Octocube

Prototyp eines Web-Basierten Simulations-Spiel

Bachelorarbeit

Abteilung Informatik
Hochschule für Technik Rapperswil

Herbstsemester 2014

Autoren:	**Nicola Jordan**
	Linda Urech
Betreuer:	**Prof. Dr. Markus Stolze**
	Prof. Dr. Andreas Rinkel
Experte:	**Remo Brunschwiler (Namics)**
Projektpartner:	**Studiengang Wirtschaftsingenieurwesen**

HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL
FHO Fachhochschule Ostschweiz

Kapitel 1

Abstract

Ausgangslage

Eine wichtige Fähigkeit von ausgebildeten Wirtschaftsingenieuren ist die Optimierung von Produktionsprozessen. Die Auswirkung von Optimierungen auf die Effizienz und Wirtschaftlichkeit sind aber häufig nur mit Fachwissen nachvollziehbar und bleiben daher vielfach zu abstrakt. Dies ist eine Herausforderung in der Ausbildung junger Wirtschaftsingenieure. Durch die weite Verbreitung von Smartphones unter Studierenden und aufgrund der Verfügbarkeit von Beamern in Unterrichtsräumen ergeben sich aber neue Möglichkeiten, den Effekt von unterschiedlich organisierten Produktionsprozessen für Studierende im Rahmen eines verteilten Real-Time-Spieles erfahrbar zu machen. Mit ihren Smartphones können sie sich an einem Spiel anmelden und durch das Spielen eines Mini-Games verschiedene Aufgaben in einem simulierten Produktionsprozess wahrnehmen. Dadurch werden die Auswirkungen von unterschiedlichen Arbeitsorganisationen nachvollziehbar. Im Vorfeld der Bachelorarbeit zeigte sich, dass bereits einige Produktions-Simulationssysteme existieren. Diese weisen jedoch einen hohen Detaillierungsgrad auf und können nicht auf Tablets und Smartphones genutzt werden. Darum wurde im Rahmen dieser Bachelorarbeit eine Grundlage für ein System erarbeitet, welches für die Nutzung auf Smartphones und Tablets im Unterricht geeignet ist.

Vorgehen/Technologie

Die Anforderungen wurden in Form von User Stories erfasst und mit dem Kunden validiert. Parallel dazu wurde das Design der Mini-Games und das Management und

die Visualisierung des Spielstandes mittels Papier-Prototyping erhoben. Mit diesem Input wurde ein Architektur-Prototyp mit Client und Server Komponenten erstellt. Mit Load-Tests wurde sicher gestellt, dass die gewählte Architektur in der Lage ist, die Realtime-Anforderungen des verteilten Multi-User-Spiels zu erfüllen. Anschliessend wurden drei zusammenhängende webbasierte Mini-Games mit Hilfe des JavaScript Game-Frameworks Phaser umgesetzt. Für die Kommunikation der Mini-Spiele und für das Management und die Visualisierung des Spielstands wurde mit Python ein Server Programm mit REST API entwickelt.

Ergebnis

Ein erster praktischer Einsatz des entwickelten Systems erfolgt voraussichtlich am kommenden Bachelor-Informationstag der HSR im Rahmen der Präsentation des Studiengangs Wirtschaftsingenieurwesen. Durch die gewählte Architektur ist das System ausbaufähig. So können neue Mini-Spiele mit geringem Programmieraufwand integriert und bei Bedarf sogar der Spielfluss (Produktionsprozess) als Ganzes adaptiert werden. Daher bietet sich das Mini-Framework als flexible Grundlage an, auf welcher weitere Systementwicklungen möglich sind und eine breite Palette von unterrichtsunterstützenden Spielen in der WING-Ausbildung entwickelt werden können.

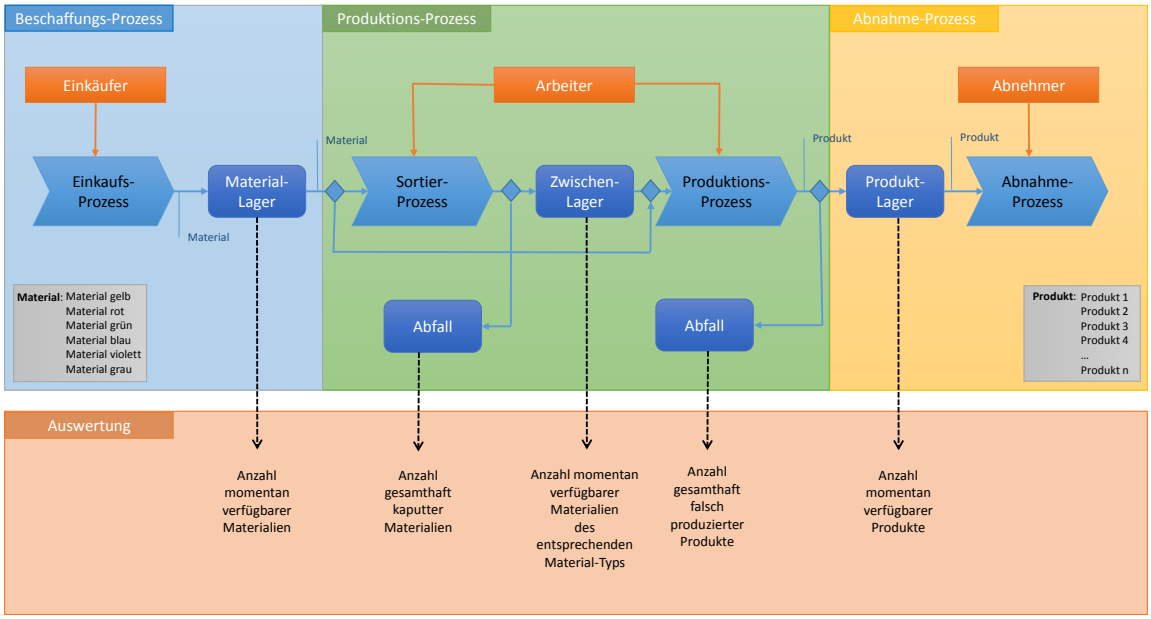
Kapitel 2

Management Summary

Ausgangslage

Eine wichtige Fähigkeit von ausgebildeten Wirtschaftsingenieuren ist die Optimierung von Produktionsprozessen. Die Auswirkung von Optimierungen auf die Effizienz und Wirtschaftlichkeit sind aber häufig nur mit Fachwissen nachvollziehbar und bleiben daher vielfach zu abstrakt. Dies ist eine Herausforderung in der Ausbildung junger Wirtschaftsingenieure. Durch die weite Verbreitung von Smartphones unter Studierenden und aufgrund der Verfügbarkeit von Beamern in Unterrichtsräumen ergeben sich aber neue Möglichkeiten, den Effekt von unterschiedlich organisierten Produktions-Prozessen für Studierende im Rahmen eines verteilten Real-Time-Spieles erfahrbar zu machen. Mit ihren Smartphones können die Studierenden sich an einem Spiel anmelden und durch das Spielen eines Mini-Games verschiedene Aufgaben in einem simulierten Produktionsprozess wahrnehmen. Dadurch werden die Auswirkungen von unterschiedlichen Arbeitsorganisationen nachvollziehbar. Im Vorfeld der Bachelorarbeit zeigte sich, dass bereits einige Produktions-Simulationssysteme existieren. Diese weisen jedoch einen hohen Detaillierungsgrad auf und können nicht auf Tablets und Smartphones genutzt werden. Darum wurde im Rahmen dieser Bachelorarbeit eine Grundlage für ein System erarbeitet, welches für die Nutzung auf Smartphones und Tablets im Unterricht geeignet ist.

Abbildung 2.1: Ablauf-Diagramm



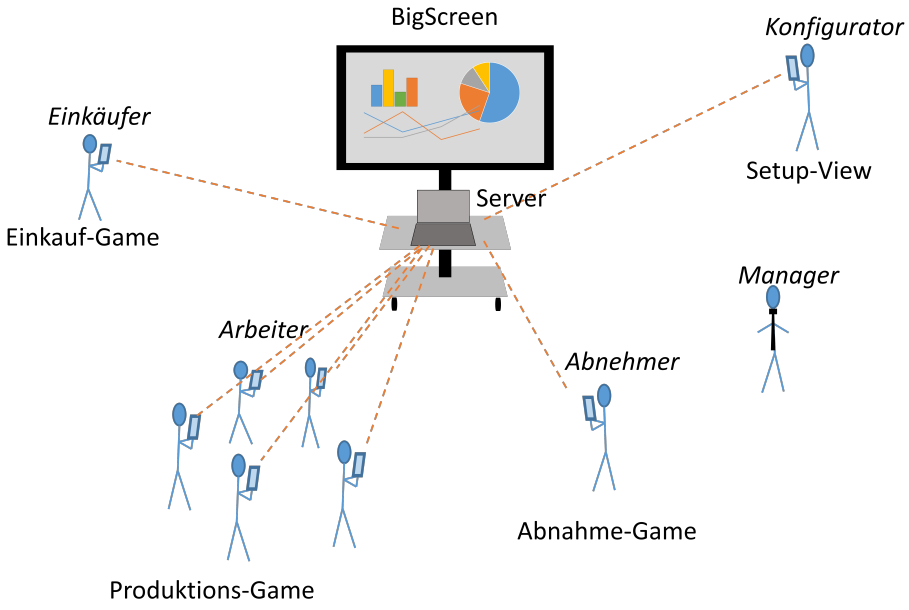
Vorgehen/Technologie

Die Anforderungen an das System wurden in diversen Sitzungen zusammen mit dem Kunden evaluiert und zusammengetragen. Anschliessend wurden diese in Form von User Stories erfasst und mit gemeinsam dem Kunden validiert. Parallel dazu wurde das Design der Mini-Games und das Management und die Visualisierung des Spielstandes mittels Papier-Prototyping erhoben. Die Paperprototypes wurden ebenfalls regelmässig mit dem Kunden besprochen und nach Wunsch angepasst. Mit diesen Inputs wurde ein Architektur-Prototyp mit Client und Server Komponenten erstellt. Mit zwei verschiedenen Load-Tests wurde sicher gestellt, dass die gewählte Architektur in der Lage ist, die Realtime-Anforderungen des verteilten Multi-User-Spiels zu erfüllen. Anschliessend wurden drei zusammenhängende webbasierte Mini-Games, welche drei Rollen einer Produtionsfirma darstellen, mit Hilfe des JavaScript Game-Frameworks Phaser umgesetzt. Für die Kommunikation der Mini-Spiele untereinander und für das Management und die Visualisierung des Spielstands wurde mit Python ein Server Programm mit REST API entwickelt.

Ergebnis

Ein erster praktischer Einsatz des entwickelten Systems erfolgt voraussichtlich am kommenden Bachelor-Informationstag der HSR im Rahmen der Präsentation des Studiengangs Wirtschaftsingenieurwesen. Durch die gewählte Architektur ist das System ausbaufähig. So können neue Mini-Spiele mit geringem Programmieraufwand integriert und bei Bedarf sogar der Spielfluss (Produktionsprozess) als Ganzes adaptiert werden. Daher bietet sich das Mini-Framework als flexible Grundlage an, auf welcher weitere Systementwicklungen möglich sind und eine breite Palette von unterrichtsunterstützenden Spielen in der WING-Ausbildung entwickelt werden können.

Abbildung 2.2: Big-Picture



Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Kapitel 3

Projektplan

3.1 Projektübersicht

In diesem Projekt soll ein (Mini-)Framework, eine entsprechende Schnittstelle dazu und eine Beispielsimplementation von drei zusammenhängenden Mini-Spielen implementiert werden.

3.1.1 Lieferumfang

Im Lieferumfang enthalten ist die Implementation des (Mini-)Framworks, die zugehörige Schnittstelle und drei Mini-Spiele. Die Dokumentation des Projektes wird mitgeliefert, welche das Deployment, die Code-Struktur den Framwork-Aufbau und dessen Nutzung erklärt.

3.2 Projektorganisation

Das Entwicklerteam besteht aus zwei einander gleichgestellten Mitglieder. Prof. Dr. Markus Stolze übernimmt die Rolle des Projektbetreuers und ist für die Formalitäten verantwortlich. Prof. Dr. Andreas Rinkel übernimmt ebenfalls die Rolle des Projektbetreuers und ist für die fachliche Betreuung in seinem Spezialgebiet Simulation zuständig. Der Studiengang Wirtschaftsingenierwesen, vertreten durch Prof. Dr. Daniel F. Keller, ist in diesem Projekt der Kunde. Ansprechspartner für das Entwicklerteam sind in erster Linie die Projektbetreuer.

3.3 Management

3.3.1 Projektkostenvoranschlag

Jedes Teammitglied hat 360 Stunden für eine Bachelorarbeit zur Verfügung. Für das gesamte Projekt stehen folglich insgesamt 720 Stunden zur Verfügung. In dieser Zeit enthalten ist Zeit für das Definieren der genauen Aufgabenstellung, Meetings mit dem Kunden und den Projektbetreuern, Einarbeitung, Programmierung, Dokumentation und andere administrativen Aufgaben.

3.3.2 Methodisches Vorgehen

Das methodische Vorgehen in diesem Projekt orientiert sich an den Grundlagen von agiler Software-Entwicklung.

3.3.3 Projektplan

Der Projektplan kann nach Absprache und mit Einverständnis aller Parteien abgeändert oder angepasst werden.

3.4 Infrastruktur

Die HSR stellt im Raum 1.206 insgesamt 2 Computer mit je einem Bildschirm für die Dauer der Bachelorarbeit zur Verfügung. Die Entwicklungsarbeiten werden grösstenteils auf den privaten Notebooks durchgeführt. Ausserdem wird dem Entwickler-Team für Testzwecke ein Samsung Tablet zur Verfügung gestellt. Der Code und die Dokumentation werden mittels “git” versioniert und über einen gemeinsamen Bit-Bucket-Account synchronisiert und sind so immer für alle Teammitglieder zugreifbar.

