idee liegen. So soll dem allgemeinen Trend, dass sich Smartphone-Nutzer zunehmend bei Aktivitäten mit ihrem Handy isolieren, entgegen gewirkt werden.

Als Anforderungen für unsere Spielidee kristallisierten sich letzten Endes die drei Schlagworte simpel, innovativ und kommunikativ heraus.

IDEENFINDUNG



Der erste Schritt der Ideenfindung war es, sich am Markt zu orientieren. Was gibt es für Multiplayer-Spielkonzepte? Welche machen Spaß? Welche Ideen und Konzepte erfreuen sich im Moment großer Beliebtheit? In welchen Spielbereichen ist der Markt gesättigt? Wie gestaltet und programmiert man plattformunabhän-

gig? Wir untersuchten die unterschiedlichsten Spiele auf verschiedenste Aspekte wie Spielspaß, Steuerung, Multiplayer-Modus, etc.. Basierend auf diesen Kenntnissen begannen wir eigene Spieleideen zu entwickeln. Die verschiedenen Ideen wurden dann vom Team nach vorher festgelegten Kriterien bewertet:

- **Erweiterbarkeit**
- **Komplexität**
- **Gestaltungsmöglichkeit**
- **Innovation**
- **Spielspaß**

Basierend auf den Ergebnissen der Teambewertung nahmen wir die Grundidee des Hindernislaufs und begannen mit der Konzeption des Spiels.



MULTIPLAYER KONZEPT

Wir entschieden uns, zwei Spieler in kurzen Runden gegeneinander antreten zu lassen. Die Verbindung erfolgt über einen **Online-Server**. Die Nutzer der Anwendung haben die Möglichkeit, sich über den **Random-Modus** mit einem zufälligen Gegner, oder über den **Friend-Modus** mit einem Freund, zu verbinden.

Dabei bekommt einer der beiden Spieler eine 4-stellige Nummer, die der andere Spieler eingeben muss, um ein gemeinsames Spiel zu erstellen.

Die Spieler müssen dazu lediglich mit einem halbwegs performanten Internet verbunden sein, keine weiteren Einstellungen sind nötig.



Im Spielfeld befinden sich **zwei Planeten**. Diese werden jeweils von einem **Hogi** bevölkert. Der eigene Planet eines Spielers befindet sich links unten am Spielrand, der gegnerische Planet rechts oben. Beide Welten drehen sich so schnell, dass Hogi wie auf einem Laufband laufen muss, um an Ort und Stelle zu bleiben. Dabei muss er den **Hindernissen**, die seinen Weg kreuzen, aus dem Weg gehen. Hin-

dernissen, die ihm am Boden entgegen kommen, muss der Spieler durch Springen ausweichen, Hindernisse aus der Luft können kriechend gemeistert werden. In der oberen rechten Ecke befindet sich eine Art Inventar, dort werden **Bonus-Hindernisse** abgelegt die ein Spieler auf den gegnerischen Planeten werfen kann um dem Konkurrenten Schaden zuzufügen. Die Bonus-Hindernisse erhalten die

Spieler indem sie auf regelmäßig erscheinende **Reaktionbuttons** klicken. Beide erscheinen **zeitgleich auf beiden Smartphones**, aber nur der schnellere Spieler erhält den Bonus und somit ein weiteres Hindernis für sein Inventar. Schafft es ein Spieler nicht, ein Hindernis zu überwinden, wird ihm eins von **drei Leben** abgezogen. Sind alle Leben verbraucht, ist der Wettkampf verloren.

OBACHT

CORPORATE DESIGN



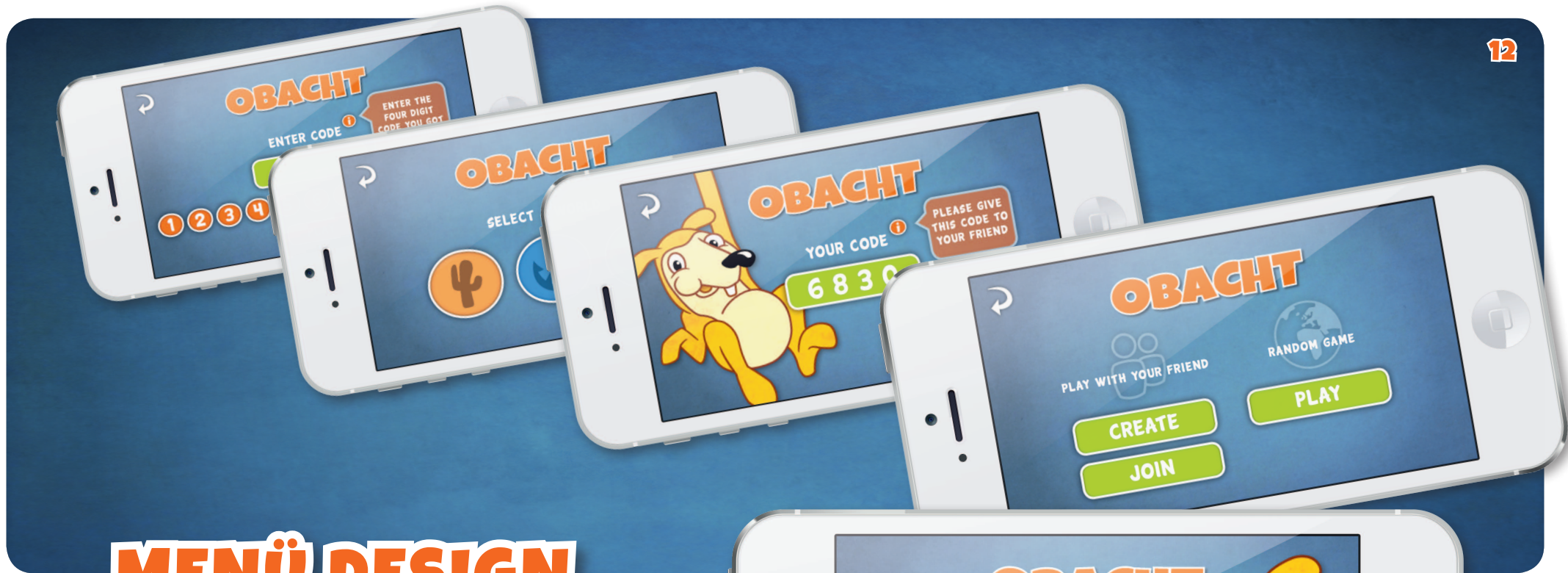
Das Erscheinungsbild von „OBACHT“ ist passend zur Anwendung **verspielt, lebendig und auffallend**. Eine strukturierte blaue Fläche mit Verlauf dient als Hintergrund. Dazu haben wir das Logo in der Komplementärfarbe **Orange (RGB 255/90/0)** mit weißer Kontur in der Schriftart **„Luckiest Guy“** (ausschließlich Versalien) entworfen. Ergänzend für

die Verwendung im Menü des Spieles haben wir die Schriftart **“Cartwheel”** ausgesucht. Für Fließtext wurde die Schrift **“Open Sans”** in verschiedenen Schriftschnitten verwendet. Der **Spiele-Charakter „Hogi“** wurde ebenfalls in das Corporate Design des Spieles integriert. Er ist in verschiedenen Posen dargestellt und wird als **Key Visual** eingesetzt.

LUCKIEST GUY

CARTWHEEL

Open Sans



MENÜ DESIGN

Das Menü ist nach dem Corporate Design gestaltet. Für die Buttons haben wir zum einen die Form des Buchstaben O aus unserem „OBACHT“-Logo aufgegriffen. Zum anderen wurden rechteckige, neongrüne Buttons mit abgerundeten Ecken und weißer Kontur für wichtige Elemente (z.B. PLAY) innerhalb des Menüs verwendet. Weiße Pfeilicons helfen bei der „zurück“-Navigation innerhalb des Menüs.





ANLEITUNG

In der Spielanleitung (zu finden im Menü unter HELP) wird weitestgehend mithilfe von anschaulichen Grafiken und kleinen Animationen das **Spielkonzept erklärt**. Auf einem, durch eine reduzierte Grafik dargestellten, Smartphone, gehalten von zwei Händen, werden der Spielaufbau, die Steuerung und das Ziel des Spiels erklärt. Kreisförmige, orangefarbene „Störer“ halten Extratipps für den Spieler bereit.





SPIEL DESIGN

Im Spiel stehen sich die **Planeten** des eigenen und des gegnerischen Hogis **gegenüber**. Jeder Spieler sieht seinen Character im unteren Bereich des Bildschirms und den Gegner „auf dem Kopf stehend“ im oberen Bereich.

Ein X im rechten unteren Eck bietet dem Spieler die Möglichkeit das Spiel jederzeit zu verlassen und in das Menü zurückzukehren.

Im linken unteren Eck befindet sich die Lebensanzeige in Form von Miniaturver-

sionen des Characters in weiß mit leichter Transparenz. Jeder Spieler hat 3 Leben. **Rechts oben befindet sich noch das Inventar** für drei zusätzliche Hindernisse für den Gegner, welches durch schnelle Reaktion aufgefüllt werden kann.

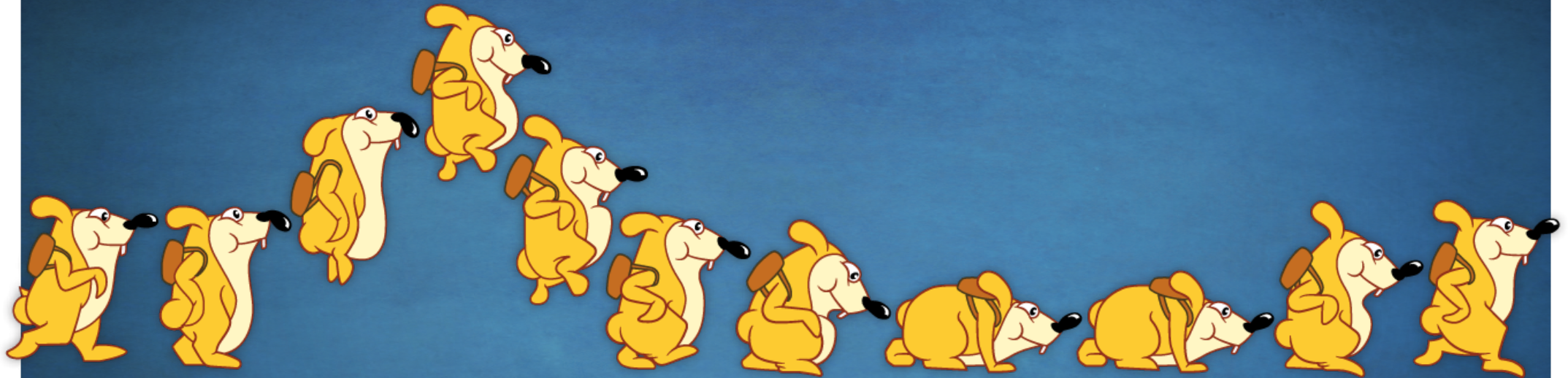


CHARACTER ENTWICKLUNG

Die Charakterentwicklung begann mit einer Reihe von relativ zusammenhangslosen Charakterentwürfen, da das Spielthema anfangs nicht feststand und theoretisch alles möglich war. Zugleich wurde ein Bewegungsprototyp in Form eines neutralen Strichmännchen-Walkcycles entworfen und die ersten **Bewegungsstudien** unternommen. Auf Grundlage dieses Bewegungsprototypen sollte der finale Charakter später

animiert werden sobald dessen Form feststand. Durchgesetzt hat sich eine von Alina Fink vorgeschlagene Figur eines Murmeltiers, passend zum bayerischen Spiele-Titel „OB-ACHT“. Diese erste Figur wurde abgewandelt, mehrmals überarbeitet, variiert und mit Fokus auf **interessante Bewegungsdynamik** weiterentwickelt. Ebenso wurde auf das Welten Farb-Konzept eingegangen, wofür passende **Farben und Accessoires** gefunden

wurden. Das Erscheinungsbild der Figur soll **witzig, sympathisch, niedlich**, aber auch etwas **treudoof** und **übergewichtig** wirken. Für Spielmenü und Website wurden spezielle Versionen des Charakters erstellt, die ihn auch frontal in unterschiedlichen Situationen zeigen und durchaus Maskottchen-Potenzial besitzen. Die Figur wurde letztendlich „Hogi“ getauft, angelehnt an das englische Wort von Murmeltier dem „groundhog“.



CHARACTER ANIMATION

Nachdem die Form der Figur feststand, wurde diese mit Hilfe des Bewegungsprototyps Schritt für Schritt zum Leben gebracht (siehe Hogi-Evolutionsvideo). Dabei wurde eine grobe Bewegungsskizze der Charakter-Form gezeichnet und **Bild für Bild** in Flash animiert. Durch die grobe Art der Zeichnung konnte im ersten Schritt schnell die Bewe-

gung definiert, und Fehler leicht abgeändert und korrigiert werden. Im nächsten Schritt wurde diese grobe Animation als PNG-Sequenz exportiert und Bild für Bild in Illustrator **vektorsiert**. Hierzu wurden auch alle Accessoires in der groben Animation mit verbaut, um diese in Illustrator über verschiedene Layer an- und ausschalten zu können.

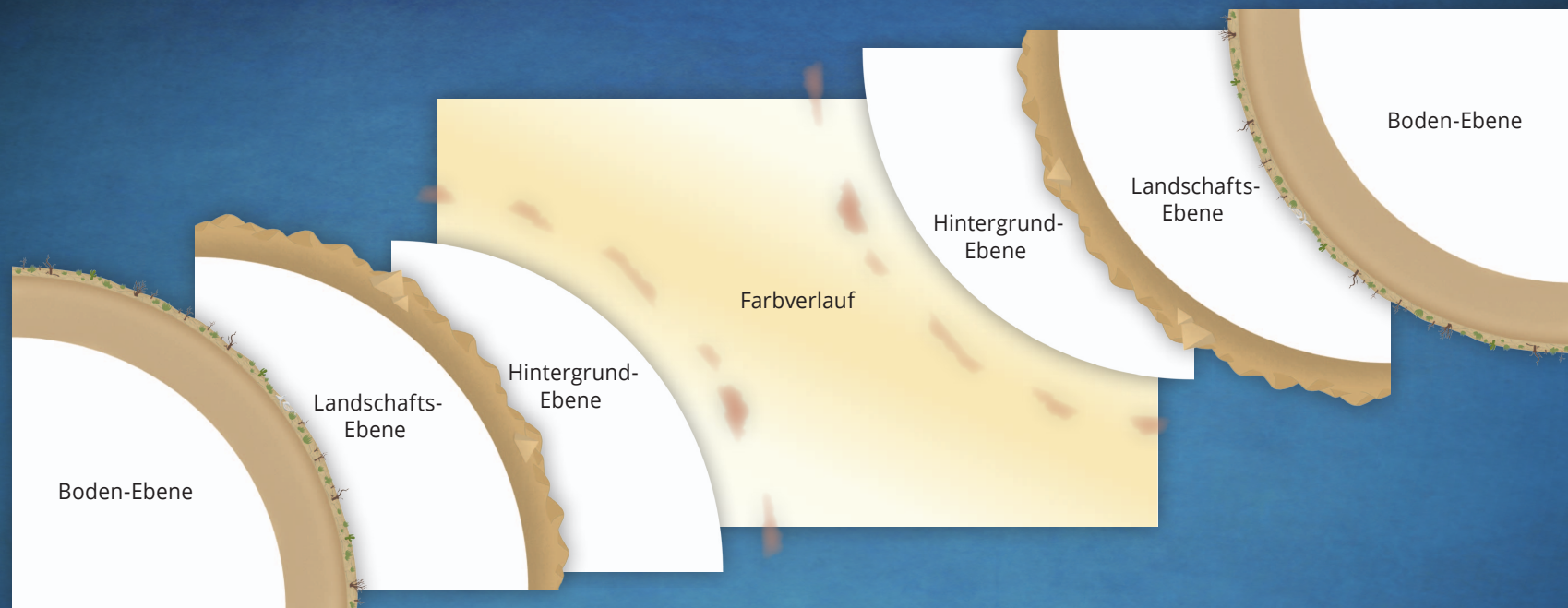
Im letzten Schritt wurden alle Animationsphasen in eine gemeinsame Szene geladen und die Einzelbilder auf eine **einheitliche Framegröße** zugeschnitten, wodurch es im Endprodukt keine Größenunterschiede und Sprünge der Animationsphasen geben sollte.



WELTEN FARBKONZEPT

Die Welten zeichnen sich durch unterschiedliche Farben aus. **Jede Welt hat eine andere Hauptfarbe** hinter der sich ein passendes Thema verbirgt. So steckt hinter der Farbe Blau eine **Unterwasser-Welt**, hinter dem sandfarbigen Beige eine **Wüsten-Welt** und das kräftige Grün vertritt eine **Wiesen-Welt**.

Ursprünglich waren mehrere Welten geplant: Braun (Höhlen-Welt), Rosa (Candyland), Hellblau (Luft-Welt), Rot (Lava-Welt), Schwarz (Nacht-Welt), etc. Aus zeitlichen Gründen haben wir uns jedoch auf die drei oben genannten Welten konzentriert.



WELTEN AUFBAU

Eine Welt besteht aus zwei identischen Planeten. Jeder Planet baut sich aus drei Ebenen auf. Die **Boden-Ebene**, auf der sich der Character als auch die Hindernisse befinden erzeugt gleichzeitig eine räumliche Tiefe. Die **Landschafts-Ebene** soll das Thema mehr hervorheben und die **Hintergrund-Ebene** soll unauffällig

den räumlichen Eindruck verstärken. Zusätzlich befindet sich hinter diesen Ebenen noch ein **dezentler Farbverlauf** um die Stimmung zu unterstreichen.

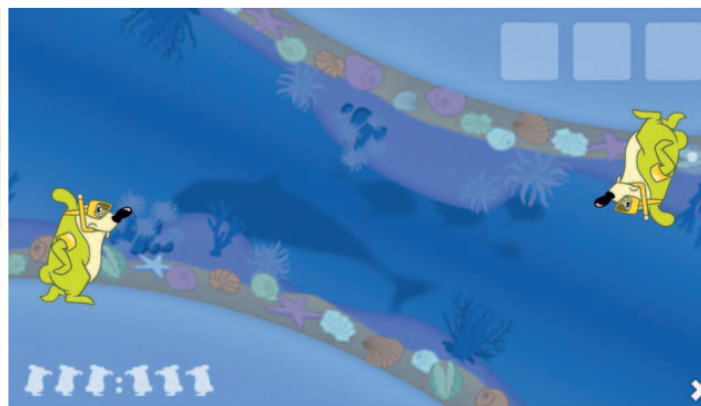
Bei der Planeten-Drehung haben wir einen **Parallax-Effekt** eingebaut. So dreht sich die Boden-Ebene schneller als die Landschafts-Ebene und diese dreht sich

wiederum schneller als die Hintergrund-Ebene. Hierbei entsteht der Eindruck, dass nach hinten die Entfernungen größer werden. Dies macht die **räumliche Illusion** perfekt.

Wüsten-Welt



Unterwasser-Welt



Wiesen-Welt



WELTEN DESIGN

Um den Farbflächen eine **leichte Struktur** zu geben, haben wir einen **Grunge-Effekt** hinzugefügt. Dieser betont sowohl die sandige Oberfläche in der Wüste und unter Wasser als auch den Rasen der Wiese. Zudem haben die Farbflächen noch einen **leichten Verlauf**, welcher zur oberen Kante hin dunkler wird. Dieser soll die **räumliche Illusion** noch verstärken.

Bei den Hintergründen wird soweit wie möglich auch **auf Konturen verzichtet**. Wenn es jedoch nicht ohne Kontur geht, wie zum Beispiel bei den Muscheln der Unterwasser-Welt, haben die Objekte **eine Kontur in dem Farbton der Farbfläche, nur etwas dunkler**.

Wüsten-Welt



Wiesen-Welt



Unterwasser-
Welt



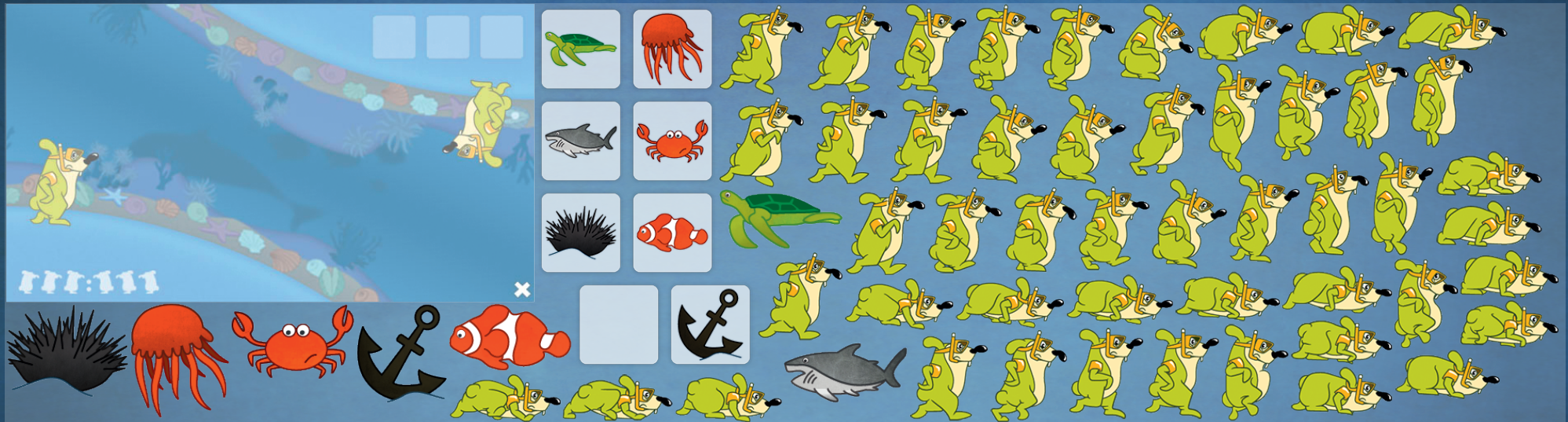
HINDERNISSE

Bei den Hindernissen gibt es zwei unterschiedliche Arten: **Boden- und Luft-Hindernisse**.

Die Motive passen sowohl sachlich als auch optisch zur jeweiligen Welt. Deshalb weisen alle größeren Farbflächen auch einen **Grunge-Effekt** auf und die **Kontu-**

ren haben immer den gleichen Farbton wie ihre Farbflächen, nur eben etwas dunkler.

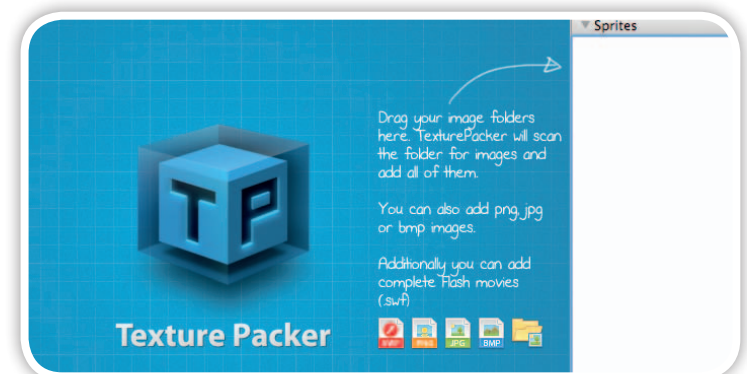
Damit die Hindernisse sofort ins Auge stechen, haben wir meist **Komplementär-Farben** oder zumindest eine starke Kontrast-Farbe verwendet.



SPRITESHEETS

Um Performance zu sparen, mussten wir die Bilder in Spritesheets verpacken. Insgesamt gibt es vier Spritesheets: Ein globales für Menü- und weltensübergreifende Bilder (z. B. Fehlermeldungen) und für jede der drei Welten eines. Anfangs machten wir es noch per Hand, was sich jedoch ziemlich schnell als unmöglich herausstellte. Deshalb haben wir uns eine

Lizenz von „Texture Packer“ gekauft. Nur dadurch konnten wir die Spritesheets ca. 50 mal ändern. Ein weiterer Vorteil ist die automatisch generierte JSON-Datei, welche durch den Befehl „bin/lime.py gensoy path/to/the/jsonFile.json“ (in der Konsole) in eine json.js-Datei konvertiert und dadurch von LimeJS ausgelesen werden kann.