3.5 Qualitätsmassnahmen

3.5.1 Dokumentation

Um eine möglichst hohe Projekt-Qualität zu erreichen, werden Entscheidungen und implementierte Features fortlaufend dokumentiert. Sobald Änderungen vorgenommen werden, werden die entsprechenden Dokumente angepasst.

3.5.2 Meetings

Von jeder Sitzung wird ein eigenes Protokoll angefertigt, in welchem alle Beschlüsse sowie die weiteren Schritte festgehalten werden. Das Protokoll wird spätestens bis am Abend vor der nächsten Sitzung an alle Beteiligten, die ein Protokoll wünschen, verschickt. Zusätzlich werden alle Protokolle im Admin-Bereich der Dokumentation wie die Dokumentation selbst mit “git” versioniert und zentral abgelegt.

Meetings Entwicklerteam/Projekt-Betreuer

Meetings mit dem Entwicklerteam und den Projekt-Betreuern finden je nach Notwendigkeit grundsätzlich einmal pro Woche jeweils mittwochs statt. In der ersten Hälfte der Projektdauer werden die Sitzungen mit beiden Projekt-Betreuern und dem Entwicklerteam gleichzeitig geführt, nach gegenseitiger Absprache und Einverständnis aller Parteien finden in der zweiten Projekthälfte morgens zwischen 09:00 - 10:00 Uhr ein Meeting mit Dr. Prof. Markus Stolze und nachmittags zwischen 13:00 -14:00 Uhr ein Meeting mit Prof. Dr. Andreas Rinkel statt. Auf diese Weise können die Traktanden den entsprechenden Zuständigkeiten des jeweiligen Projekt-Betreuers angepasst werden und das Meeting kann entsprechend verdichteter durchgeführt werden.

Meetings Entwicklerteam/Kunde

Meetings mit dem Entwicklerteam und dem Kunden finden je nach Notwendigkeit grundsätzlich einmal pro Woche jeweils Dienstags von 14:15 bis 15:00 Uhr statt. In diesen Meetings werden dem Kunden die neuesten Fortschritte vorgestellt und besprochen sowie Änderungswünsche des Kunden aufgenommen.

3.5.3 Gegenseitige Reviews

Sämtliche verfassten Dokumente und Code werden vom jeweils anderen Teammitglied gesichtet und wenn nötig, nach Rückmeldung/Besprechung korrigiert. Auf diese Weise wird die Qualität von Dokumenten und Code erhöht.

3.5.4 Risikomanagement

Das Risikomanagement wird im Kapitel 9 erläutert.

3.5.5 Testing

Usability Tests

Vom Entwicklerteam ausgewählte Personen testen die Prototypen und geben Rückmeldung bezüglich ihrer Erfahrungen. Dieses Feedback fliesst ins Projekt ein und soll Schwächen und Probleme im Design aufzeigen.

Paperprototyping

Mittels Paperprototyping wird das Design der Applikation und deren Menü-Führung entworfen und anschliessend mit Personen getestet. Paperprototyps werden auch dem Kunden regelmässig gezeigt, so dass frühzeitig auf Änderungswünsche eingegangen werden kann.

3.6 Zeitplan

Der Zeitplan gliedert sich grundsätzlich in 4 verschiedene Phasen:

- Einrichten der Arbeitsumgebung, Einarbeitung in unbekannte Technologien, Erstellung der Risikoanalyse und -management
- Entwicklung Prototypen, grobes Design des GUI, Paperprototyping
- Implementation der Feature, Dokumentation, Testing
- Feinschliff der Dokumentation, Einarbeitung des Code-Review, Poster erstellen, Abgabe vorbereiten

Dabei entsprechen Phase 1 und 2 der Inception und Elaboration (nach Software Engeneering, kurz SE), Phase 3 der Construction (nach SE).

In den ersten 8 Wochen wurden die Phasen 1 und 2 durchgeführt, die restlichen 4 Wochen wurden für die Phase 3 und 4 verwendet. Die Transition nach SE konnte aufgrund der Zeitknappheit nicht ordentlich durchgeführt werden.

Um eine agile Arbeitsweise verfolgen zu können, werden in den ersten zwei Phasen UserStories erarbeitet, welche für das Entwicklungsteam in Tasks aufgeteilt werden. Diese Tasks werden auf Zettel geschrieben und anschliessend geschätzt. Die Tasks werden den Bereichen “Mini-Game”, “Backend”, “BigScreen” und “Admin” zugeordnet und an den entsprechenden Stellen an die “Planning-Wall” geklebt. Die Teammitglieder nehmen sich von dort selbstständig noch offene Tasks und erledigen diese. Jede

Woche wird die tatsächlich verbrauchte Zeit und die zuvor geschätzte verglichen. Entstehen grössere Abweichungen oder treffen Risiken ein, welche eine Verkleinerung des Projektumfangs verlangen, so werden die geringer priorisierten Tasks und die zugehörigen UserStories aussortiert. In der Abbildung 3.1 findet sich ein Beispielszustand der "Planning-Wall".

Abbildung 3.1: Planung der Tasks

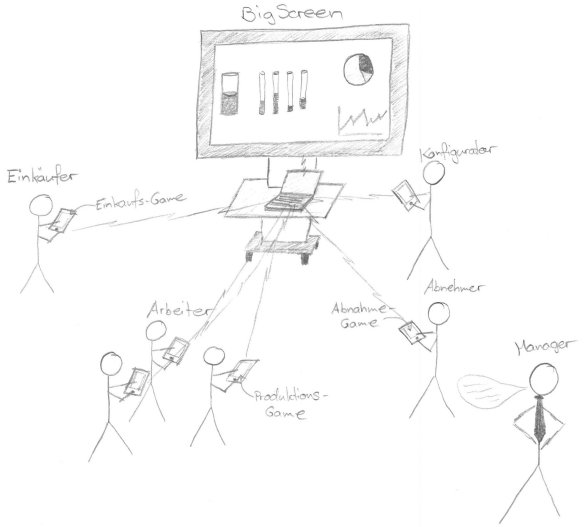
Kapitel 4

Vorstudie

4.1 Idee: Webbasiertes Multiplayer-Firmenführungsspiel

4.1.1 Big Picture

Abbildung 4.1: Big-Picture-Entwurf



Die Abbildung 4.1 stellt eine grobe Übersicht über das Spielumfeld dar. Vor dem eigentlichen Spielbeginn nimmt der Konfigurator auf der Konfigurations-View die für das Spiel notwendigen initialen Setup-Einstellungen vor. Nach dem Spielbeginn ist es Aufgabe des Einkäufers, Material für die Arbeiter einzukaufen. Der Abnehmer beginnt nach dem Spielbeginn mit dem Bestellen seiner gewünschten Produkte. Die Arbeiter versuchen nun mit dem zuvor vom Einkäufer eingekauften Material die vom Abnehmer bestellten Produkte zu produzieren. Auf dem BigScreen sehen alle Beteiligten

die aktuellen Spielwerte. Anhand dieser können sie durch Ändern der Spielstrategie bessere Spielwerte erhalten. Der Manager gibt den Arbeitern (und dem Einkäufer und Abnehmer) mündliche Anweisungen um eine bessere Spielstrategie umzusetzen.

4.2 Rollen innerhalb des Spiels

Konfigurator

Der Konfigurator eröffnet das Spiel und setzt vor dem eigentlichen Spielbeginn die initialen Parameter, welche die später im Spiel verfügbaren Materialtypen und Produkttypen bestimmen.

Manager

Der Manager spielt eine übergeordnete Rolle und kann nach Bedarf die Rollen des Konfigurators, des Einkäufers und/oder des Abnehmers übernehmen. Er überwacht das gesamte Spiel und gibt den Arbeitern mündliche Anweisungen, um den Spielverlauf zu steuern. Er ist ausserdem zuständig für das Starten und Beenden des Spiels.

Arbeiter

Der Arbeiter symbolisiert den "Arbeiter" einer Fabrik und erstellt durch das Spielen aus Materialien Produkte. Er spielt das Produktions-Game.

Einkäufer

Der Einkäufer bewirtschaftet durch Materialbestellungen das Hauptlager mit Materialien auf, welche den Arbeitern zur Verfügung stehen. Er spielt das Einkauf-Game.

Abnehmer

Der Abnehmer bestellt mittels Aufträgen Produkte, welche die Arbeiter produzieren. Er spielt das Bestell-Game.

4.3 Spiel-spezifische Begriffe

Materialtyp

Der Materialtyp wird durch verschiedenfarbige Rhomben dargestellt, welche zum Beispiel Holz, Metall oder Kunststoff symbolisieren.

Material

Material ist die Instanz eines bestimmten Materialtyps. Materialien können zu Produkten verarbeitet werden.

Produkttyp

Ein Produkttyp wird durch eine bestimmte Anordnung von benötigten Materialien dargestellt.

Produkt

Ein Produkt ist eine konkrete Instanz eines Produkttyps, die bestellt und produziert werden kann.

Materiallager

Die vom Einkäufer bestellten Materialien werden in das Materiallager (Anfangslager) abgelegt. Aus diesem Lager beziehen alle Arbeiter die zum Erstellen der bestellten Produkte benötigten Materialien.

Zwischenlager

Für jeden Materialtyp gibt es ein entsprechendes Zwischenlager, in welchem die im Moment gerade nicht benötigte Materialien eingelagert werden können. Alle Arbeiter haben Zugriff auf dieselben Zwischenlager und können so Materialien austauschen. Ist die Maximalkapazität eines Zwischenlagers erreicht, so werden alle anschliessend noch eingefüllten Materialien automatisch in den Abfall verschoben. Nimmt ein Arbeiter wieder Material aus dem Zwischenlager, so kann auch wieder neues Material eingelagert werden.

Produktlager

Im Produktlager werden alle Produkte der entsprechenden Bestellung zugewiesen. Hier werden auch alle eingegangenen Bestellungen gespeichert.

Abfall

Der Abfall stellt eine Mülltonne dar, in welche alle fehlerhaft erstellten Produkte, falsch eingelagerte Materialien, etc geworfen werden. Anhand der Anzahl Elemente im Abfall können Rückschlüsse auf die Fehlerquote gezogen werden.

4.4 Spielablauf

Konfiguration

Rolle Konfigurator

View Konfigurations-View

Vor dem eigentlichen Spielbeginn nimmt der Konfigurator in der Konfigurations-View die für das Spiel notwendigen initialen Setup-Einstellungen vor. In der Option "Anzahl Material-Typen" kann festgelegt werden, wie viele Material-Typen insgesamt im Spiel vorkommen sollen (entspricht der Anzahl Farben, die verwendet werden sollen). Wird beispielsweise 4 gewählt, so kommen im Spiel nur 4 verschiedene Material-Typen (Farben) vor. Um die Varianten-Vielfalt der produzierbaren und bestellbaren Produkte zu bestimmen, kann die gewünschte minimale und maximale Anzahl Material-Typen, welche in einem Produkt vorkommen sollen, gewählt werden. Mit den gewählten Werten 3 und 5 werden beispielsweise keine nur mit zwei verschiedenen Material-Typen produzierbare Produkte generiert, sondern nur solche, die zwischen drei und fünf verschiedene Material-Typen beinhalten. Als letzte Option kann die Anzahl der im Spiel verfügbaren verschiedenen Produkt-Typen bestimmt werden. Wird als Beispielswert 4 gewählt, so kann der Abnehmer nur zwischen vier verschiedenen bestellbaren Produkt-Typen auswählen.

Am Spiel anmelden

Nach den initialen Setup-Einstellungen können sich die Spieler durch wählen einer Rolle am Spiel anmelden. Das Anmelden kann wahlweise über das Scannen eines vom

Manager zur Verfügung gestellten QR-Codes oder durch das Eingeben der entsprechenden Adresse in der Adresszeile des Browsers erfolgen. Dabei gibt es für jede Rolle (und die entsprechend zu verwendende View) einen eigenen QR-Code oder eine eigene Adresse.

Material einkaufen

Rolle Einkäufer

View Einkaufs-View

Nachdem sich für jede Rolle mindestens eine Person angemeldet hat, beginnt der Einkäufer Materialien für die Arbeiter einzukaufen. Dies erfolgt durch das Klicken auf die gewünschte Bestellmenge des gewünschten Materialtyps. Die eingekauften Materialien werden in das Materiallager eingefüllt.

Produkt bestellen

Rolle Abnehmer

View Abnahme-View

Zeitgleich mit dem Einkäufer kann der Abnehmer mit Spielen beginnen. Der Abnehmer kann durch klicken eines Buttons in der Abnahme-View ein bestimmtes Produkt bestellen. Auf dem Button ist jeweils eine Vorschau des entsprechenden Produktes abgebildet. Werden mehrere Produkte nacheinander bestellt, so werden sie in einer Auftragswarteschlange zwischengespeichert und von den Arbeitern so bald als möglich abgearbeitet.

Materialien sortieren/ Produkt erstellen

Rolle Arbeiter

View Produktions-View

Nachdem Material eingekauft und erste Produkte bestellt sind, können die Arbeiter mit ihrer Aufgabe beginnen. Jeder Arbeiter erhält ein zufälliges Material aus dem Materiallager. Passt das Material nicht in das zugeteilte zu produzierendes Produkt, so kann der Arbeiter es in das entsprechende Zwischenlager verschieben. Auf dort zwischengespeicherte Materialien können alle Arbeiter zugreifen. Einmal verbautes

Material kann nicht mehr aus dem Produkt entfernt werden. Sobald ein Material für ein Produkt verbaut ist, erscheint - sofern es noch Materialien im Materiallager hat - ein neues Material im oberen Bereich des Spielfeldes. Wird darauf ein Material aus einem Zwischenlager geholt, so wird das vorher aus dem Materiallager erschienene Material wieder ins Materiallager zurückgelegt. Sobald ein Produkt fertig gestellt ist, wird es automatisch abgeschickt und überprüft. Der Arbeiter sieht anhand der Counter im unteren Bereich des Spielfeldes, wie viele Produkte er fehlerhaft abgeliefert hat und wie viele Materialien er falsch in die Zwischenlager einsortiert hat.