Zum Soundtrack ist zu sagen, dass der latent **bayerische Einschlag** zur Geltung kommen sollte, aber auch die "Drolligkeit" der Spielfigur und Spielwelt. Im Hintergrund spielen **Blasinstrumente** (Tuba) und im Vordergrund spielt eine "lustige" **Flötenmelodie**, die dann durch ein **Xylophon** abgelöst wird. Für die Softwareumsetzung haben wir einen Mac mit Apples Logic Pro verwendet, der viele virtuelle Instrumente in seiner Bibliothek

hat. Den Soundtrack gibt es Grundsätzlich in zwei Tempovarianten: **Langsam**: Angepasst an das Menü, **Schnell**: Ungefähr an die Sprungfrequenz der Spielfigur angepasst. Sie sollte daher im Spiel verwendet werden.

● **Umsetzung in LimeJS**

Die Umsetzung in LimeJS ist leider gescheitert. Der Sound lies sich zwar einbauen, aber das Spiel ist in Folge dessen in verschiedenen Browsern abgestürzt. Allgemein ist die

Umsetzung von CrossPlattform-Sound auf mobilen Endgeräten keine einfache Sache.

● **Zukunftsaussichten**

Da eine alternative Umsetzung aus Zeitgründen nicht mehr zu schaffen war haben wir dennoch Wege recherchiert, wie man das fehlerhafte LimeJS umgehen kann. Dabei sind wir auf Sound.js gestoßen, das für eine Realisierung tauglich wäre: www.createjs.com/#!/SoundJS



Die Website wurde als simples **One-Page-Konzept** umgesetzt. Wir haben uns für diese simple Art der Umsetzung entschieden, da wir ohnehin nicht viele Inhalte zu kommunizieren haben. Jedoch wurde die Seite so angelegt, dass sie sehr leicht erweiterbar ist. Die Designstruktur besteht aus einem **2-spaltigen Grid-Layout**. Zwei Felder werden nebeneinander **bildschirmfüllend** angelegt, wobei eines dieser Felder

für Bildinhalte und das andere für Textinhalte gedacht ist. So ist jeder Menüpunkt bildschirmfüllend und bekommt zusätzlich eine bestimmte Farbe, was der Übersichtlichkeit und Strukturierung des One-Pagers dienlich ist. Wir haben beschlossen in der ersten Grid-Zeile einen **Pfeil** anzubringen, der signalisiert, dass weitergescrollt werden kann, da dies durch die bildschirmfüllenden Elemente für manche Menschen zu Missver-

ständnissen führen könnte. Bei den Hintergrundfarben haben wir **CSS3-Farbverläufe** verwendet, was die einzelnen Elemente weniger erscheinen lässt. Für die Website wurden auch zwei Grafiken erstellt, die das Spiel im Einsatz auf einem Mobile Device zeigen. So können auch Leute, die die Seite nur kurz überfliegen, auf den ersten Blick erkennen, dass es sich um ein Spiel für mobile Geräte handelt.



IDENT-VIDEO KONZEPT

Im Rahmen der Vorbereitungen für die Projektpräsentation im Stadttheater Augsburg, bat Prof. Dr. Michael Kipp die einzelnen Gruppen darum, ein Ident-Video zu erstellen. Diese sollten als **Werbung sowohl auf dem Webauftritt der Veranstaltung, als auch auf sozialen Netzwerken**, dienen. An den Schluss eines jeden Ident-Videos wurde von einem Studenten aus dem Organisationsteam der Veranstaltung Sichraum noch

ein „Sichraum-Packaging“ angehängt, das nochmal explizit auf Datum und Ort der Veranstaltung hinweist: www.youtube.com/watch?v=5WQp0lxvrWM. Das Video ist nach einem sehr simplen Schnittmuster aufgebaut und zeigt so immer **abwechselnd Animations-Film und Real-Film**. Der Einstieg erfolgt durch das typische **Obacht-Intro**, wie es auch beim Starten eines neuen Spiels gezeigt wird. Anschließend

wird immer mit einem **Statement** erklärt, was man im darauf folgenden Realfilmausschnitt sieht. So werden auf eine anschauliche Art die **Grundzüge des Spielekonzepts erklärt**. Kurz vor Ende wird explizit darauf hingewiesen, dass es sich um ein Multiplayer-Spiel handelt und in einer Grafik angedeutet, wie dies technisch funktioniert. Zu guter letzt wird das Logo, der Link zur Webseite, als auch der Link zur Facebook-Seite gezeigt.



IDENT-VIDEO UMSETZUNG

Bei der Gestaltung des Videos haben wir uns streng an das Obacht Corporate Design gehalten. Die **Statements**, die sich mit gefilmtem Video-Material abwechseln, wurden sehr **spielerisch** animiert, angelehnt an die Bewegungsabläufe unseres Hogi-Characters. So wird von den Bewegungsabläufen der Statements eine Brücke zu den gezeigten Spielszenen aufgebaut.

Für das **Realfilm-Material** hatten wir auf das Smartphone ein grünes Papier als **Green-Screen** aufgeklebt, auf dem **zwei Trackingpunkte** angebracht waren. So konnte, nach einem Trackingprozess, die Spielsequenz einfach auf das Videomaterial gelegt werden und folgte somit den Bewegungen des Gerätes. Die Finger, die vor dem Bildschirm zu sehen sind, mussten al-

lerdings von Hand rotoskopiert werden. Die **Spielszenen** wurden **von Hand animiert**, da hier bestimmte Spielaktionen in guter Auflösung benötigt wurden. Dies wäre auf anderem Wege nicht möglich gewesen. Zum Schluss wurde der Clip noch mit der **Musik** unterlegt, die wir für das Projekt gekauft hatten. Sie war ursprünglich für die Implementierung in das Spiel gedacht.



VERSIONSKONTROLLE

Bei der Versionskontrolle haben wir uns für „Git“ entschieden. Speziell im Bezug auf unsere Teamgröße und die Wahrscheinlichkeit, dass mehrere Leute an derselben Datei arbeiten, ist Git einfacher zu bedienen als SVN. In Verbindung mit GitHub konnten wir unser öffentliches Repository auf allen von uns benutzten

Betriebssystemen (Win, Linux und Apple) koordinieren und eine funktionierende Versionskontrolle aufbauen.

Anfangs gab es in der Einarbeitungszeit ein paar Bedienungsprobleme, die aber sehr schnell mit der stetigen Verwendung von Git gelöst bzw. vermieden wurden.



WebStorm



Aptana

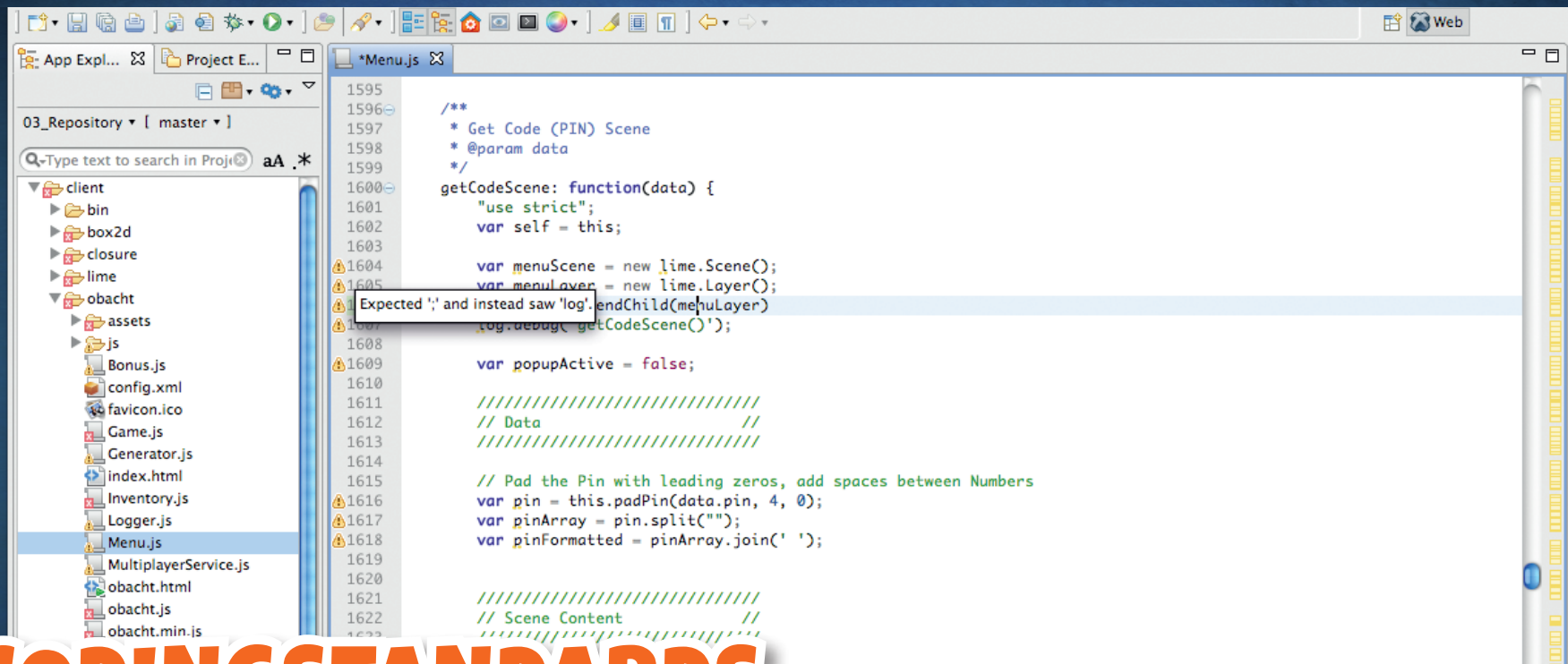


Sublime Text 2

ENTWICKLUNGSUMGEBUNGEN

Zu Beginn haben wir uns nicht auf eine IDE (Entwicklungsumgebung) festgelegt. Nach einiger Zeit konnten wir dank Herrn Professor Kowarschick eine **Studentenlizenz von „JetBrains WebStorm“** über die Hochschule Augsburg besorgen.

Diese IDE wurde dann von den meisten Programmierern benutzt. Zusätzlich waren die IDE **„Aptana“** und der Editor **„Sublime Text 2“** in Gebrauch.



CODINGSTANDARDS

● Allgemeines

Am Anfang der Projektphase haben wir uns als Team auf Coding-Standards geeinigt und diese in einer Präsentation vorgestellt und in unserem Wiki dokumentiert (s. Anhang: CodingStandards.pdf)

● Linter

Um die Coding-Standards einfacher durchzusetzen und auch um die Codequalität besser zu sichern wurde JSHint (www.jshint.com) verwendet. Ausnahmeregelungen werden im Code am Anfang der jeweiligen Datei als Kommentar angegeben.

● JSDoc

Die Kommentare wurden im JSDoc 3 Standard (www.usejsdoc.org) geschrieben. Im Unterordner /docs/ findet sich eine automatisch generierte Dokumentation des Projekts. Der Code ist generell so geschrieben, dass er möglichst aus sich selbst heraus verstanden werden kann.

HTML



WAHL DER PLATTFORM

Es wurde zwischen nativer Entwicklung, der Verwendung von Hybrid-Engines und HTML5, also plattformunabhängig, abgewägt:

- **Native App-Entwicklung** (Android oder iOS). **Pro:** Alle Features, performant, relativ übersichtlich. **Kontra:** Nicht Cross-Plattform: App muss für andere Systeme neu geschrieben/konvertiert werden.

- **Hybrid:** Game Engines mit „Nativ-Buildern“ (z.B. Unity). **Pro:** Viele Features, per-

formant, relativ übersichtlich. **Kontra:** Kostet immer irgendwo Geld, bindet an die Engine

- **HTML5** (cross platform): **Pro:** Entwickelt sich rasant, plattformunabhängig (auch für zukünftige Plattformen), Open Source. **Kontra:** Features oft noch eingeschränkt, nicht immer beste Performance, kann schnell unübersichtlich werden.

Die Entscheidung fiel auf HTML5, allerdings tendierten wir auch stark zu Unity. Ein

Grund für HTML5 war, dass es sich aktuell stark weiterentwickelt und der Trend in diese Richtung geht. Auch hatten die einzelnen Gruppenmitglieder sehr unterschiedliche Geräte in Gebrauch und so schlossen wir niemanden aus. Zusätzlich lässt sich das erlernte Wissen sehr gut auf andere Projekte/Bereiche anwenden, da HTML5/JavaScript in allen möglichen Gebieten/Anwendungen eingesetzt wird.

LimeJS is a **HTML5 game framework** for building fast, native-experience games for all modern touchscreens and desktop browsers.

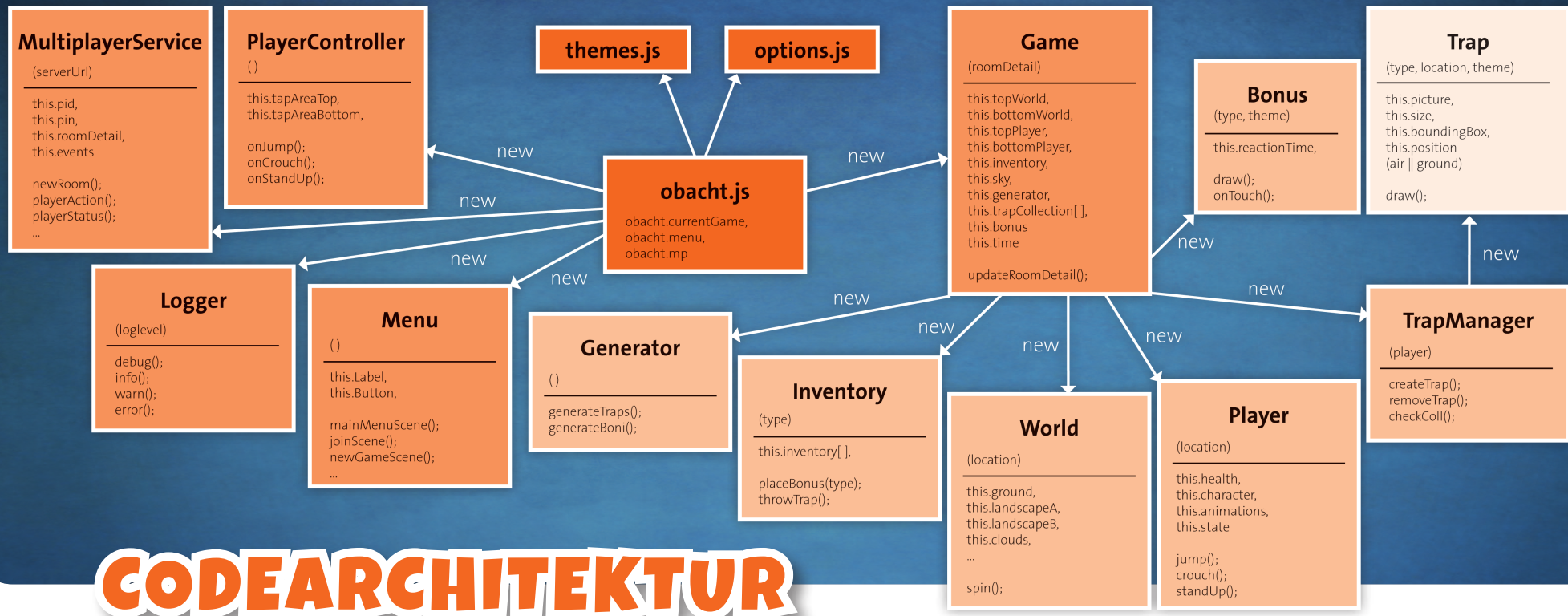
WAHL DER ENGINE

In die nähere Auswahl kamen für uns die HTML5-Game-Engines „ImpactJS“, „LimeJS“ und „Crafty“. Wir sahen uns die Engines grob an, wobei wir uns bei der Abwägung der Vor- und Nachteile auf die Angaben der Entwickler verlassen muss-

ten, da keiner aus unserem Team schon tiefere Erfahrungen mit Game-Engines hatte. **Wir entschlossen uns einfach dazu „LimeJS“ zu verwenden** und dann bei der Umsetzung zu sehen, ob sie unseren Ansprüchen gerecht wird.

● Installation

Eine Installationsanleitung von LimeJS findet sich in der offiziellen Dokumentation: <http://www.limejs.com/0-getting-started>



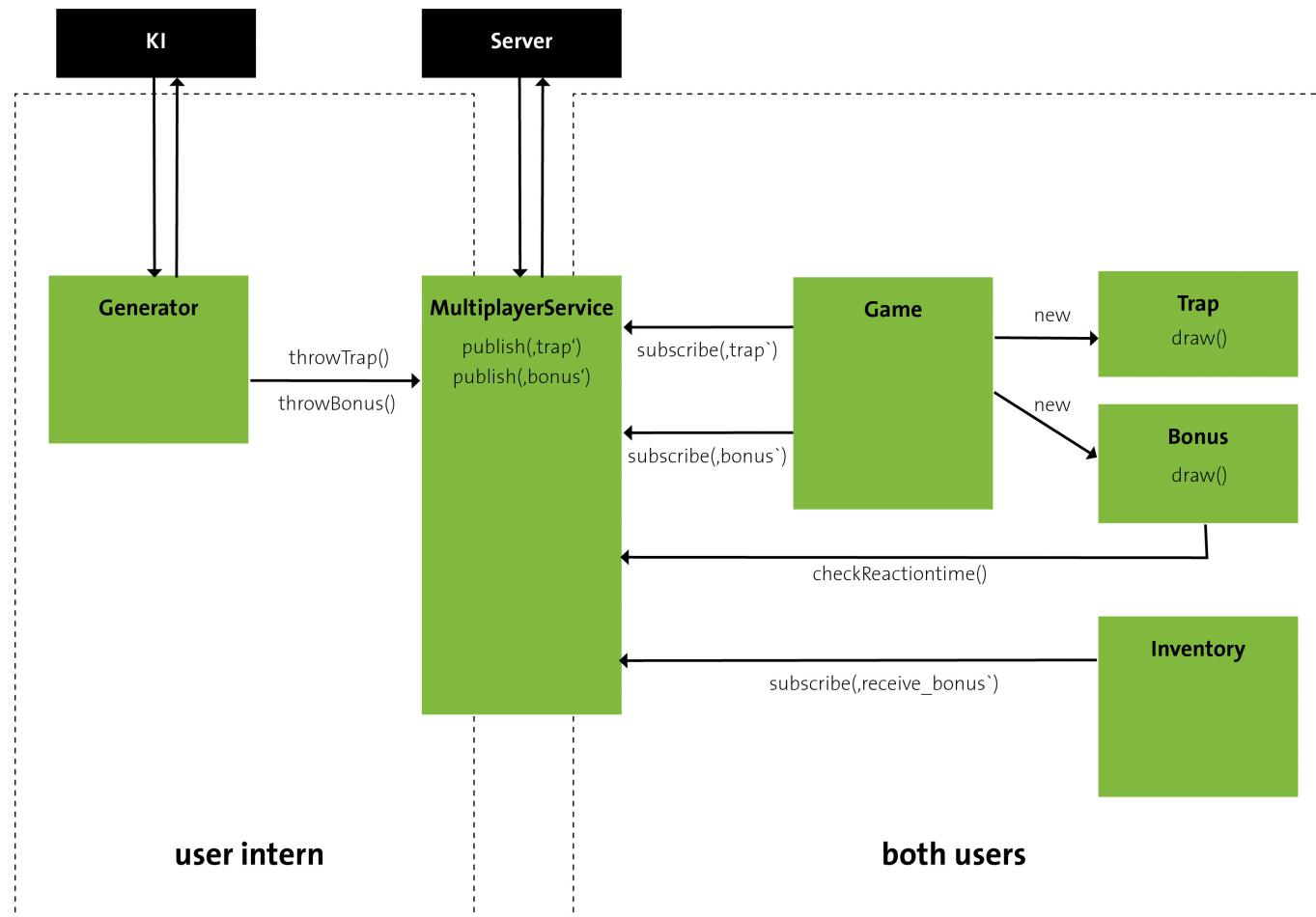
Der Client ist modularisiert in einzelne Klassen aufgeteilt. Das Modulhandling wird von der Google Closure Library gestellt, die von der LimeJS-Engine auch verwendet wird. So ließ sich der Code gut abstrahieren und aufteilen.

Hier eine visuelle Darstellung der verschiedenen **Klassen und deren Abhängigkeiten**. (siehe Klassendiagramm oben)
Das Menü und das Spiel sind allerdings stark event-basiert. Daher wurde das Publish/Subscriber-Pattern verwendet um

User-Events und Server-Events zu verwalten. Die Beziehungen und Abhängigkeiten sind hier weitestgehend asynchron und meist klassenübergreifend. (Diagramme siehe nächste Seite)