Spiel steuern

Rolle Manager

View Manager-View

Der Manager kann das Spiel jederzeit starten, pausieren oder beenden. Seine View ist durch ein Login geschützt.

4.4.1 Anforderungsanalyse - NFR's

Für komplexes System, welches dynamische Events generiert und verschiedene Prozesse anstösst, eignet sich eine Event-Driven-Architecture besonders gut. Daraus lassen sich, aufbauend auf Bruns und Dunkels "Event-Driven Architecture: Softwarearchitektur für ereignisgesteuerte Geschäftsprozesse"[1], folgende NFR's ableiten:

Agilität (hohe Veränderbarkeit) Es soll eine Anpassung an verschiedene Aufgabenstellungen möglich sein.

Aktualität Der Zustand und das Resultat der verschiedenen Arbeitsschritte und Prozesse muss möglichst aktuell gehalten werden und sichtbar sein.

Effizienz Es sollen bis zu 60 Spieler gleichzeitig an einem Spiel teilnehmen können. Dafür müssen die von den Spielern generierten Events in angemessener Zeit verarbeitet werden können.

Robustheit Events werden über einen Push-Mechanismus an einen Server geleitet. Somit sind nur selten Änderungen an der Schnittstelle zwischen Client und Server notwendig.

Wartbarkeit Das System muss [1] einfach erweiterbar sein und mit wenig Aufwand aktuell gehalten werden können.

Skalierbarkeit Um zu einem späteren Zeitpunkt auch mehr als 60 Spieler gleichzeitig unterstützen zu können, muss das System gut skalieren.

Flexibilität Der Unterbau des Systems muss auch für andere Aufgabenstellungen und deren Anforderungen[2] verwendet werden können.

Die oben aufgeführten Anforderungen bergen folgende Probleme:

Redundanz Da die Daten nicht zentral gespeichert sondern dupliziert werden, kann die Konsistenz der Daten im jetzigen Zustand nicht gewährleistet werden. Sollte dies trotzdem erforderlich sein, so erfordert die entsprechende Umsetzung zusätzlichen Aufwand.

Fehlersuche Dadurch, dass die verschiedenen Komponenten eine äusserst lose Kopplung aufweisen, wird die Fehlersuche erschwert, da nur schwer vorhergesagt werden kann, wo genau das Problem aufgetreten ist.

Unsicherheit Regelbasierte Anweisungen sind weniger vorhersagbar als fixer Code.

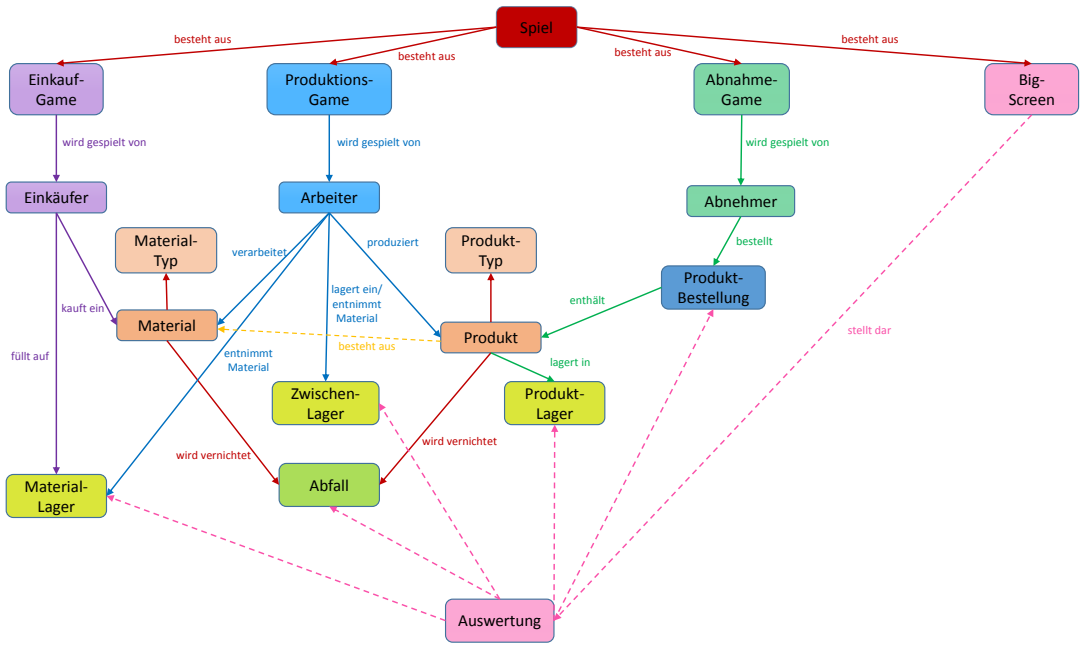
Kontrollfluss Aufgrund der vielen Optionen und Kombinationsmöglichkeiten von Regeln ist es viel schwieriger, einen Kontrollfluss zu beschreiben und zu verfolgen, als in einer synchronen Abfolge.

Da mit User Stories gearbeitet wird, wird auf eine funktionale Anforderungsanalyse bewusst verzichtet. User Stories sind für den Kunden einfacher zu erfassen, da ein ihm bekanntes Vokabular verwendet werden kann. Dadurch können die User Stories durch den Kunden einfacher validiert werden.

[1] Durch einen Programmierer
[2] Aus derselben Domäne.

4.4.2 Domain Model

Abbildung 4.2: Domainmodell



Die Abbildung 4.2 beschreibt das Domainmodell.

Ein Spiel besteht aus einem Big-Screen und den drei Mini-Spielen Einkauf-Game, Produktions-Game, Abnahme-Game.

Das Einkauf-Game wird vom Einkäufer gespielt und dient dem Befüllen des Materiallagers mit Materialien. Der Abnehmer bestellt im Abnahme-Game verschiedene Produkte, die die Arbeiter herstellen sollen. Im Produktions-Game sortieren und verarbeiten die Arbeiter die Materialien, die sie aus dem Materiallager erhalten und produzieren die vom Abnehmer bestellten Produkte.

Fertige Produkte werden in das Produktlager verschoben, von wo aus sie das Spiel - als abgenommen durch den Abnehmer - verlassen.

Defekte Materialien, durch falsches Einsortieren oder fehlerhaftes Produzieren entstanden, werden in den Abfall verschoben. Aus diesem berechnet sich der Error-Counter.

Der Big-Screen besteht aus der Anmelde-View und der Auswertungs-View. Die Auswertungs-View stellt die verschiedenen Aspekte des Spielverlaufs dar: Die Anzahl noch verfügbarer Materialien im Materiallager, die Anzahl der defekten Materialien (Fehler-Counter), die Anzahl der gesamthaft erstellten Produkte (Produkt-Counter) oder die Anzahl der bestellten und noch zu produzierenden Produkte.

4.4.3 Geräte

Clientseitig kommen Tablets zum Einsatz, serverseitig findet sich ein leistungsfähiger Laptop. Dieser ist mit einem Router verbunden, mit welchem sich die Clients verbinden können. Alternativ zum Router kommt das HSR-Netz zum Einsatz.

4.5 User Stories

Umgesetzte Userstories

Rolle Konfigurator

User Story: Für geschützten Bereich anmelden Als Konfigurator möchte ich mich mit einem Benutzernamen und Passwort anmelden können, um im geschützten Bereich spielspezifische Einstellungen vornehmen zu können.

User Story: Anzahl Materialtypen und Anzahl Produkttypen festlegen Als Konfigurator möchte ich die Anzahl der in diesem Spiel verfügbaren Materialtypen und Produkttypen vor dem eigentlichen Spielbeginn festlegen können.

User Story: Spiel eröffnen Als Konfigurator möchte ich ein Spiel eröffnen können, so dass sich Mitspieler daran anmelden können.

Rolle Manager

User Story: Spiel starten Als Manager möchte ich ein eröffnetes Spiel starten können.

User Story: Spiel pausieren Als Manager möchte ich das Spiel unterbrechen können mit der Möglichkeit weiter zu spielen oder das Spiel zu beenden.

User Story: Spiel beenden Als Manager möchte ich ein gestartetes Spiel beenden können.

Rolle Einkäufer

User Story: Material einkaufen Als Einkäufer möchte ich Materialien verschiedener Materialtypen einkaufen und dem Materiallager hinzufügen.

Rolle Abnehmer

User Story: Produkt abnehmen Als Abnehmer möchte ich einen Produkttyp bestimmen, von welchem ich ein spezifisches Produkt wünsche.

Rolle Arbeiter

User Story: An Spiel anmelden Als Arbeiter möchte ich mich an einem eröffneten Spiel anmelden.

User Story: Material in Zwischenlager verschieben Als Arbeiter möchte ich ein im Moment unpassendes Material in das entsprechende Zwischenlager verschieben.

User Story: Material aus Zwischenlager holen Als Arbeiter möchte ich ein Material eines bestimmten Materialtyps aus dem entsprechenden Zwischenlager holen.

User Story: Material in Produkt einfügen Als Arbeiter möchte ich ein Material in das zu fertigende Produkt einfügen.

User Story: Produkt fertigstellen Als Arbeiter möchte ich, sobald ich das aktuelle Produkt fertiggestellt habe, einen neuen Auftrag für ein Produkt erhalten.

User Story: Fehler am erstellten Produkt erkennen Als Arbeiter möchte ich Rückmeldung erhalten, ob das erstellte Produkt korrekt oder fehlerhaft gefertig worden ist.

User Story: Verschieben in falsches Zwischenlager erkennen Als Arbeiter möchte ich Rückmeldung erhalten, ob ich das Material in das korrekte oder in ein falsches Zwischenlager verschoben habe.

Optionale Userstories

Rolle Manager

User Story: Lager versiegeln Als Manager möchte ich das Materiallager versiegeln können, so dass die Arbeiter kein Material mehr beziehen können, sondern nur noch mit bereits im Spiel vorhandenen Materialien arbeiten können.

User Story: Lager wieder öffnen Als Manager möchte ich ein zuvor versiegeltes Lager wieder öffnen können.

User Story: Arbeiter blockieren Als Manager möchte ich einen beliebigen Arbeiter blockieren können, so dass er nicht mehr weiter arbeiten kann.

User Story: Blockierten Arbeiter aktivieren Als Manager möchte ich einen beliebigen blockierten Arbeiter wieder reaktivieren können, so dass der Arbeiter wieder weiterarbeiten kann.

Rolle Arbeiter

User Story: Vom Spiel abmelden Als Arbeiter möchte ich mich von einem gestarteten Spiel abmelden können.

4.6 Paperprototyping

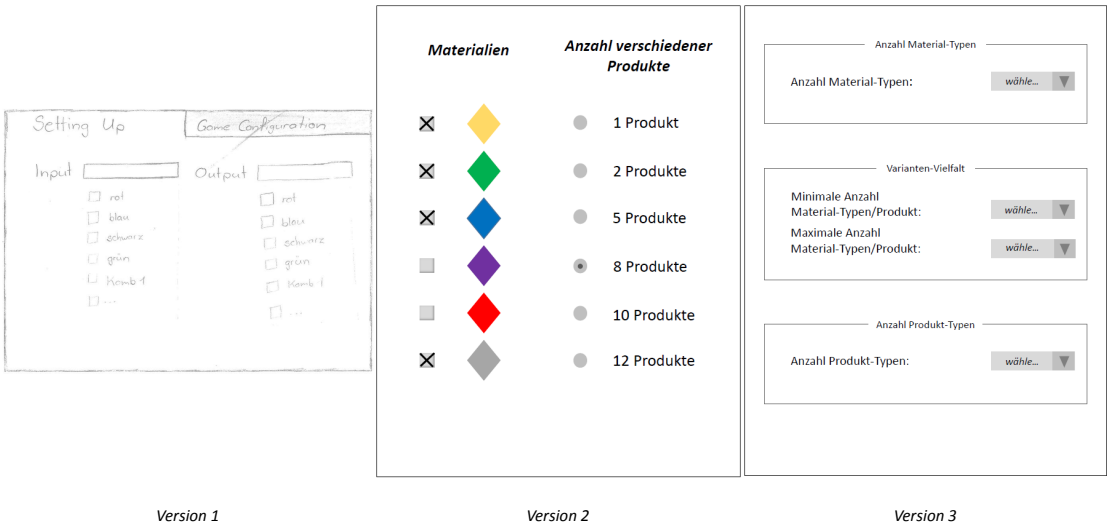
4.6.1 Settings

In der Settings-View kann der Konfigurator vor dem eigentlichen Spielbeginn die initialen Einstellungen vornehmen. Darunter fällt auch die Entscheidung, welche Materialien im Spiel vorkommen sollen und wie viele verschiedene Produkte bestellt und produziert werden können.

In der Abbildung 4.3 sind die verschiedenen Versionen des Paperprototyps dargestellt. In einer ersten Version können in einer Dropdown-Liste die gewünschten Materialien und Produkte markiert werden. Die Dropdown-Liste erweist sich jedoch als sehr unpraktisch um mittels Touch-Input bedient zu werden. Ausserdem kann der Benutzer die Materialien nicht sehen, sondern nur deren Namen lesen und muss anhand dieser wissen, um welches Material oder Produkt es sich handelt.

In einer zweiten Version wird mit mehr visuellen Elementen gearbeitet. Der Benutzer kann sich jetzt unter den auszuwählenden Materialien etwas vorstellen. Ausserdem wird die Möglichkeit, ein spezifisches Produkt auszuwählen, entfernt, da die Testbenutzer viel zu sehr über die möglichen Auswirkungen im Spiel nachdachten und ihnen deshalb diese Entscheidung jeweils sehr schwer fiel. Als viel einfacher wird es empfunden, nur die Anzahl der verschiedenen Produkte zu bestimmen. Die Produkte werden anschliessend vom Spiel generiert.

Abbildung 4.3: Paperprototyp-Settings



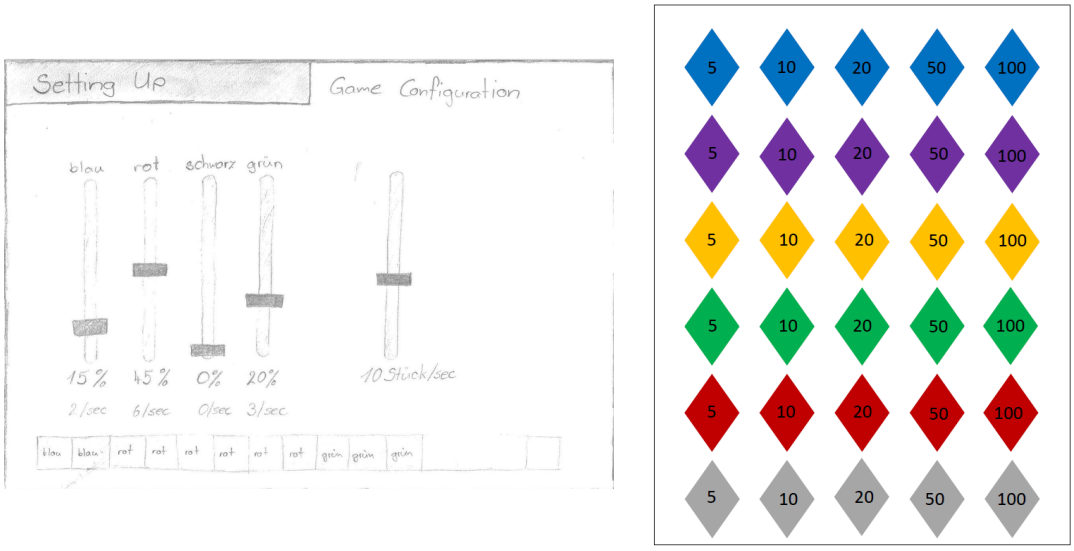
In der finalen Version (Version 3) werden wieder Dropdown-Listen verwendet, da beschlossen wurde, dass der Nutzer dieser View (in der Rolle des Konfigurators), eingewiesen sein wird. Neu können mehr Parameter eingestellt werden. In der ersten Box kann die Anzahl im Spiel vorkommender Materialtypen bestimmt werden. In der zweiten Box kann die Varianten-Vielfalt der Produkte eingestellt werden. In der letzten Box kann die Anzahl der Produkttypen bestimmt werden.

Die Idee eines kleinen Baukastens, in welchem der Konfigurator die Produkte, welche im Spiel vorkommen sollten, selber zusammenbauen kann, wurde aus Zeitgründen verworfen.

4.6.2 Einkaufs-Game

Das Einkaufs-Game (Abbildung 4.4) war zuerst nicht als eigenständiges Game, sondern als weitere View für den Manager geplant (Version 1). Der Manager kann in dieser View nach dem Spielstart Einfluss auf das Spielgeschehen nehmen durch die gezielte prozentuale Verteilung der verschiedenen Materialien im Materiallager. Die View stellt für jedes Material (in jenem Beispiel noch vier verschiedene, später dann sechs) einen Slider zur Verfügung, um die prozentualen Anteile der Materialien im Materiallager festzulegen. Mittels eines zusätzlichen Sliders kann die Produktionsgeschwindigkeit (in Stück pro Sekunde) der eingestellten Materialmischung bestimmt werden. Der untere

Abbildung 4.4: Paperprototyp Einkaufs-Game



Version 1 Version 2

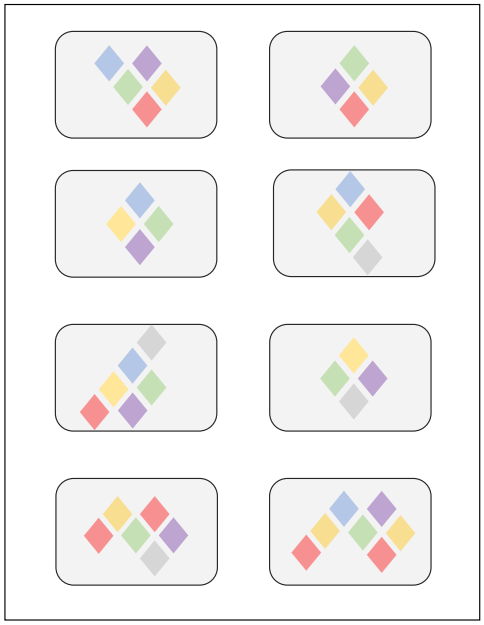
Bereich stellt die eingestellte prozentuale Verteilung der Material-Typen in Bezug auf das gesamte Materiallager dar. Diese View sollte anfangs nur vom Manager bedient werden, später wünschte der Kunde, diese Aufgabe einem Spieler überlassen zu können. Da die View sehr technisch gehalten war, wurde gemeinsam entschieden, ein eigenes Game daraus zu machen - das Einkaufs-Game.

In einem nächsten Schritt wird aus der Game-Konfigurations-View das eigenständige Mini-Game Einkaufs-Game (Version 2) entworfen. In diesem Game kann nun ein Spieler die neu kreierte Rolle des Einkäufers übernehmen und das Materiallager für die Arbeiter auffüllen. Da die View von einem nicht-instruierten Benutzer bedient werden können soll, wurde auf die visuelle Komponente besonders Wert gelegt und für jeden Einkaufs-Auftrag ein eigener Button kreiert, auf welchem die Menge steht, die eingekauft werden soll.

4.6.3 Abnahme-Game

Das Abnahme-Game (Abbildung 4.5) soll durch die Rolle des Abnehmers einen Gegenpol zum Einkäufer schaffen. Der Abnehmer bestellt verschiedene Produkte, welche anschliessend von den Arbeitern produziert werden sollen. In den Settings wurde vor dem Spielbeginn durch den Manager die Anzahl verschiedener Produkte festgelegt und das Spiel generierte diese anschliessend. Der Abnehmer sieht eine entsprechende

Abbildung 4.5: Abnahme-Game-View

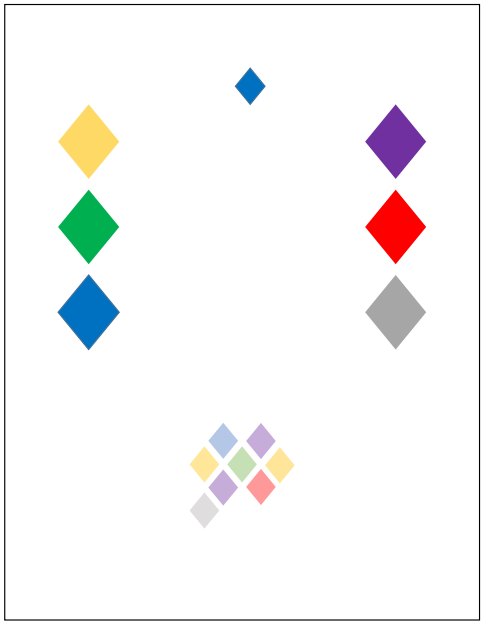


Anzahl Buttons, auf denen die bestellbaren Produkte abgebildet sind. Damit der Abnehmer im Vergleich zum Einkäufer nicht massiv weniger zu tun hat, können Produkte nur einzeln nacheinander und nicht in Massenbestellungen geordert werden.

4.6.4 Produktions-Game

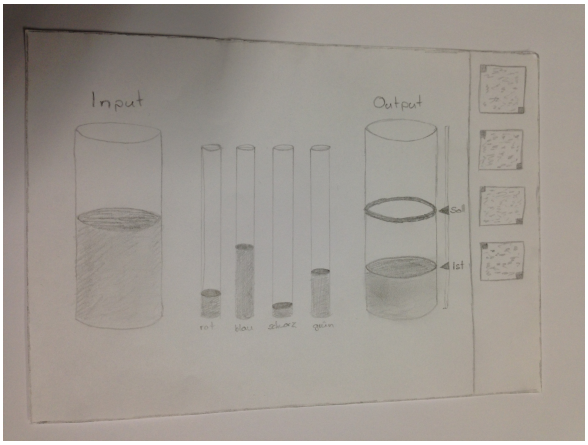
Diese View stellt das Hauptspiel dar und wird von den Arbeitern gespielt. Im oberen Bereich kann der Sortierprozess ausgeführt werden, indem das Material auf ein entsprechendes Zwischenlager gezogen wird. Im unteren Bereich kann der Produktionsprozess ausgeführt werden, indem ein Material (zum Beispiel aus einem Zwischenlager) in das im unteren Teil abgebildete Produkt-Muster eingesetzt wird. Sobald das Produkt komplett zusammengesetzt ist, wird es durch ein nächstes herzustellendes Produkt aus der Auftragswarteschlange ersetzt.

Abbildung 4.6: Produktions-Game-View



4.6.5 BigScreen

Abbildung 4.7: BigScreen-Auswertung



Auf dem BigScreen (Abbildung 4.7) werden die verschiedenen Auswertungen des Spielverlaufs dargestellt. Die Spieler sehen dort, wie viele und welche Materialien sich noch im Materiallager befinden, wie viele Produkte korrekt und wie viele fehlerhaft produziert wurden oder wie viele Materialien falsch sortiert wurden.

In einer ersten Version werden nur das Materiallager und das Produktlager je mit

einer grossen Röhre und die Zwischenlager mit einer kleinen Röhre dargestellt. An den Röhren kann der ungefähre Bestand abgelesen werden. Auf der rechten Seite befinden sich die QR-Codes für die Anmeldung am Spiel.

4.7 Auswahl JavaScript Game-Framework

Um einen Überblick über die verschiedenen JavaScript Game-Engines zu bekommen, wurde die Website "Breakouts"[3] zu Hilfe gezogen. Der Betreiber der Seite hat es sich zur Aufgabe gemacht, Entwicklern die Entscheidung für eine JavaScript Game-Engine zu erleichtern. Dafür hat er dasselbe "Breakout" Spiel in insgesamt 12 verschiedenen Frameworks implementiert. Die Varianten können gespielt werden um sich ein Bild bezüglich der Bedienbarkeit zu machen und der Source-Code kann angeschaut werden um sich über die Komplexität des Frameworks zu informieren. Alle vorgestellten Framworks beinhalten folgende für dieses Projekt relevante Features:

- Collision Detection
- Sprite Animation
- Menu & Scene Transitions
- Player Input
- Text Rendering
- Mobile Support

Kandidaten

Die verschiedenen Varianten wurden eingehend ausprobiert und getestet. Aufgrund erster Ergebnisse sind die folgenden 4 Kandidaten in die nähere Auswahl gerückt:

- Crafty
- Impact
- Phaser
- Lime

[3] http://www.jsbreakouts.org, 17.12.2014

Kriterien

Anhand der folgenden 5 Kriterien wurden die Frameworks miteinander verglichen:

Erlernbarkeit Wie viele Stunden müssen für die Einarbeitung investiert werden? Sind Getting-Started-Tutorials vorhanden? Wie einfach ist der Code der Beispiel-Implementationen zu lesen und nachzuvollziehen?

Zukunftssicherheit Wird das Framework noch aktiv weiter entwickelt? Wann wurde das letzte Update herausgegeben?

Functionality Stellt das Framework alle benötigte Funktionalität zur Verfügung? Müssen gewisse Grund-Features (wie zum Beispiel Drag&Drop) selber implementiert werden?

Community Wie gross und aktiv ist die Community hinter dem Framework? Wie lange dauert es, bis in einem Forum auf eine Frage geantwortet wird? Wie gut sind die Antworten?

Usability Gibt es Beispielsimplementationen der wichtigsten Grund-Features? (Zum Beispiel Laden eines Bildes, Bewegen eines Objekts durch User-Input, Drag&Drop, etc.) Wie gut ist die Dokumentation?

Dabei wurden die Kriterien wie in Abbildung 4.8 gewichtet:

Abbildung 4.8: Gewichtung der Kriterien

Kriterien	1	2	3	4	5	Gewicht	Faktor
1 Erlernbarkeit		1	1	2	2	6	0.3
2 Zukunftssicherheit	1		1	2	2	6	0.3
3 Functionality	1	1		2	2	6	0.3
4 Community	0	0	0		1	1	0.05
5 Usability	0	0	0	1		1	0.05

Die Gewichtung der einzelnen Kriterien bedeuten:

0 Faktor aus der entsprechenden Zeile ist *weniger wichtig* wie der Faktor aus der entsprechenden Spalte

1 Faktor aus der entsprechenden Zeile ist *gleich wichtig* wie der Faktor aus der entsprechenden Spalte

2 Faktor aus der entsprechenden Zeile ist *wichtiger* wie der Faktor aus der entsprechenden Spalte

Bewertungserklärung

In der Abbildung 4.9 findet sich die Bewertungserklärung.

Abbildung 4.9: Bewertungserklärung

Skala Kriterium	0-2 "schlecht"	3-5 "mittel"	6-8 "gut"
1 Erlernbarkeit	Einarbeitungsaufwand beträgt mehr als 25 Stunden	Einarbeitungsaufwand beträgt zwischen 15 und 25 Stunden	Einarbeitungsaufwand beträgt weniger als 15 Stunden
2 Zukunftssicherheit	Framework wird nicht mehr aktiv weiterentwickelt	für das Framework gab es im letzen Jahr kein Update mehr	für das Framework wurde in diesem Jahr bereits ein Update herausgegeben
3 Functionality	das Framework unterstützt die geforderte Funktionalität nicht	das Framework unterstützt die geforderte Funktionalität nur teilweise	das Framework unterstützt die geforderte Funktionalität vollumfänglich
4 Community	es gibt keine oder nur eine ungepflegte Community hinter dem Framework	es gibt nur eine kleine Community hinter dem Framework	es gibt eine grosse und aktive Community hinter dem Framework
5 Usability	es gibt keine offiziellen Beispiele zum Implementation häufig verwendeter Features, eine Dokumentation existiert nicht	es gibt nur von der Community einige Beispiele zur Implementation häufig verwendeter Features, die Dokumentation ist knapp gehalten oder mangelhaft	es gibt viele offizielle Beispiele zur Implementation häufit verwendeter Features, die Dokumentation ist umfangreich und aktuell

Auswertung

Phaser hat die höchste relative Bewertung (6.75) erhalten und ist somit der Kandidat der Wahl. Die genaue Punkteverteilung ist in der Abbildung 4.10 ersichtlich.