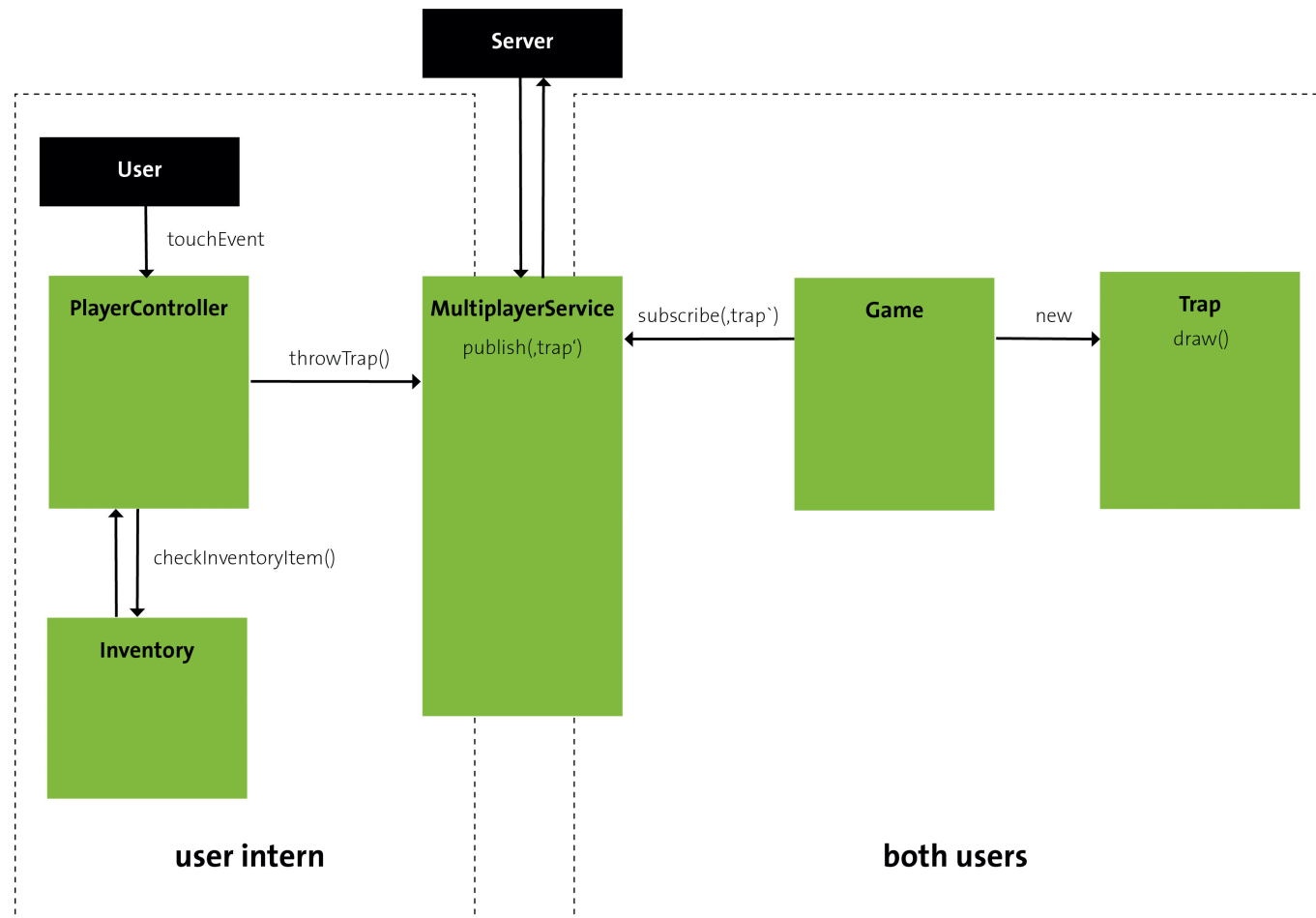
EVENTDIAGRAMM

Generator generates new traps and boni



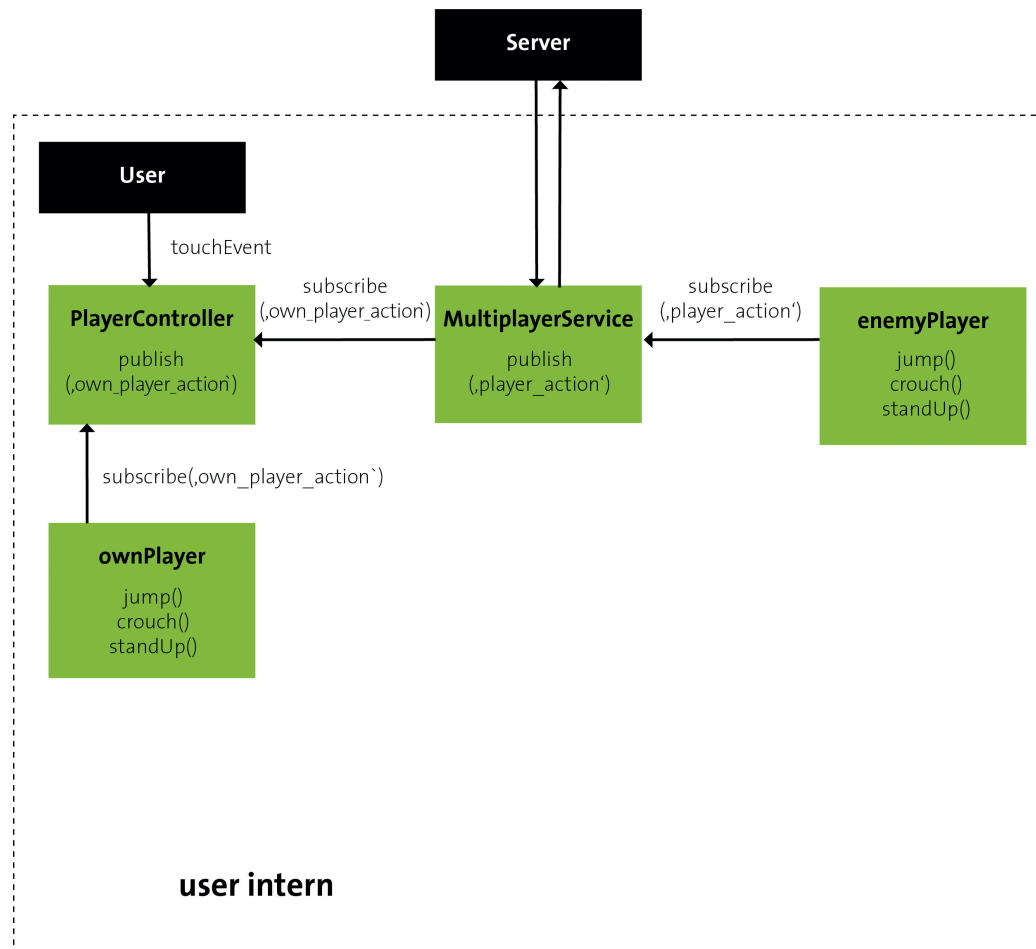
EVENTDIAGRAMM

Player uses inventory item



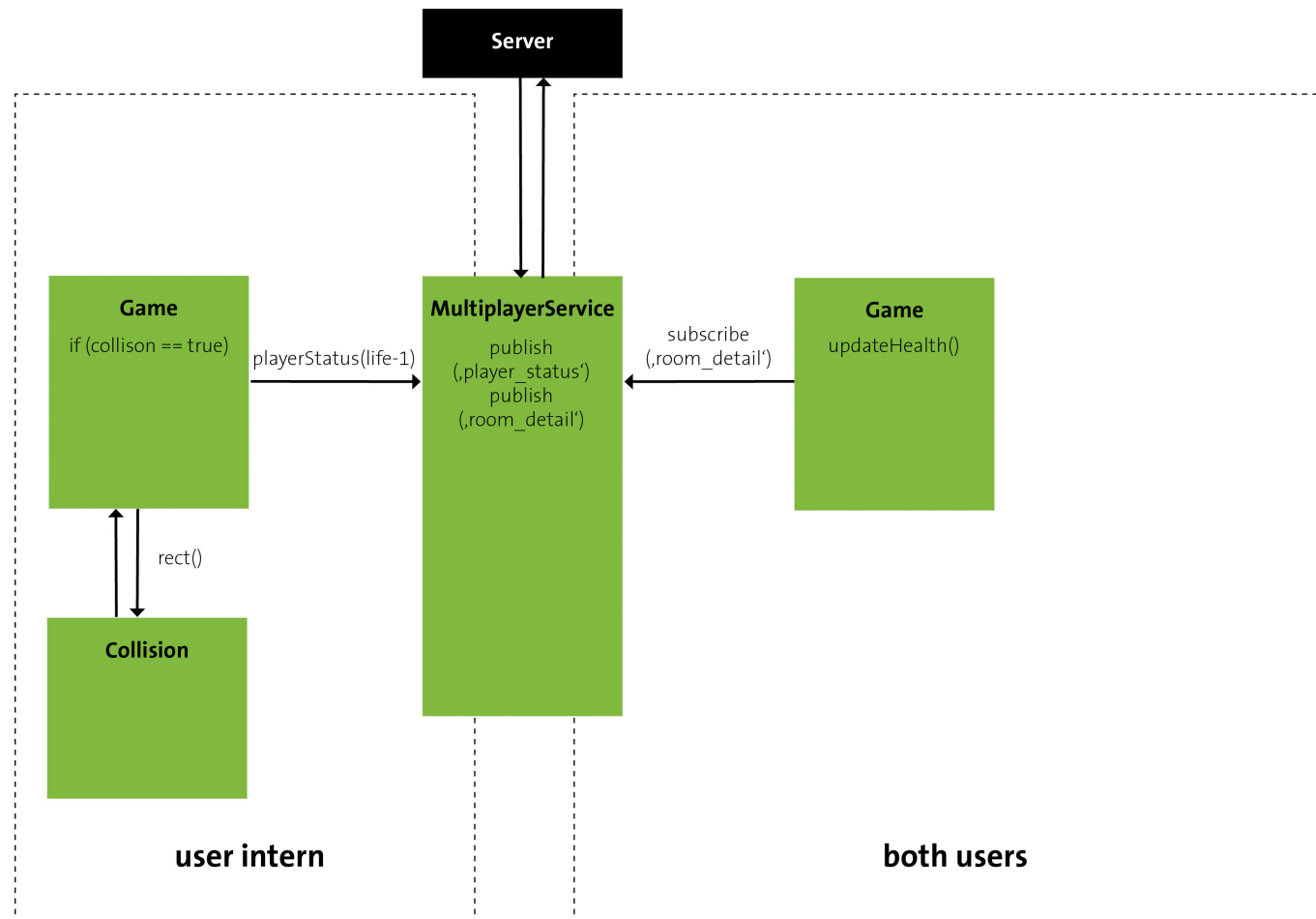
EVENTDIAGRAMM

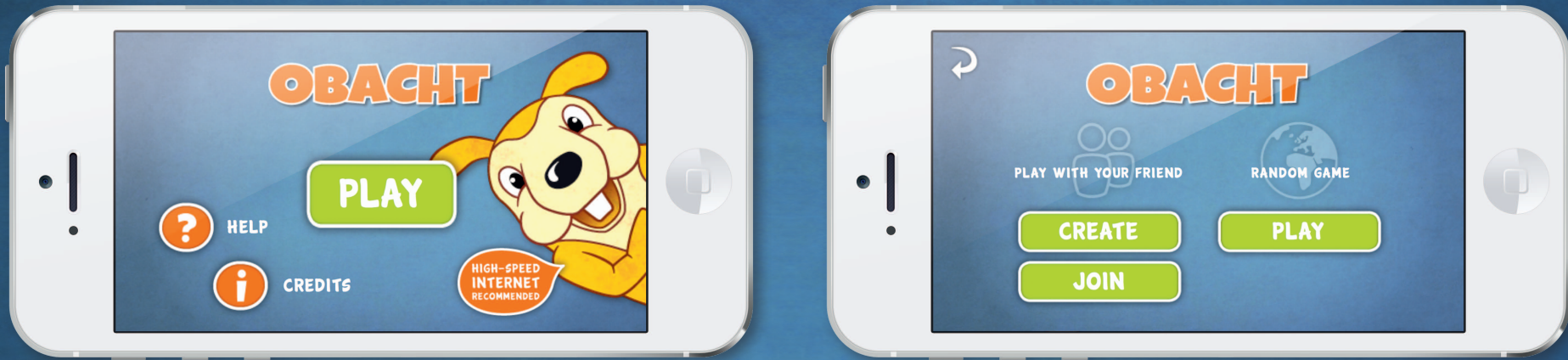
Player actions



EVENTDIAGRAMM

Collision and health status





MENÜ AUFBAU

● Allgemein

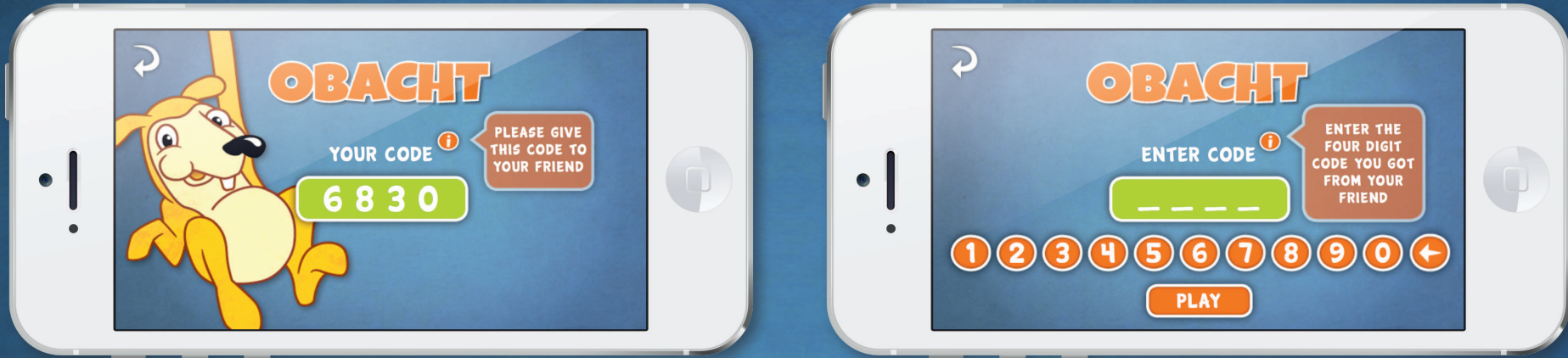
Ausgehend von den Use-Cases wurde das Menü mit LimeJS (Scenen, Layer) gebaut. Jeder Button besitzt einen Listener auf ein Touch- oder ein Mousedown-Event. Wird dieser getriggert wird die entsprechende nächste Scene geladen. Vor dem Laden können Events und Subscriptions abgefragt/gesetzt werden, um eine komplexere Logik widerspiegeln zu können.

● Hauptmenü

Die erste Scene stellt das Hauptmenü dar. Von hier hat man die Möglichkeit sich eine Anleitung „HELP“ anzusehen. Diese erklärt auf einigen Seiten alle wichtigen Spielelemente und die eigentliche Steuerung des Spiels. Unter dem Punkt „Credits“ findet man alle Mitwirkenden des Projektes. Zur Spielübersicht kommt man über den größten sichtbaren Button – „Play“.

● Spielübersicht

Die erste Spalte ist für ein Spiel mit einem Freund, mit dem man bereits in Kontakt steht. Über „Join“ kann man einem Spiel, das bereits von einem Freund gestartet wurde, beitreten. Wenn man gerade nicht mit einem Freund spielen kann, kann man über den Play-Button mit einer zufällig ausgewählten Person spielen. Über den weißen Pfeil in der linken oberen Ecke kommt man zurück ins Hauptmenü.



MENÜ AUFBAU

● Codeausgabe (durch CREATE)

Sobald der PIN angezeigt wird, befindet sich der Spieler im „Player ready“-Modus. Die PIN muss dann an den Freund weitergegeben werden.

● Codeeingabe (durch JOIN)

Die vom Freund erhaltene PIN muss hier eingegeben werden. Befindet sich der Freund immer noch auf der Code-Ausgabe-Seite wird beim Klick auf „Play“ das Spiel gestartet.

● Randomspiel (durch PLAY)

Hier gibt es zwei Möglichkeiten. Entweder ist kein Randomspiel vorhanden – dann wird ein neues erstellt und auf einen zweiten Spieler gewartet. Oder es ist bereits ein Spiel erstellt worden – dann wird der Spieler dem bereits wartenden Spieler zugeordnet und das Spiel wird gestartet.