Abbildung 4.10: Nutzwerte

		Crafty		**Impact**		**Phaser**		**Lime**	
Kriterium	Gewichtung	Bewertung	Gesamt	Bewertung	Gesamt	Bewertung	Gesamt	Bewertung	Gesamt
Erlernbarkeit	30%	5	1.5	6	1.8	7	2.1	5	1.5
Zukunftssicherheit	30%	8	2.4	8	2.4	8	2.4	7	2.1
Functionality	30%	6	1.8	6	1.8	5	1.5	5	1.5
Community	5%	7	0.35	6	0.3	8	0.4	6	0.3
Usability	5%	7	0.35	7	0.35	7	0.35	7	0.35
Total		**6.4**		**6.65**		**6.75**		**5.75**	

Kapitel 5

Architektur

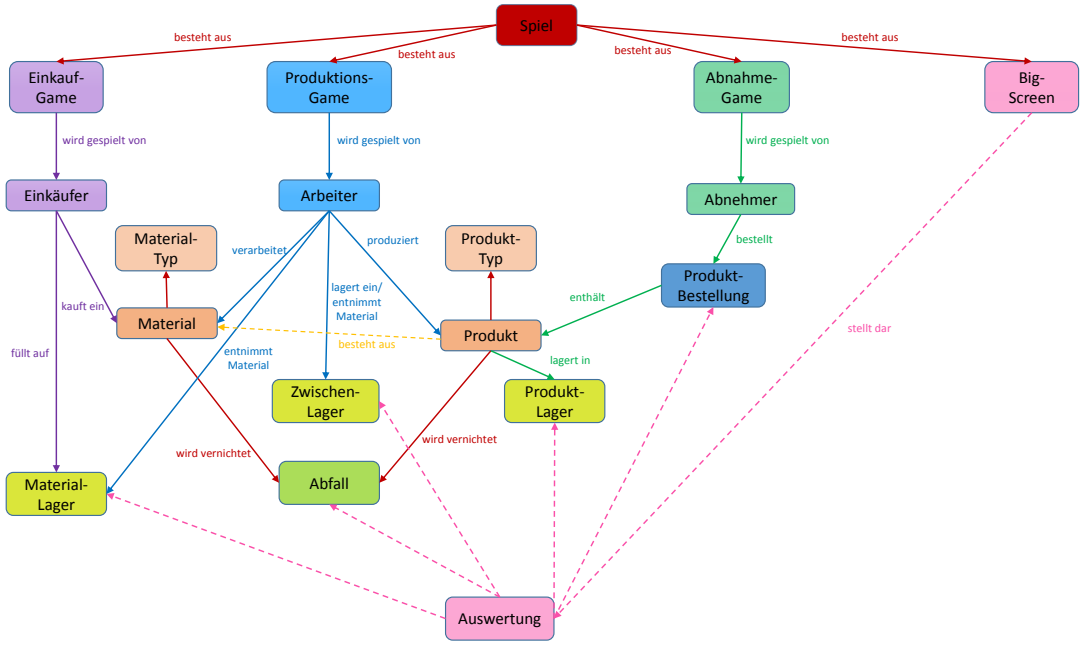
5.1 Multiviewpointanalyse

Im folgenden Kapitel wird der Aufbau und Ablauf des Spielkonzepts von verschiedenen Sichtpunkten aus beschrieben.

5.1.1 Domainmodell

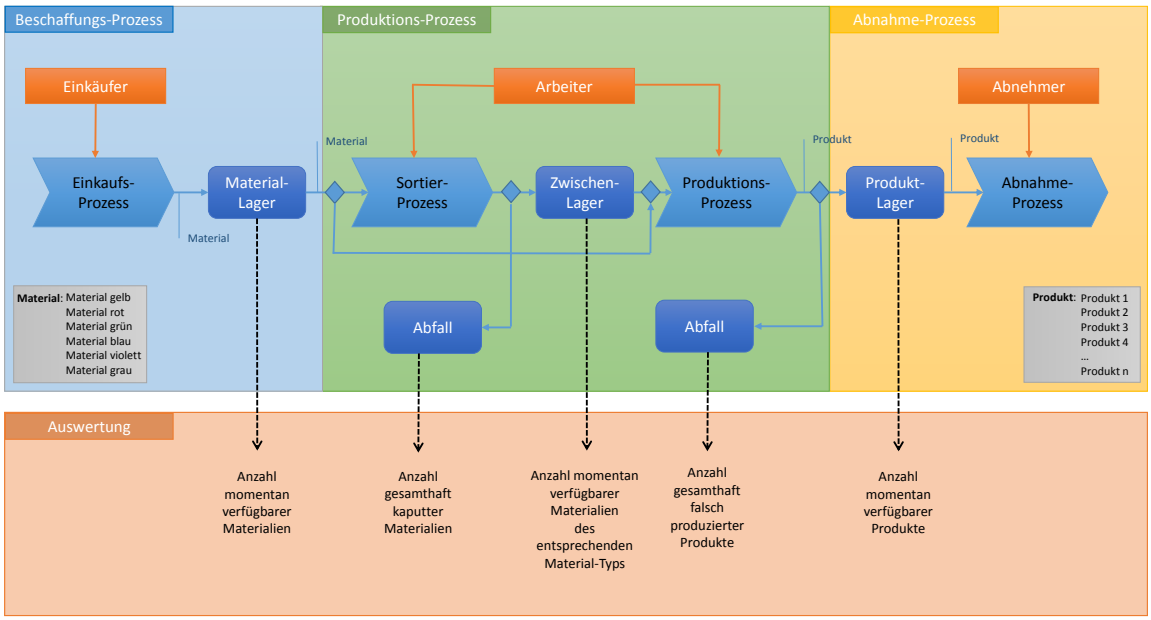
Das Domainmodell in Abbildung 5.1 stellt die wichtigsten Komponenten der Problemlösung dar. Eine genaue Beschreibung dieser und eine Erklärung derer Zusammenhangs findet sich in der Vorstudie.

Abbildung 5.1: Domainmodell



5.1.2 Ablauf

Abbildung 5.2: Ablaufdiagramm



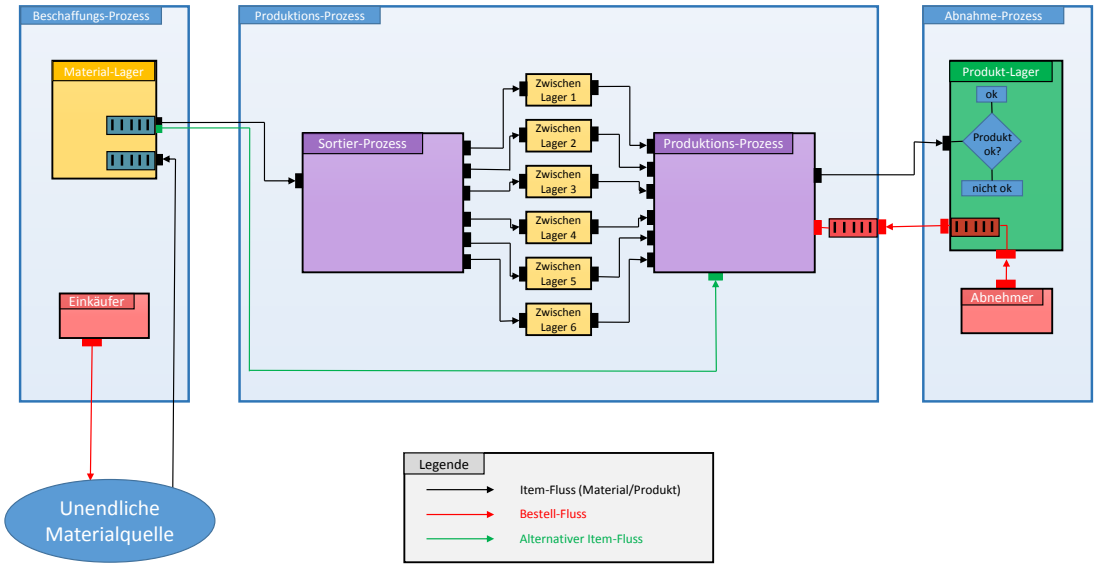
Im Diagramm 5.2 sind die drei Prozesse Beschaffung, Produktion und Abnahme zu sehen, welche in den drei Spielen Einkaufs-Game, Produktions-Game und Abnahme-Game stattfinden. Orange gekennzeichnet sind “human resources” (Einkäufer, Arbeiter, Abnehmer), also die Rollen der Spieler, welche die entsprechenden Games bedienen. Die einzelnen Unterprozesse (Beschaffungs-Prozess, Sortier-Prozess, Produktions-Prozess und Abnahme-Prozess) sind in hellblauen Pfeilformen dargestellt, während die Lager (Materiallager, Zwischenlager, Produktlager und Abfall) durch dunkelblaue Rechtecke symbolisiert sind.

Das Diagramm zeigt den Flow des Materials und später des Produkts durch die verschiedenen Prozesse. Sobald der Einkäufer ein Material einkauft, wird es in das Materiallager verschoben. Dort wird das Material vom System wieder herausgenommen und dem Arbeiter zur Verfügung gestellt. Der Arbeiter entscheidet sich, ob er das Material für die spätere Verwendung einsortieren oder direkt für den Produktionsprozess verwenden möchte. Wird das Material korrekt einsortiert, gelangt es in das entsprechende Zwischenlager; wurde es ins falsche Zwischenlager einsortiert, wird das Material in den Abfall verschoben. Ein Arbeiter kann entscheiden, ob er den Sortier- oder den Produktions-Prozess übernehmen möchte. Nach dem Produktions-Prozess wird das fertige Produkt auf dessen Korrektheit überprüft. Wurde das Produkt fehlerhaft zusammengesetzt, so wird es in den Abfall verschoben; wurde das Produkt korrekt zusammengesetzt, so kommt es in das Produktlager.

Im unteren Bereich des Diagramms ist die Auswertung dargestellt. Diese setzt sich aus verschiedenen Werten zusammen, welche von den entsprechenden Stellen stammen.

5.1.3 Prozesse

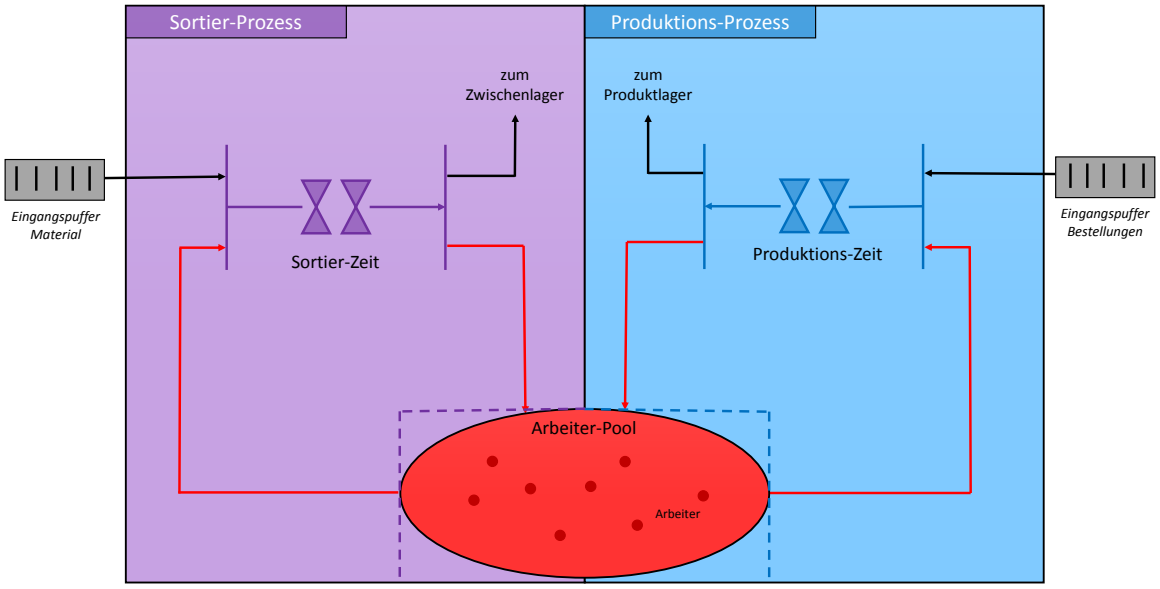
Abbildung 5.3: Flow-Diagramm



Im Diagramm 5.3 sind die drei Prozesse aus Simulations-Sicht dargestellt. Der im Folgenden verwendete Begriff Item entstammt der “Simulations-Sprache” und entspricht in diesem Fall einem Material oder Produkt.

Die blauen Bereiche symbolisieren die drei Prozesse, die in den drei verschiedenen Games durchgeführt werden. Der Einkäufer schickt eine Bestellung für ein Item an eine unendliche Materialquelle. Diese speist ein Item ins Spiel ein, indem das Item in die Queue des Materiallagers eingefügt wird. Anschliessend gelangt das Item direkt zum Produktions-Prozess oder zum Sortier-Prozess. Der Weg vom Materiallager zum Produktions-Prozess ist grün als alternativer Item-Fluss gekennzeichnet, da er nur im unstrukturierten Zustand der Arbeiter vorkommt. Sobald der Manager bestimmt hat, welche Arbeiter sortieren und welche produzieren sollen, sollte kein Arbeiter mehr ein Material direkt aus dem Materiallager verarbeiten. Nach dem Sortier-Prozess werden die Items in das entsprechende Zwischenlager verschoben. Aus dem Zwischenlager gelangen die Items in den Produktions-Prozess, von welchem aus sie in das Produktlager gelegt und dort kontrolliert werden. Der Abnehmer sendet eine Bestellung in das Produktlager, von wo aus sie an den Produktions-Prozess weitergeleitet wird, sobald ein Arbeiter sein gefertigtes Produkt abgeliefert hat.