MENÜ AUFBAU

● Spielwiederholung

Ist ein Spiel zu Ende, bekommen beide Spieler eine Meldung, ob sie gewonnen oder verloren haben, und die Möglichkeit über den Button „Play again“ erneut ein Spiel zu starten. Hier muss unterschieden werden ob das Spiel ein Freundspiel oder ein Randomspiel ist. Bei einem Randomspiel wird der Raum verlassen und ein erneutes Randomspiel gestartet bzw. einem bereits bestehendem Randomspiel

beigetreten. Bei einem Freundspiel verlassen beide Spieler den Raum, werden danach automatisch sofort gemeinsam einem neuen Raum zugeteilt. Das Theme wird automatisch zugeteilt und das Spiel gestartet.

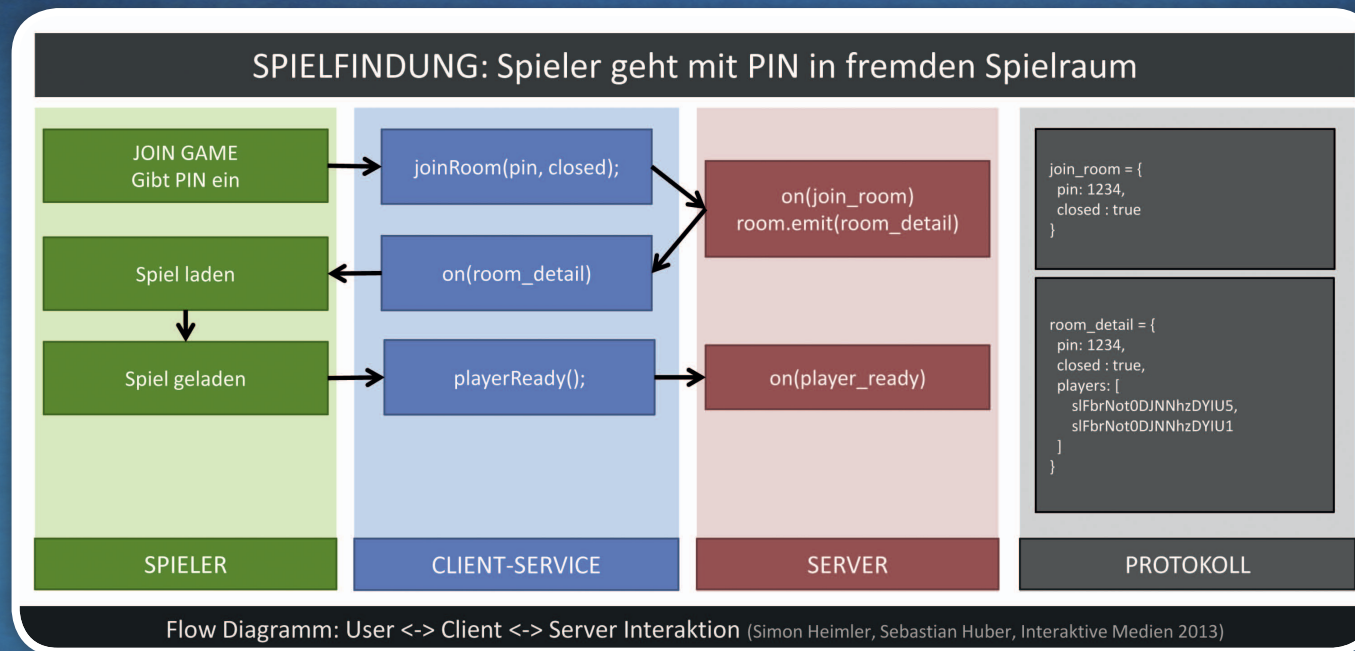
● Popups

Um dem Spieler Rückmeldungen (Fehler, Infos, usw.) geben zu können, wurden global Popup-Meldungen eingeführt. Diese bestehen aus einem einheitlichen Grundgerüst und der Mel-

dung, die automatisch vom Client, Multiplayer-Service oder Server übergeben wird. Auslöser wäre z. B. der Verbindungsverlust zum Server.

● Fazit

Im Nachhinein wäre es sinnvoller gewesen das Menü in HTML5 zu schreiben. Dies würde die Performance verbessern und eine Menüerweiterung erleichtern. Aus zeitlichen Gründen wäre dies allerdings nicht mehr zu realisieren gewesen.

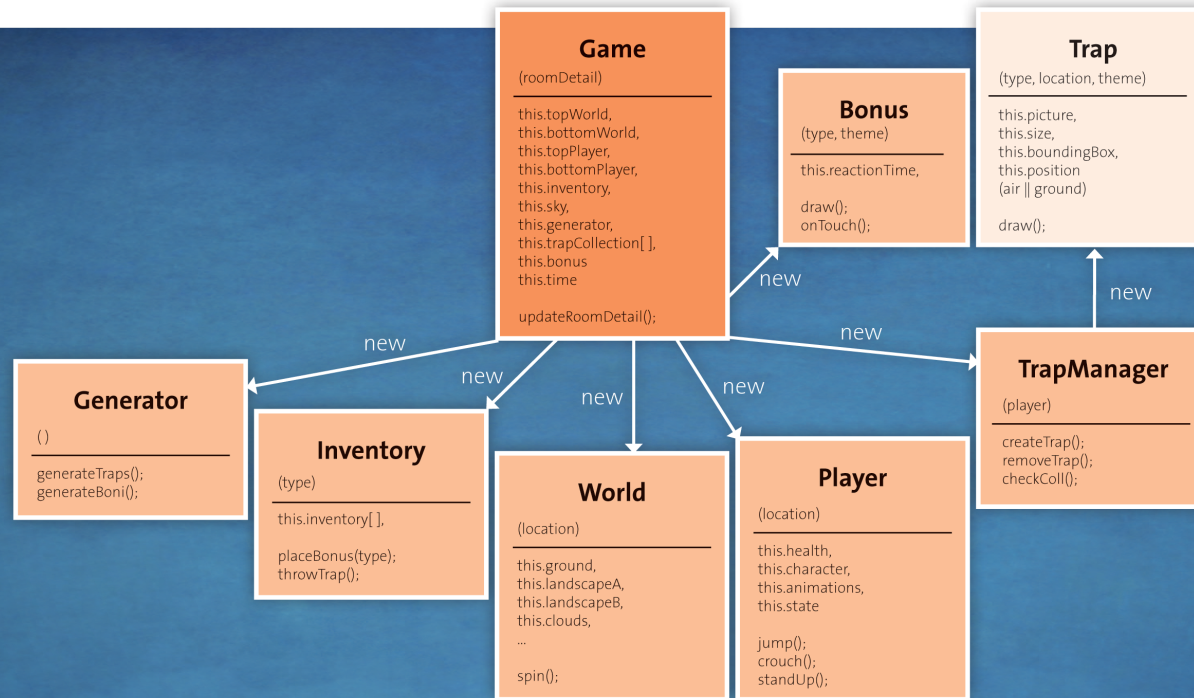


MULTIPLAYER-SERVICE

Der Multiplayer-Service stellt die Verbindung zwischen dem Client und dem Multiplayer-Server dar. Er beinhaltet das Kommunikationsprotokoll der Clientseite und registriert Funktionen und Events. Benutzt werden hierfür Socket.io und die Verbindung zum Node.js-Server.

Die Multiplayer Events werden über ein Publish/Subscriber-Pattern an das Menü und das Spiel zur Verfügung gestellt. Der Multiplayer-Service fungiert hier als ein Interface, das eine möglichst einfache Schnittstelle nach außen hin frei gibt und tiefere Logik versteckt.

Um gleichzeitig am Server, dem Client und der Kommunikation zu arbeiten, wurde der Kommunikationsablauf in einem Protokoll beschrieben. Somit wurde eine öffentliche Richtlinie bereit gestellt, an der sich die Programmierer orientieren konnten.



GAME (UND UNTERKOMPONENTEN)

Für jedes Spiel wird eine Instanz von der Klasse **Game** und des **Player-Controllers** aufgebaut. Das Game verwaltet die Grundfunktionen des laufenden Spieles. Der PlayerController legt einen unsichtbaren Layer über das Game, der auf die Touchgesten des Users reagiert. Die Spiele-Instanz verwendet folgende Klassen:

- **Player:** Zwei Player-Instanzen für die Darstellung, Aktionen und Daten der beiden Spieler.
- **World:** Zwei Welten-Instanzen

- **Generator:** Läuft nur auf dem Client, der das Spiel erstellt. Erstellt zufällig neue Hindernisse und Boni, die erst an den Server geschickt und von dort an beide Clients verteilt werden.
- **TrapManager:** Erstellt Trap Objekte entsprechend der eingehenden Events. Kümert sich um die Kollisionserkennung auf der eigenen Welt. (Siehe nächste Seite)
- **Trap:** Hindernisobjekt von TrapManager
- **Bonus:** Bonus-Button, der bei beiden Spie-

lern gleichzeitig auftaucht und der bei Tap die Geschwindigkeit der Reaktion misst. Die Reaktionszeit wird an den Server geschickt. Dieser gibt zurück, wer den Bonus bekommt.

- **Inventory:** Sobald ein Bonus gewonnen wurde, wird er in das Inventar eingetragen. Wenn dort ein gewonnener Bonus berührt wird, wird der Bonus zum Server gesendet, wodurch dem Gegner ein Trap in den Weg gestellt wird.



Schafft es der User nicht, dem Hindernis auszuweichen, so erscheint eine Staubwolke und der User verliert ein Leben.



KOLLISIONSERKENNUNG

Die Prüfung einer Kollision zweier achsenausgerichteten (nicht rotierenden) Rechtecken kann sehr effizient durchgeführt werden. Es wird im Obacht Game also zuerst gecheckt, ob zwei Objekte nah genug aneinander sind, damit eine Kollision möglich ist. Dann testet man die sie umgebenden Rechtecke (**Bounding Boxes**) auf Kollision.

Wenn mindestens eine Kante eines Rechtecks ein anderes Rechteck überschneidet, dann liegt eine Kollision vor. Es werden die Positionen der einzelnen Eckpunkte der Rechtecke benötigt. Wichtig beim erfassen der Eckpunkte ist die Tatsache, dass LimeJS eine etwas gewöhnungsbedürftige Ausrichtung der Achsen hat.

Der Player besitzt selbst nur eine BoundingBox für die Kollision, die **bei Sprung und Ducken mitbewegt und -skaliert** wird. Die Hindernisse (Traps) können aus beliebig vielen Bounding-Boxen bestehen. In der options.js kann unter options.debug.showBoundingBoxes die Darstellung der BoundingBoxen zum testen aktiviert werden.



SERVER

● Wahl der Technologien

Am Anfang des Projektes musste die Entscheidung getroffen werden, wie die Multiplayer-Kommunikation zwischen den Geräten hergestellt werden soll. Dazu haben wir mehrere Technologien miteinander verglichen und Prototypen angefertigt, um Machbarkeit und Performance zu messen und einzuschätzen.

Insbesondere wurden **Bluetooth, Websockets und WLAN-AdHoc miteinander verglichen**. Die Entscheidung viel auf **Websockets**, da es die einzige Technologie ist, die komplett plattformunabhängig funktioniert und die Umsetzung des Prototyps hier schon gute Ergebnisse erzielte. Die Wahl ist also auf **Node.js** (www.nodejs.org) als Grundlage für den Server ge-

fallen. Um die Datenstruktur (das Model) zu verwalten wurde das ebenfalls weit verbreitete MV* Framework **Backbone.js** (www.backbonejs.org) verwendet.

● Installation

Voraussetzung für die Installation des Servers ist eine Node.js-Installation. (<http://nodejs.org>). Alle wichtigen Abhängigkeiten sind in der Datei `/server/package.json`



SERVER

erfasst und können mit dem **Node Package Manager npm** automatisch installiert werden: Im Ordner /server/ muss dafür der Befehl „npm install“ ausgeführt werden. Updates per „npm update“. Gestartet wird der Server mit dem Befehl „node obacht-server.js“. Auf einem Produktionsserver empfiehlt sich der Einsatz des Node Modules Forever (<https://github.com/nodejitsu/forever>), das ab-

gestürzte Prozesse automatisch überwacht und neustartet. In dem BashScript „server/update.sh“ wird Forever verwendet. Anmerkung: Unter Windows muss eventuell Visual Studio (Express) installiert sein, damit das Modul Socket.io erfolgreich gebuildet werden kann.