Abbildung 5.4: Prozess-Diagramm



Im Diagramm [ProzessDiagramm] werden der Sortier-Prozess und der Produktions-Prozess näher beschrieben. Fliesst ein Item in den Sortier-Prozess, so wird ein Arbeiter aus dem Arbeiter-Pool akquiriert, der das Item sortiert. Nach Ablauf der Sortierzeit wird das Item zum entsprechenden Zwischenlager gesendet und der Arbeiter in den Arbeiter-Pool zurück geschickt. Der selbe Ablauf findet sich im Produktions-Prozess. Ein Item fliesst in den Produktions-Prozess, ein Arbeiter aus dem Arbeiter-Pool wird akquiriert und nach Ablauf der Produktionszeit wird das fertige Produkt zum Produktlager gesendet und der Arbeiter zurück in den Arbeiter-Pool geschickt. Im nicht gemanagten Zustand kann ein Arbeiter mal im Sortier-Prozess und mal im Produktions-Prozess tätig sein. Sobald der Manager den Arbeitern fixe Arbeiten zugeteilt hat, wird der Arbeiter-Pool in zwei Hälften geteilt, was bedeutet, dass jeder Prozess seinen eigenen Arbeiter-Pool besitzt und Arbeiter nur noch einem Prozess dienen können.

5.2 Server-API

Das API des Servers wird durch zwei Endpunkte bestimmt : Eine REST Schnittstelle und ein Websocket-Service.

5.2.1 REST

REST oder RESTful wurde von Roy Thomas Fielding[2] im Jahre 2000 in seiner Doktorarbeit definiert. REST bezeichnet eine auf dem HTTP Protokoll aufsetzende Web-Service-Abstraktion. Dabei wird eine URL als Ressource angesehen. Diese wird mittels verschiedener HTTP-Protokoll-Verben (Wörter) GET, POST, PUT, PATCH, DELETE manipuliert.

Abweichungen von REST

Üblicherweise enthält eine reine REST-Protokoll Implementierung in einer erfolgreichen Serveranfrage mit GET, POST und PUT gleich die möglichen Links, über welche die Resource abgerufen oder weitere Oprationen ausgeführt werden können[1]. In einem System, in dem die Echtzeit eine Rolle spielt, ist auf diese Operationen bewusst verzichtet worden, da diese nicht benötigt werden und nur zu einem Datenoverhead führen.

Die grösste Abweichung vom REST-Protokoll findet sich bei der Implementation der PUT Operation. Das REST-Protokoll besagt, dass für die PUT Operation (Update-Operation) die ganze Resource mit der Veränderung mitgeschickt wird.

Ein Problem mit Echtzeitdaten ist, dass der eine Benutzer noch alte Daten erhält, weil in der Zwischenzeit ein anderer Benutzer etwas verändert hat. Ein servseitiger Lock auf die Schreiboperation ist mit REST unzureichend, da immer die gesamte Resource geholt, manipuliert, und dann im neuen Zustand zurückgeschickt wird. Dadurch kann nicht garantiert werden, dass der Zustand noch aktuell ist zum Zeitpunkt der Update-Funktion. Um dieses Problem zu umgehen, wird nicht die ganze Ressource geschickt, sondern ledglich die Operation, die darauf ausführen werden soll. Diese wird im Payload mitgeteilt. Der angenehmer Nebeneffekt besteht darin, dass die Nachrichten für den Update-Fall sehr wenig Daten beinhalten, die übertragen werden müssen. Wie ein Payload, wie oben beschrieben, aussieht, wird im Abschnitt JavaScript-API erklärt. Die Tabelle REST-Endpunkte zeigt die REST-Artigen Endpunkte des Servers.

REST sieht auch eine Lösch-Operation vor. Diese wurde bewusst abgeschaltet, so entsteht ein gewisser Schutz der Resource, da diese so nur noch im Admin-Bereich gelöscht werden kann.

[1] Häufig ist dies über den OPTIONS-Header erhältlich, der auflistet, welche Operationen auf der angefragten Resource möglich sind.

Tabelle 5.1: REST-Endpunkte

Endpunkt	HTTP-Methode	Parameter	HTTP-Status Code und Antworten
Genereller Eintrittspunkt: "/api/v1/"			
/queues/	get	-	200: Liste mit allen momentan verfügbaren Queues; 404: Endpunkt nicht vorhanden; 500: Server nicht erreichbar
/queues/	post	-	201: Resource wurde erstellt, 500: Server Error, Request konnte nicht verarbeitet werden
/queues/ <id:int>	get	id: eine numerische ID	200: Element (Resource) mit Attributen wird zurückgegeben; 404: ungültige ID
/queues/ <id:int>	put	id: eine numerische ID	201: erfolgreiche manipulation; 200: Anfrgae in Ordnung, Manipulation jedoch fehlgeschlagen; 404: ungültige ID oder Resource nicht vorhanden

Erklärung der verwendeten HTTP Status Codes

200 OK Alles korrekt, wohldefinierte Anfrage.

404 NOT FOUND Anfrage hat zu einem Fehler geführt, und zwar konnte die angeforderte Ressource nicht gefunden werden.

201 CREATED Anfrage ist erfolgreich gewesen, die Ressource wurde erstellt.

204 NO CONTENT Anfrage ist erfolgreich gewesen, es gibt keinen Inhalt in der Antwort.

500 SERVER ERROR Anfrage konnte aufgrund eines (unbekannten) Serverfehlers nicht beantwortet werden.

5.2.2 Websocket

Um den Overhead des Verschicken von Nachrichten zu minimieren, werden alle Queue-Veränderungen während einer Sekunde von einem Hintergrund-Prozess gesammelt und

anschliessend zusammengefasst und in die Datenbank geschrieben. Wenn neue Daten eintreffen, sprich nach jeder Schreib-Operation auf diese spezifische Tabelle, wird ein Push-Event ausgelöst, welcher durch einen Websocket an die Websocket-Listener gesendet wird. Websockets eignen sich in diesem Fall besonders gut, da ein Server-Seitigen-Push von Daten mittels REST-Protokoll nicht möglicht ist.

Diese Daten werden dann mittels der Highcharts-JavaScript-Library auf dem Dashboard dargestellt und bei jedem neu gepushten Datenpunkt aktualisiert. Für diesen Service gibt es keine JavaScrip-API (Middleware), sondern eine direkte Verbindung zum Server.

5.3 JavaScript-API

Die Javascript API bildet die Schnittstelle zwischen dem Server und einem Mini-Game/Applikation.

Sie ist einerseits relativ Applikations-Agnostisch gehalten, wurde für eine einfachere Integration in die Mini-Games mit spezifischem Code ergänzt, damit die Wartung verhältnismässig bleibt und der Code nicht auf noch mehr Repositories aufgeteilt wird.

Um blockierende Aufrufe zu verhindern, wurde die “q” JavaScript Library[2] genutzt.

Applikations-Agnostische Nutzung der API

Die API dreht sich in erster Linie um sogenannte “queues”, welche befüllt und geleert werden können. Dabei kümmert sich das Backend um die Details: Es kommt eine Rückmeldung, wenn die Queue beim Befüllen voll ist oder leer beim Anfordern eines Elementes. Zudem kann die Queue geschlossen sein, dann sind keine Operationen mehr darauf ausführbar.

Queues werden durch einen eindeutigen Namen identifiziert, dieser ist zwingend, die numerische id wird automatisch vom Server erzeugt. Folgende Attribute existieren auf einer Queue:

id numerische Identifikation; wird vom Server vergeben

name Zeichenbasierte Identifikation; eindeutiger Name, zwingendes Attribut

max_capacity numerisches Attribut; gibt an, ab wann die Queue voll ist, -1 bedeutet kein Limit, Optionales Attribut; Default: -1

[2] https://github.com/kriskowal/q

utilization numerisches Attribut; gibt den (momentanen) Stand der Queue an, Optionales Attribut, Default: 0

sealed Wahr/Falsch Attribut; wenn Wahr, können keine Einfüge oder Entnahme Operationen mehr darauf ausgeführt werden; Default: Falsch

Queues können über das Admin-Interface erstellt werden, wenn die nicht gebraucht wird, oder eine dynamischere Generierung erwünscht ist, so kann dies direkt über die API erfolgen.

Eine neue Queue, welche zB den Namen "pink-farbene-socke" tragen soll, mit einer Obergrenze bei 100 "Stück", kann über die API so erzeugt werden:

```
var API = new OctospiceAPI({
      'config': {
         'apiUrl': 'http://server_ip_oder_adresse'
      }
    });
// API ist eine Instanz welche
API.queues.createQueue({
    'name': 'pink-farbene-socke',
    'max_capacity': 100
});
```

Setzen respektive Verändern von *name*, *sealed* oder *max_capacity* ist nur beim Erzeugen möglich über die API. Diese Werte sollen in einer späteren Fassung jedoch im Admin-Interface verändert werden können, um den Spielverlauf zu beinflussen zu können.

Hauptfunktion einer Queue ist, Elemente hinzuzufügen oder zu entfernen oder auch der Queue einen bestimmten Wert zu geben. Dafür dienen folgende Operationen, welche das Resultat nicht blockierend in Form einer JavaScript-Promise zurückliefern:

Tabelle 5.2: Queue Grund-Operationen

Operation	Parameter	Erklärung	Rückgabe
getItem	queueName:string	Ein einzelnes Element aus der Queue herausnehmen	True: Erfolg; False: Misserfolg
getItems	queueName:string, quantity:int	Anzahl *quantity* Elemente aus der Queue herausnehmen	True: Erfolg; False: Misserfolg
putItem	queueName:string	Ein einzelnes Element in die Queue hinein geben	True: Erfolg; False: Misserfolg
putItems	queueName:string, quantity:int	Anzahl *quantity* Elemente in die Queue hinein geben	True: Erfolg; False: Misserfolg
getValue	queueName:string	Aktuelle Anzahl Elemente in der Queue anfragen	int: aktuelle Belegung
setValue	queueName:string, number:int	setzt die *utilization* der Queue auf die Zahl, falls diese innerhalb der Schranke liegt	True: Erfolg; False: Misserfolg
resetValue	queueName:string	stellt die *utilization* der Queue auf 0 zurück	True: Erfolg; False: Misserfolg

Um ein Element in die vorher als Beispiel neu erstellte 'pink-farbene-socke'-Queue zu legen, ist folgender Code nötig:

```
var API = new OctospiceAPI({
      'config': {
         'apiUrl': 'http://server_ip_oder_adresse'
      }
    });
// API gibt eine q-Promise zurück
API.queues.putItem('pink-farbene-socke')
.then(function(result){
    if (result === true) {
        // etwas machen, wenn erfolgreich
        console.log("jippie, ein Socken");
    } else {
```

```
        // etwas machen, wenn Misserfolg
        console.log("ooops, Socken konnte nicht mehr eingetragen werden");
    }
});
```

Mini-Game spezifische Erweiterungen der API

Die API mit den Grundoperationen ist relativ beschränkt. Um eine saubere Anbindung des Mini-Games zu erlauben, gibt es deshalb eine Erweiterung. Diese besteht einerseits aus Events, welche dem Mini-Game mitteilen, ob ein Spiel gestertet, pausiert oder gestoppt ist. Anderseits stellt es vereinfachende Operationen zur Verfügung, welche nur auf den Queues funktionieren, welche im Szenario des Mini-Games vorhanden sind.

Spielzustand Weitergeben

Um zu wissen, ob das Spiel bereits gestartet ist, oder zum Beispiel pausiert, ohne dass ein Wechsel dieses Zustandes stattgefunden hat, dienen die Abfrage-Funktionen:

started() gibt Boolean zurück, welcher signalisiert, ob das Spiel gestartet ist oder nicht

running() Alias für started

paused() gibt Boolean zurück, welcher signalisiert, ob das Spiel pausiert ist

stopped() gibt Boolean zurück, welcher signalisiert, ob das Spiel beendet ist; Wenn wahr: finaler Zustand, kann nur durch neues Spiel wechseln

Um Zustandsänderungen des Spiels, zum Beispiel von gestarted zu pausiert, auch an das Mini-Game weiter zu geben, werden Events verwendet, an welchen sich ein Mini-Game anmelden kann. Die Tabelle 5.3 beschreibt die möglichen Events.

Tabelle 5.3: Events der JS-API

Event Name	Rückgabewert	Beschreibung
startChanged	Boolean, ob gestartet oder nicht	Spielzustand wechselte von nicht gestartet auf gestartet; wird auch ausgelöst, wenn ein Spieler an einem bereits laufenden Spiel teilnimmt
pauseChanged	Boolean, ob pausiert oder nicht	Das Spiel wechselt von pausiert auf unpausiert
stopChanged	Boolean, ob das Spiel schon gestoppt wurde	Das Spiel ist gestoppt
queuesCounter Updated alias *queuesUpdated*	Javascript Liste mit allen verfügbaren Queues als Inhalt	Der aktuelle Zustand aller verfügbaren Queues wurde erneuert

Beispiel für eine Event-Anmeldung um auf den Start-Event zu hören:

```
var API = new OctospiceAPI({
      'config': {
         'apiUrl': 'http://server_ip_oder_adresse'
      }
    });
// Funktion für callback
var gameStartEvent = function(value){
    // liefert den boolean aus dem Callback zurück
    console.log("value hat den Wert: ", value);
};
// API ist eine Instanz welche
API.on('startChanged', gameStartEvent);
```

Vereinfachende Operationen auf Szenario-Spezifischen-Queues

Spezielle Operationen sind nur im Falle eines Mini-Games sinnvoll nutzbar, da diese davon ausgehen, dass die entsprechenden Queues vorhanden sind:

- getNext() -> String: “queue-Name”: gibt ein nächstes freies Element aus einer beliebigen, globalen Queue zurück
- getNextPattern() -> List<String>: [“color-1”, “color-2”...]: gibt ein nächstes freies Pattern (Liste von Farben) aus einer beliebigen, globalen Produktions-Queue zurück

Auch hier kommt der Nicht-Blockierende Aufruf zum Einsatz, die Rückgabe ist eine q-Promise und kann mit ".then" abgeholt werden.

Ein Beispiel zur Nutzung der API kann im Kapitel Nutzung des Mini-Frameworks gefunden werden.

Kapitel 6

Verwendete Technologien

6.1 MiniGames

Die drei Mini-Games Einkaufs-Game, Produktions-Game und Abnahme-Game sind alle in JavaScript geschrieben und verwenden zur Unterstützung das Game-Framework Phaser. Das Dashboard des BigScreens ist mit Angular und HighCharts umgesetzt, während der Anmeldescreen in HTML gehalten ist. Nachfolgend findet sich eine kurze Beschreibung der eingesetzen Frameworks und Technologien und wofür diese grundsätzlich gebraucht werden. Dabei wird zwischen Tools zur direkten Umsetzung der Aufgabenstellung und Tools zur Vereinfachung der Entwicklungsarbeit unterschieden.

Tools zur Umsetzung der Aufgabenstellung

Phaser Phaser ist ein in Javascript geschriebenes open-source Game-Framework um Desktop- und Mobile-Browser-HTML5-Games zu erstellen. Das Framework Phaser basiert auf dem Framework Pixi.js, welches ein 2D-Rendering mit webGL oder Canvas realisiert.

Phaser zeichnet sich vorallem aus durch:

- eine aktive Community mit eigenem Forum
- guten Getting-Started-Guides
- vielen Demo-Games und Examples für einzelne Features

Am 23. Oktober 2014 kam die neue Version “Ravinda” 2.1.3 heraus, was zeigt, dass der Entwickler “Photon Storm” das Framework noch immer aktiv pflegt und erweitert.

Phaser kann intern sowohl mit webGL als auch mit Canvas arbeiten und bietet vollen Physik-Support basierend auf Ninja Physics und P2.js. Durch den bereitgestellten Preloader können alle Sourcen im Voraus geladen werden, so dass das Spiel anschliessend fliessend läuft.

Tools zur Vereinfachung der Entwicklungsarbeit

Node Nodejs ist ein in Javascript geschriebener Webserver mit einem Package Manager (NPM). Dieser erlaubt dem Entwickler mittels Paketen, die nachinstalliert werden können, zusätzliche Funktionalität zu bekommen. Viele der Pakete haben jedoch untereinander Abhängigkeiten, was eine manuelle Installation schwierig macht. NPM installiert alle Abhängigkeiten eines gewünschten Pakets selbständig, der Entwickler muss nur mittels des Befehls "npm install [package]" angeben, welches Paket zu installieren ist. Der Nodejs-Server selbst wird nicht gebraucht, kann bei der lokalen Entwicklung jedoch eingesetzt werden.

Yeoman-Generator Mit Hilfe von NPM kann der Yeoman-Generator "phaser-browserify" installiert werden. Mit diesem Generator kann ein Skelett für ein Phaser-Game, welches Browserify nutzt, um den JavaScript-Code zu strukturieren, erstellt werden.

Browserify Browserify kann mittels des Befehls "npm install -g browserify" mit Hilfe des Package Managers NPM von Node installiert werden und ist ein Tool um "node-flavored" commonjs Module für Browser zu kompilieren. Es kann verwendet werden, um den eigenen Code zu verwalten und "third-party"-Libraries zu verwenden, ohne den Nodejs-Server selbst einzusetzen. Das Modul-System, welches Browserify verwendet, ist das gleiche, das auch Nodejs-Server verwendet - dadurch können auch Pakete, welche ursprünglich für den Nodejs-Server erschienen sind und nicht für Client-Side gedacht sind, nun in Browsern verwendet werden.

Bower Bower ist ein Paket Manager Tool für "third-party" Web-Libraries. Früher musste der Entwickler manuell zur Website des Publishers navigieren, das entsprechende File herunterladen, entpacken und ins eigene Projekt einbinden. Bower vereinfacht diesen Prozess durch das zur Verfügungstellen des Befehls "bower install".

Gulp/Grunt Grunt wie Gulp sind JavaScript Task Runner. Sie helfen beim Automatisieren von repetitiven Aufgaben wie minifizieren, kompilieren, unit testing

und so weiter. Im Unterschied zu Grunt verwendet Gulp Pipes für Datenströme, welche verarbeitet werden müssen. Gulp ist etwas neuer und einfacher zu benutzen, viele "third-party"-Libraries verwenden zur Zeit jedoch noch Grunt.

Angular Angular ist eines der momentan am häufigsten eingesetzten Javascript-Frameworks und dient in erster Linie zur Erstellung von dynamischen Single-Page-Apps[1].

6.2 Backend

Das Backend ist in Python geschrieben und verwendet einige Libraries und Frameworks. Die Betriebsumgebung ist auf Linux beschränkt, da es Abhängigkeiten zu Linux System-Libraries gibt.

6.2.1 Kurzbeschreibung der eingesetzten Libraries und Frameworks

Flask ist ein Micro-Webframework für Python mit wenigen Verwendungs-Vorschriften. Es liefert Grundbausteine für die Programmierung eines Webservers für dynamische HTML Seiten. Zusätzliche Anforderungen wie Datenbank-Anbindung, Websocket-Unterstützung und RESTful-Webservices können per Plugin dazu installiert werden. Die Datenbank kann dank dem ORM SQLAlchemy frei gewählt werden. Postgres ist eine Opensource Datenbank, welche dank JSON und HSTORE auch nicht-relationale Features anbietet. Dies macht Postgres zu einer guten Wahl für die Verwendung mit SQLAlchemy.

Flask bietet eine WSGI-konforme API an und kann so mit Gunicorn deployed werden. Gunicorn ist ein dünner HTTP-Webserver-Wrapper um das WSGI-Protokoll. Dank des in Gunicorn konfigurierbaren asynchronen Servierens wird die Durchsatzrate erheblich erhöht.

Celery ist eine verteilte Task-Warteschlange mit Redis[2] als Warteschlangendatenbank. Die angestauten Tasks werden mit vorher definierten Workers abgearbeitet. Zudem kann Celery einen sogenannten Heartbeat in einem bestimmten Zeitintervall senden. So können zeitliche Aufgaben in Auftrag gegeben werden.

[1] Multi-Page Apps sind mit jedem dieser Frameworks auch machbar, die Stärke der Frameworks liegt eindeutig bei Single-Page-Apps.

[2] Auch mit RabbitMQ und anderen möglich.

Nginx ist ein mittlerweile sehr populärer Webserver. Er wird als ReverseProxy für Websockets, REST und dynamische HTML-Seiten verwendet und dient als schneller statischer Content Server.

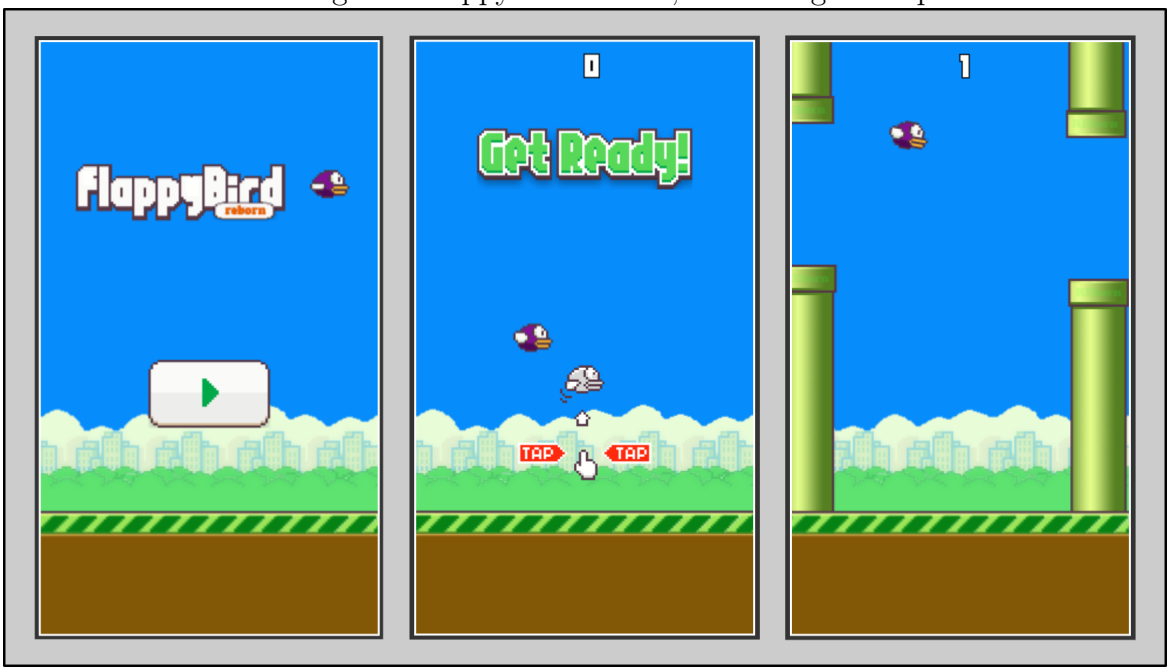
Kapitel 7

Nutzung des Mini-Frameworks

Das Mini-Framework besteht aus einem Backend und einem JavaScript Schnittstellen-Programm, welches zur Steuerung des Spielverlaufs dient und entsprechende Rückmeldungen gibt, so dass auf dem Dashboard zum Beispiel der aktuelle Spielstand angezeigt werden kann. Das Framework macht jedoch grundsätzlich keine Annahmen bezüglich Art und Verhalten bei der Verwendung. Statt eines Spieles könnte auch ein anderes System, welches ein Dashboard mit Anzeige des aktuellen Zustandes verwenden möchte, auf dem Framwork aufbauen. Im folgenden wird beschrieben, wie das Framework allgemein genutzt werden kann. Wie das Flappy Bird Tutorial ergänzt werden muss um es als ein Mini-Spiel zu nutzen, wird im Kapitel Code Dokumentation, Abschnitt Beispiel-Integration “Flappy Bird” als Produktionsspiel beschrieben.

7.1 Flappy Bird Beschreibung

Abbildung 7.1: Flappy Bird: Start, Erklärung und Spiel



Kurzbeschrieb Flappy Bird Flappy-Bird ist ein einfaches, frustreiches Spiel, bei welchem das Ziel ist, den Vogel sicher durch die Aussparungen zwischen den Röhren hindurch zu navigieren. Der Vogel kann durch Berühren des Touchbildschirms (oder drücken der Maustaste auf einem Desktopgerät) aufwärts bewegt werden, und kommt automatisch wieder herunter. Berührung mit dem Boden oder mit einer Röhre führen zu Crashes, welche es zu vermeiden gilt.

7.2 Allgemeine Nutzung

Um zu demonstrieren wie sich das entwickelte Framework allgemein nutzen lässt, wird als Beispiel im Folgenden ein einfaches Leaderboard [1] für den Flappy-Bird Klon aus dem Phaser Tutorial erstellt. Dafür werden zwei grundlegende Schritte benötigt:

- API-Integration: Ergänzen des Codes im “play.js” um den globalen Highscore über alle Spieler mitzuteilen
- Dashboard: Nutzung des integrierten Dashboards um eine aktuelle Anzeige des Highscores zu erhalten

[1] Einzig der höchste Highscores aller Spieler wird angezeigt.

7.2.1 API Integration

Queues über die API erstellen

In diesem Abschnitt wird nur erklärt, wie eine neue Queue erstellt werden kann. Auf die weiteren Funktionen der entwickelten API wird im Kapitel Architektur ausführlicher eingegangen. Um eine Queue zu erstellen, wird der folgende Aufruf benötigt:

_oAPI.queues.createQueue(queueParameter)

Dabei werden die folgenden Parameter verwendet:

_oAPI: Instanz der OctocubeAPI

queueParameter Objekt mit dem zwingenden Attribut *name* und den optionalen Attributen *max_capacity, utilization* und *sealed*

name: Benennung, eindeutige Identifizierung, auch für spätere Operationen

max_capacity: (optional) maximale Kapazität der zu erstellenden Queue; ist diese erreicht, können keine neuen Elemente eingefügt werden können; Default: -1, was unendliche Kapazität entspricht

utilization: (optional) Vorbelegung; die gewünsche Vorbelegung kann mit einem Wert > 0 angegeben werden; Default: 0

sealed: (optional) Flag um die Entnahme- und Zugabe-Operationen auf die zu erstellende Queue zu blockieren; Default: False

Im folgenden Beispiels-Code wird das zuvor genannte Spiel “Flappy Bird” aus dem Phaser-Tutorial als Beispiel verwendet. Diesem Spiel soll ein globales Dashboard hinzugefügt werden, welches den High-Score über alle Spieler und optional die Anzahl der “Crashed Birds” anzeigt.