● Codearchitektur

Grundsätzlich ist der Server mit Node.js umgesetzt. Dadurch ergebe sich besonders

zwei Paradigmen: **Persistent:** Der Server läuft durchgehend, muss also auch bewusst gestartet und beendet werden. Dadurch kann der Server alle wichtigen Daten auch im Arbeitsspeicher halten und muss nicht bei jeder Verbindung neu aufgebaut werden. Eine Datenbank ist daher nicht nötig. Das macht die Anwendung für Echtzeitkommunikation sehr effizient.



SERVER

Asynchron: Alle Funktionen müssen asynchron (event-driven) programmiert werden, d.h. dürfen die Ausführung nicht „blockieren“. Dieser Ansatz funktioniert für ein Kommunikationsprotokoll sehr gut, da dieses naturgemäß auch event-driven ist. Allerdings kann asynchrone Programmierung schnell unübersichtlich und kompliziert werden und leicht Situationen entste-

hen lassen, die sich gegenseitig stören oder bedingen. Der Server ist hauptsächlich in `obacht-server.js` (Einstiegspunkt, Kommunikationshandling) und `RoomManager.js` (Verwaltung der Spielräume) unterteilt.

● **Socket-Kommunikation**

Zur Kommunikation mit den Spieleclients wird das Node Modul `Socket.io` (www.socket.io/) verwendet. Dieses verwendet HTML5

Websockets, um persistente Verbindungen zwischen Client und Server aufzubauen.

● **Raumverwaltung**

Um die Spielräume komfortabel verwalten zu können, wurde die Klasse `RoomManager` geschrieben. Diese beinhaltet im wesentlichen eine `Room Collection`, die das Auf- und Abbauen, aber auch verwalten von Räumen, unterstützt.



WEBSITE

WWW.OBACHT-GAME.COM

Umgesetzt wurde die Website mit HTML5, CSS3 und JavaScript mit Zuhilfenahme der JavaScript-Bibliothek jQuery (www.jquery.com). Die **bildschirmfüllende Anordnung** der einzelnen Seitenbestandteile erfolgt durch das Auslesen der Bildschirmgröße beim Seitenaufruf und wenn die Fenstergröße vom Betrachter geändert wird. Anschließend werden die Elemente des Grids neu berechnet und angepasst dargestellt. Die Schrift, die wir

auch in unserem Spiel verwenden, haben wir über **Google Fonts** (www.google.com/fonts) eingebunden. Bevor unser CSS auf die Seite angewandt wird, erfolgt eine Normalisierung, um eine annähernd gleiche Darstellung in verschiedenen Browsern zu gewährleisten. Zusätzlich haben wir „The HTML5 Shiv“ (<https://github.com/aFarkas/html5shiv>) eingebunden, das HTML5-Bestandteile der Seite **auch für ältere Browser darstellbar** macht.

Der Pfeil auf dem ersten Element der Seite wird mittels jQuery animiert und angezeigt, wenn sich die Seite im völlig nach oben gescrollten Zustand befindet. Der Zustand lässt sich mittels des Scroll-Events, das jQuery zur Verfügung stellt, auslesen. Desweiteren gibt es **auch noch eine mobile Version** der Website in abgespeckter Form. Diese ist für Screen Größen unter 1000px konzipiert und wurde via CSS Media Query realisiert.

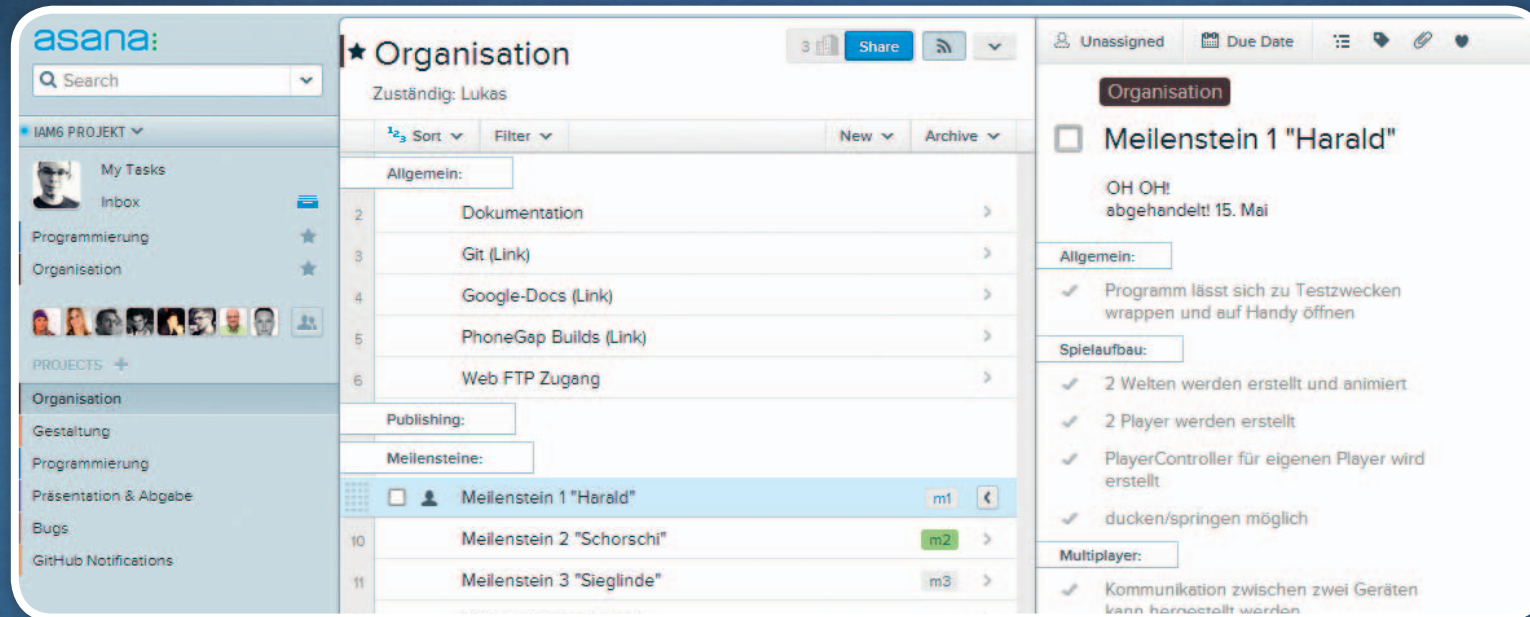


AGILE ENTWICKLUNG

Wir orientierten uns am Ansatz der agilen Entwicklung. Schon in frühen Projektphasen fertigten wir **Skizzen und Prototypen** an, die ein erstes Bild von der Applikation und der Reaktionsfähigkeit geben sollten. Es wurde während des Projektverlaufs an fast allen Bereichen gleichzeitig gearbeitet, so dass diese iterativ gewachsen

sind. Wir hatten daher schon relativ früh Ergebnisse die man „fühlen“ konnte. Das ermöglichte uns ein **frühzeitiges Eingreifen**, wenn sich etwas nicht nach unseren Vorstellungen entwickelt hatte. In den **Teammeetings, die immer mittwochs stattfanden**, wurden alle konzeptionellen und organisatorischen Dinge bespro-

chen und die Aufgaben für die nächste Woche verteilt. Im Rahmen der Teammeetings erfolgte auch die **Rücksprache mit den Betreuern**.



AUFGABENVERWALTUNG

Für die Aufgabenverwaltung verwendeten wir das webbasierte Anwendung „Asana“ (www.asana.com). Damit ließen sich sehr schnell und unkompliziert Aufgaben anlegen, zuweisen und die Kommunikation zwischen den beteiligten

Teammitgliedern abwickeln. Durch eine API Verbindung mit Github war es auch möglich Git-Commits einer bestimmten Asana-Task-ID zuzuweisen und automatisch verlinken zu lassen.

Meilensteine:	
☐ 	Meilenstein 1 "Harald"
10	Meilenstein 2 "Schorschi"
11	Meilenstein 3 "Sieglinde"
12	Meilenstein 3.5 "John"
13	Meilenstein 4 "Eckbert"
☐ 	Meilenstein 5 "Resl"
15	Meilenstein 6 "Brunhilde"

MEILENSTEINE

Um das Projekt zeitlich überschaubar zu planen, hat sich das Team Meilensteine gesetzt. Die Meilensteine wurden so angelegt, dass am Ende des Semesters noch genügend Puffer blieb, um notfalls noch nachbessern zu können.

● **MS 1 (15. Mai 2013):** Konzeption abgeschlossen, zwei Geräte lassen sich verbinden, Welten und Player werden erstellt, Spiel lässt sich zu Testzwecken wrappen.

● **MS 2 (29. Mai 2013):** Funktionierende Server-Kommunikation, finales Design der Welten, vollständiges Design aller Hindernisse, finale Version des Charakters, fehlerfreie Kollisionserkennung

● **MS 3 (12. Juni 2013):** Vollständig implementiertes Menü, Charakteranimationen abgeschlossen, Spieleraktionen funktionieren fehlerfrei, Grafiken optimiert und in Spritesheets eingebunden, Website online

● **MS 4 (26. Juni 2013):** Vollständige App, Code stabilisiert, Spielbalancing abgeschlossen, Performance Optimierung abgeschlossen

● **MS 5 (3. Juli 2013):** Testting abgeschlossen, Bugfixing abgeschlossen

● **MS 6 (10. Juli 2013):** Dokumentation und Abgabe abgeschlossen, Präsentation abgeschlossen



Da das komplette Team von Beginn des Projektes an voller Tatendrang, Ideen und Motivation war, waren zwar eigentlich keine Teambuilding- und Motivations-Tätigkeiten nötig. Jedoch gingen wir trotzdem

in unserer Freizeit gemeinsamen Unternehmungen nach, um das Gruppengefühl noch mehr zu stärken.



MARKETING

Um auf das Projekt und das Spiel aufmerksam zu machen, entschieden wir uns eine **Facebook-Seite** einzurichten, die auch als Kommunikation der Gruppe nach außen verwendet wurde. Außerdem betreiben wir eine kleine Homepage, auf der alle wichtigen Fakten und Links des Spieles zusammen getragen wurden.

Zusätzlich haben wir **Flyer im Visitenkartenformat** drucken lassen. Diese enthalten neben den Fakten auch QR-Codes zu den App-Stores von Apple und Google. Für Präsentationen und Ausstellungen haben wir **Roll-ups und T-Shirts** entworfen, um die Marke „Obacht“ besser und nachhaltiger zu präsentieren.



S2INTERMEDIA

FINANZIERUNG

Um benötigte Software und Werbemittel zu finanzieren haben wir beschlossen, uns zusätzlich zum Grundbudget der Hochschule mit Sponsorengeldern abzusichern.

Die Internetagentur **S2 Intermedia** aus München hat das Projekt zusätzlich mit 250 € bezuschusst. So hatten wir ein **Gesamtbudget von 650 €**, dem Ausgaben in Höhe von ca. 600 € gegenüberstehen.

USABILITY



Der Usability Test wurde am 18 Juni 2013, im Testlabor der HS Augsburg, durchgeführt. Hierbei sollte die Handhabung und Verständlichkeit des Spieles durch Personen getestet werden, die das Spiel noch nicht kannten. Dabei wurden die Tests in Spielmenü und Gameplay unterteilt.

Für das Spielmenü wurde getestet ob:

- das Menü intuitiv, oder umständlich ist
- es Logik- oder Denkfehler gibt

- es unklare Punkte gibt, die zur Verwirrung der Testperson führen
- das Starten und Erstellen eines neuen Spieles verständlich ist
- das Starten und Beitreten eines vorhandenen Spieles verständlich ist
- das Tutorial verständlich ist bzw. ob dieses überhaupt durchgelesen wird und wie lange die Testpersonen für jede Aufgabenstellung benötigen.

Für das Gameplay wurde getestet ob:

- die Spielsteuerung intuitiv oder schwierig ist
- das vorhandene Tutorial zum Verstehen der Spielmechanik ausreicht
- es klar wird wie „Bonus-Items“ gesammelt werden?
- es klar wird wie „Bonus-Items“ verwendet werden?

Dabei wurden sowohl Einzeltests, als auch Zweiertests durchgeführt, bei de-



USABILITY

nen zwei Testpersonen gegeneinander Spielen durften. Es wurden **Testszenarien** erstellt, die möglichst alle Use Cases des Spieles abdecken. Als Testmethoden wurde lautes Denken, mit gleichzeitiger Bild und Tonaufnahme gewählt. Zusätzlich dazu machte ein stiller Beobachter Notizen. Auf das anfänglich geplante Eye Tracking wurde aufgrund der kleinen Displaygröße verzichtet.

Getestet wurde eine lauffähige Beta, in der jedoch nicht alle Animationen enthalten waren. Am Ende der Testszenarien wurde ein kurzer **Bewertungsbogen** ausgefüllt, um einen groben Notenschnitt der einzelnen Bereiche des Spieles zu erkennen, sowie bei Zustimmung noch ein gefilmtes, jedoch lockeres **Interview** bei Sonnenschein im Freien gehalten.

Als Belohnung gab es für die Testpersonen Muffins und Getränke. Die Auswertungen wurden noch vor Ort begonnen. Unter anderem wurde dabei auch ein Best-of-Video zusammengeschnitten, welches später beim Herrn John vorgestellt wurde.



Vor- und Nachteile

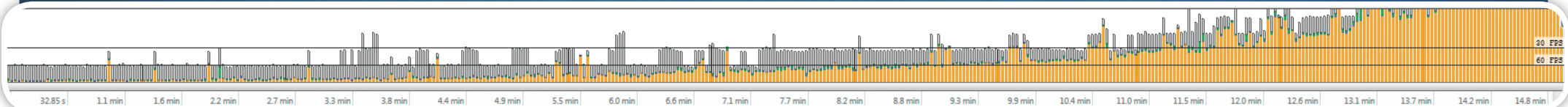
WAHL DER PLATTFORM & GAME-ENGINE

Die Entscheidung für **HTML5** als Plattform halten wir für gut, auch wenn sich gezeigt hat dass CrossPlattform Entwicklung doch einigen Mehraufwand zu nativer Entwicklung hat. Die Performance kann sehr viel schlechter sein als die von nativen Apps, muss aber nicht (s. AngryBirds). **LimeJS** hat uns am Anfang noch recht gute Dienste geleistet, allerdings wurde erst später klar dass die Performance besonders auf Android Geräten sehr zu wünschen übrig lässt, und

dass daran nur kaum etwas zu ändern ist. Dies liegt an der Entscheidung, hauptsächlich DOM Rendering zu verwenden und an einer relativ langsamen Entwicklung die sich bisher kaum auf bessere Android Kompatibilität konzentriert hat. Die Dokumentation ist eher dünn und die Engine wird nur von wenigen Leuten weiterentwickelt. Allerdings wurde sehr schnell auf Issues in GitHub reagiert und der Hauptentwickler zeigte sich sehr hilfsbereit. Im Nachhinein haben

wir den Wrapper Cocoon.js (www.ludei.com/tech/cocoonjs) entdeckt, der anders als PhoneGap speziell für Spiele entwickelt wurde und die Performance von HTML5 Spielen deutlich verbessern kann. Notwendig ist dafür aber eine GameEngine die Cocoon.js unterstützt, was bei LimeJS leider nicht gegeben ist. Rückblickend hätten wir also besser mit Cocoon.js als Grundlage angefangen und eine Game Engine anhand der Kompatibilität zu dem Wrapper ausgesucht.

Verwendeter Arbeitsspeicher, Arbeitsweise des GarbageCollectors: Keine Leaks erkennbar



Performance Leak, Timeline von Google Chrome Entwicklertools

PERFORMANCE

Im späteren Verlauf des Projektes wurde die Performance zunehmend zum Problem. Um die Performanceprobleme zu lösen mussten wir ständig den aktuellen Stand in verschiedenen Geräten/Browsern messen und vergleichen.

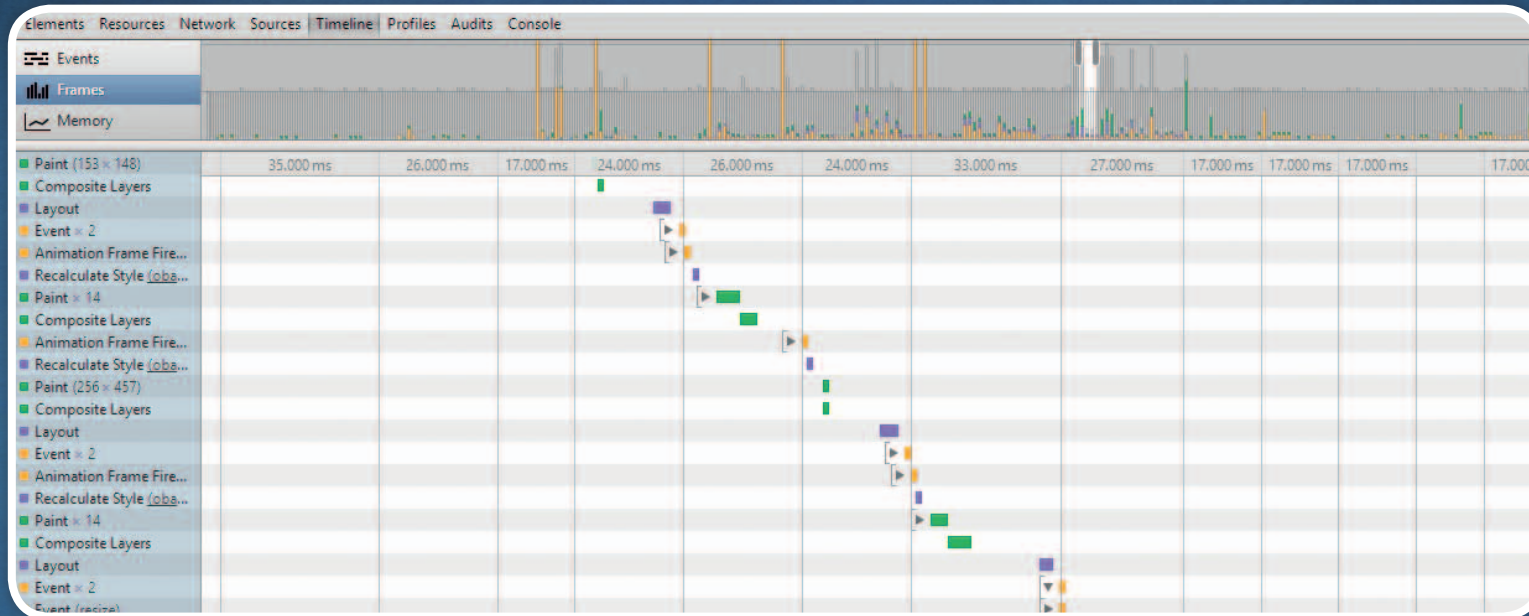
Da sich das besonders bei mobilen Geräten als schwierig erwies, haben wir einige Funktionen in den Client eingebaut, um die Geschwindigkeiten messen zu können. So

lassen sich die aktuellen FPS (Frames Per Second) nicht nur direkt am Bildschirm ausgeben, sondern auch durchschnittliche FPS berechnen, die auch remote zurück ins Serverlog gesendet und dort einfach ausgelesen werden können.

Google Chrome besitzt bei den Entwicklungstools einen schon sehr guten **Profiler**, der allerdings auch nicht bis in alle Details geht. Um Engstellen zu identifizieren haben wir

Teile des Spieles deaktiviert bzw. isoliert um ein Gefühl dafür zu bekommen, wie rechenintensiv einzelne Programmteile sind. Falls größere Performance-Einbußen aufgetreten sind, haben wir auch einzelne Git-Revisionen miteinander verglichen.

Die **Performance Leaks** bekamen wir gut in den Griff – auch ein sehr lange laufendes Spiel verliert über die Zeit jetzt keine nennenswerten Frameraten mehr. Dazu haben



PERFORMANCE

wir z.B. Destruktoren für alle relevanten Klassen eingeführt und die setTimeout und setInterval Funktion von JavaScript so erweitert, dass sie unkompliziert deaktiviert werden können.

Wir mussten auch die **Canvas-Auflösungen reduzieren** und die Spritesheets pixelgenau darauf anpassen. Auch wurden viele Einstellungen in und um die Spiele-Engine herum ausprobiert und die Performance verglichen.

Wir konnten also einige Optimierungen durchführen die auch deutliche Verbesserungen mit sich gebracht haben. Allerdings erwies sich die Game Engine an sich auch als Engpass und hier könnte man höchstens die Game Engine wechseln, was vom Aufwand für uns allerdings nicht mehr realisierbar war. **Auf Desktop-Browsern wurde insgesamt eine recht gute Performance erzielt, auf einem aktuellen iPhone und iPad ebenfalls.**

Android Geräte zeigten deutliche Schwächen wegen der schlechten DOM-Rendering-Performance des internen Browsers. Unter Android waren für uns selten mehr als 30 FPS erreichbar – auf dem iPhone 5 kamen wir auf 60 FPS.

Auf neueren Android Geräten läuft das Spiel allerdings mittlerweile zufriedenstellend.



ANDROID: <https://play.google.com/store/apps/details?id=com.obacht.game&hl=de>

APPLE: <https://itunes.apple.com/de/app/obacht/id666113251?mt=8>

FAZIT

Das Team hat es geschafft, in der vorgegebenen Zeit, eine für die Endkunden nutzbare Spieleapp zur Verfügung zu stellen. Das Spiel wurde in den App-Stores von Apple und Android zum kostenlosen Download bereit gestellt.

Die Aufgabenverteilung im Team mit den flexiblen Rollen und Schwerpunkten hat sehr gut funktioniert. Auch mit dem Zeitmanagement gab es keinerlei Probleme,

da wir sehr früh mit konkreten Arbeiten begonnen haben.

Es gab allerdings auch Schwierigkeiten. Besonders die Spiele-Engine LimeJS hat sich gegen Ende als Problem herausgestellt. Die Performance war bestenfalls mäßig und auch die Cross-Plattform Kompatibilität war nicht wie erhofft gegeben. Rückblickend hätten wir einige technische Entscheidungen anders gefällt.

Allerdings hat uns zu Beginn des Projekts diese Erfahrung gefehlt.

Positiv ist noch zu erwähnen, dass wir die Erfahrungen und das gelernte Wissen bezüglich der HTML5 Plattform gut auf andere Bereiche übertragen können.



AUSBLICK

Unser Spiel wurde von Anfang an modular konzipiert und programmiert. Es ist also relativ einfach erweiterbar.

Einige mögliche Erweiterungsmöglichkeiten wären z. B. die Umsetzung von **mehr Themes. Boni mit besonderen Eigenschaften** (zusätzliches Leben) könnten zum Gameplay beitragen. Ein sinnvoller Punkt wäre noch die **Einbindung von Musik und Sound**, die wir in der gegebenen Zeit leider nicht mehr

problemfrei implementieren konnten. Auch gibt es noch einiges Optimierungspotential: Wir hatten überlegt einen **High-Quality und einen Lowl-Quality Modus** einzuführen, so dass das Spiel auch auf langsameren Geräten flüssig läuft. Auch wäre es noch möglich/sinnvoll spezifischer auf die verschiedenen Gerätetypen einzugehen und einige Einstellungen entsprechend anzupassen. Eine sehr große Aufgabe, aber vermutlich

sinnvoll wäre das **Wechseln der Spiele-Engine**. Mit unserem Spielekonzept sind wir bei LimeJS schon an und über die Grenzen gekommen. Die größten Performancesprünge könnten wir nur erreichen indem wir die Engine wechseln. Aktuell sieht der HTML5 Wrapper Cocoon.js sehr vielversprechend aus, man müsste eine Engine finden die mit diesem Wrapper kompatibel ist, aber auch unsere sonstigen Ansprüche erfüllt.

VIELEN DANK!

SCHLUSSWORT

Zuerst wollen wir uns bei den betreuenden Professoren Dr. Wolfgang Kowarschick und Michael Stoll bedanken. Beiden standen uns nicht nur mit wichtigem Rat und Tat zur Seite sondern ließen uns auch bei der Ideenfindung, der gestalteri-

schen Konzeption sowie der technischen Umsetzung sehr viele Freiräume. Dem Team waren so nur wenige kreative Grenzen gesetzt. Die Rücksprachen zwischen Betreuer und Projektteam haben uns immer auf einen sehr guten Weg gebracht.

Der Hochschule Augsburg wollen wir für die bereitgestellten Räumlichkeiten sowie dem Budget danken.

Bei **S2 Intermedia** bedanken wir uns für das Sponsoring.