Um die JavaScript-API einbinden zu können, muss diese zuerst au dem BitBucket-Repository https://bitbucket.org/octospice/octocube-js-api.git heruntergeladen und anschliessend mittels script-Tag in das HTML eingebunden werden. Alternativ kann dies auch via einer entsprechenden Ergänzung im Bower-File geschehen. In diesem Beispiel wird die Variante “Ergänzung im Bower-File” demonstriert:

```
// bower.json
{
  "name": "flappy-bird-reborn",
  "version": "0.0.0",
  "dependencies": {
    "phaser-official": "2.0.3", // Komma ergänzen
    "octocube-js-api": "git@bitbucket.org:octospice/octocube-js-api.git" // diese Zeile hinzufügen
  }
}
```

Nach dem Abändern des Bower-Files kann auf der Konsole der Befehl *"bower install"* ausgeführt werden. Anschliessend kann das Skript für die API im File "*index.html*" zwischen den bereits vorhandenen Anweisungen eingebunden werden.

```
/* index.html */
/* ... */
  <script src="js/phaser.js"></script>
  <script src="js/octocube-js-api.js"></script>
  <script src="js/game.js"></script>
/* ... */
```

Wird Bower mit einem Build-Tool wie Grunt oder Gulp verwendet, muss dieses zusätzlich noch entsprechend angepasst werden, so dass die API auch gefunden wird. Damit der Grunt-Befehl auch das API kopiert, muss dies im Grunt-File entsprechend ergänzt werden:

```
// Gruntfile.js
// ...
    copy: {
      prod: {
        files: [
          // includes files within path and its sub-directories
          { expand: true, src: ['assets/**'], dest: 'dist/' },
          { expand: true, flatten: true, src: ['game/plugins/*.js'], dest: 'dist/js/plugins/' },
          { expand: true, flatten: true, src: ['bower_components/**/build/*.js'], dest: 'dist/js/' },
          // folgende Zeile Ergänzen
          { expand: true, flatten: true, src: ['bower_components/**/dist/*.js'], dest: 'dist/js/' },
          { expand: true, src: ['css/**'], dest: 'dist/' },
          { expand: true, src: ['index.html'], dest: 'dist/' }
        ]
      }
    },
// ...
```

In einem nächsten Schritt muss im File "play.js" eine Instanz der JavaScript-API erstellt werden. Im Beispiel sieht dies folgendermassen aus[2]:

```
// play.js
// ...
Play.prototype = {
 _oAPI: this._oAPI || new OctocubeAPI({
    'config': {
      'apiUrl': window.location.protocol + '//' + window.location.hostname + ':' + window.location.port,
      'noUpdate': true
    }
  }),
  // ...
```

Nun sind alle benötigten Bestandteile hinzugefügt um auf den laufenden Server zuzugreifen.

Jetzt werden die gewünschten "queues" erstellt, so dass diese anschliessend ansprechbar sind:

```
// play.json
//...
  create: function(bird, enemy) {
// ...
  // Am Ende der Funktion Zeile ergänzen
    this._oAPI.queues.createQueue({'name': 'score'}).done();
```

[2]Zu beachten: da es kein Update des Servers braucht, wird das regelmässige Update in den Einstellung bei der Instanziierung explizit unterbunden.

```
    // Wenn Optionaler Zähler für Anzahl tote Vögel gewünscht
    this._oAPI.queues.createQueue({'name':'dead-birds'}).done();
  },
//...
```

Anschliessend muss noch die gewünschte Spiellogik hinzugefügt werden. In diesem Beispiel soll jeweils der neue High-Score an den Server geschickt werden:

```
// play.json
//...
  if(enemy instanceof Ground && !this.bird.onGround) {
        this.groundHitSound.play();
        this.scoreboard = new Scoreboard(this.game);
        this.game.add.existing(this.scoreboard);

// folgende Zeilen ergänzen
        var score = this.score;
        var that = this;
        this._oAPI.queues.getValue(this.name)
          .then(function(result){
          if (result < score) {
            that._oAPI.queues.setValue(that.name, score).done();
          }
        });
// Ende Ergänzung
        this.scoreboard.show(this.score);
        this.bird.onGround = true;
    } else if (enemy instanceof Pipe){
        this.pipeHitSound.play();
    }
//...
```

Optional kann die Anzahl der "Crashed Birds" aufgezeichnet werden, so dass auch diese Grösse auf dem Dashboard ausgegeben werden kann. Der folgende Code meldet die Anzahl der "Crashed Birds" dem Server.

```
// play.json
// Optional: Tote Vögel zählen
//...
  deathHandler: function(bird, enemy) {
    // ...
    if(!this.gameover) {
// folgende Zeile ergänzen
        this._oAPI.queues.putItem('dead-birds').done();

        this.gameover = true;
        this.bird.kill();
        this.pipes.callAll('stop');
        this.pipeGenerator.timer.stop();
        this.ground.stopScroll();
    }
//...
```

Im nächsten Abschnitt wird erklärt, wie die Inhalte der erstellten Queues auf einem Dashboard dargestellt werden können.

7.2.2 Dashboard

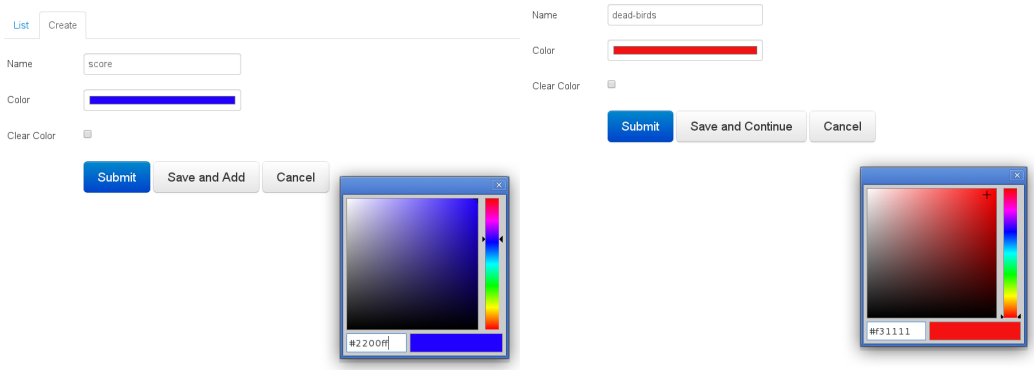
Das Dahboard muss für eine breitere Nutzung noch erweitert werden. Anhand des folgenden Beispiels wird erklärt, wie ein neues, personalisiertes Dashboard erstellt werden kann.

Dashboard erstellen

Um ein eigenes neues Dashboard zu kreieren, muss in das Admin-Menu innerhalb des geschützen Bereichs gewechselt werden. Ein Passwort kann im Deployment-Prozess festgelegt werden und muss zu diesem Punkt gesetzt und bekannt sein. Auf das Admin-Menu kann üblicherweise via <url-des-servers>/admin/ oder mittels des direkten Links in der oberen Navigationsleiste zugegriffen werden.

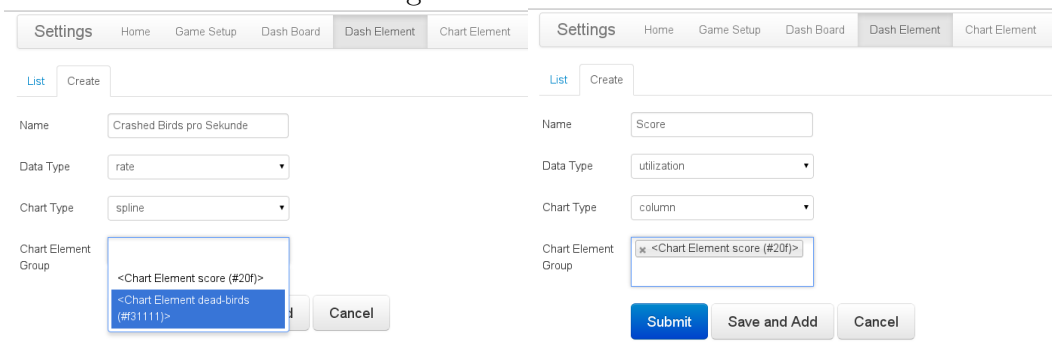
Als erster Schritt wird das Dashboard-Chart-Element erstellt. Hier können die Farben der Chart-Linien und der Balken des Balkendiagramms festgelegt werden. Im folgenden Beispiel werden die Farben rot und blau gewählt.

Abbildung 7.2: Backend: Auswahl für Chart-Linien einer Queue



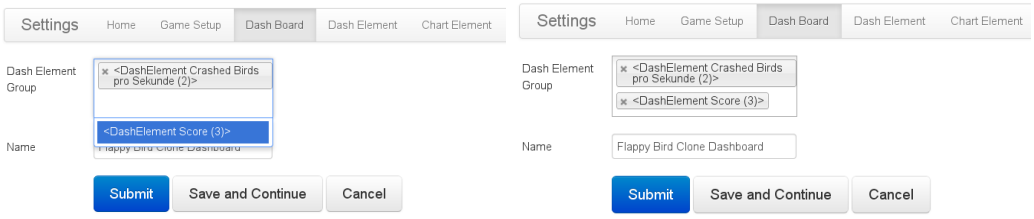
Anschliessend werden zwei Dashboard-Elemente erstellt. Diese zeigen die im ersten Schritt erstellten Chart-Elemente anschliessend dar. Das eine Dashboard-Element wird die Rate der “Crashed Birds” pro Sekunde anzeigen, das zweite Dashboard-Element den momentanen High-Score. Die benötigten Einstellungen sind in der Abbildung 7.3 dargestellt.

Abbildung 7.3: Backend: Chartwahl



In einem letzen Schritt müssen die soeben erzeugten Charts dem Dashboard hinzugefügt werden, damit sie angezeigt werden können.

Abbildung 7.4: Backend: Charts zu Dashboard hinzufügen

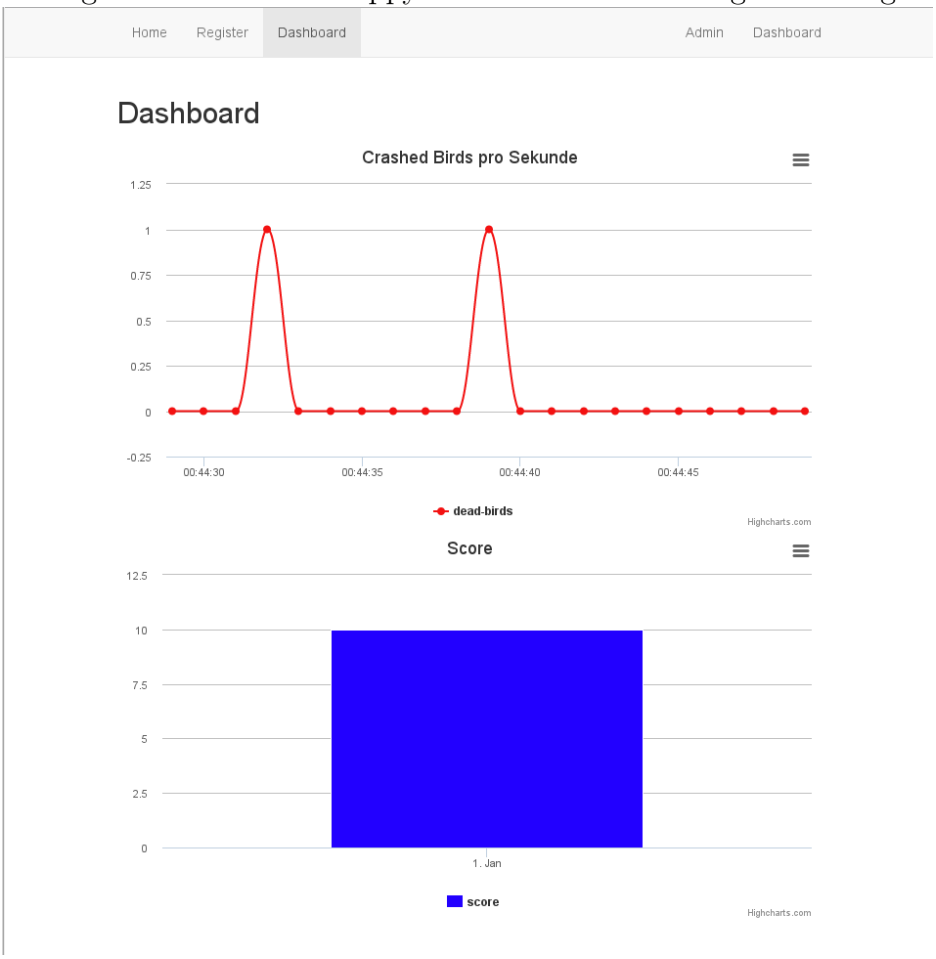


Nach dem Speichervorgang, kann das neue Dashboard unter der entsprechenden ID abgerufen werden:

/dashboard/special/<id>/

Das in diesem Beispiel erstellten Dashboard ist in der Abbildung 7.5 dargestellt.

Abbildung 7.5: Dashboard Flappy Bird Death Rate und globaler Highscore



Kapitel 8

Code Dokumentation

8.1 Mini-Games

8.1.1 Assets

Als Assets werden die grafischen Elemente eines Spiels bezeichnet. In diesem Projekt sind dies die einzelnen Bilder, die im Spiel verwendet werden.

8.1.2 Prefabs

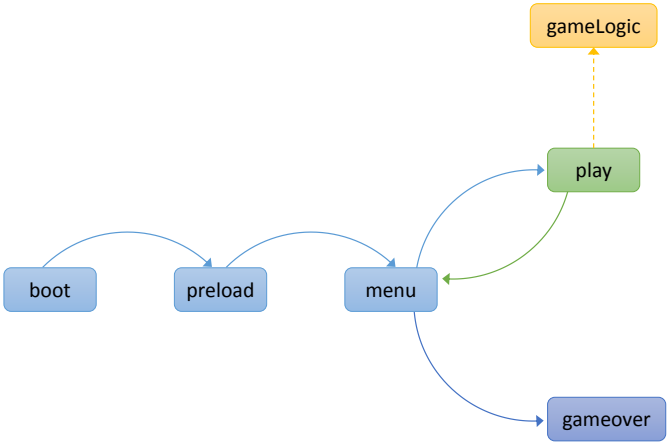
Prefab bedeutet “pre-fabricated”, also vor-fabriziert. Wird ein Objekt in einem Spiel mehrere Male gebraucht, so sollte ein Prefab erstellt werden. Dieses Prefab kann anschliessend so oft instanziiert werden, wie es gebraucht wird.

8.1.3 States

Ein Phaser-Game besteht aus einer Abfolge verschiedener States (mindestens ein State ist zwingend). Da Phaser einen integrierten State-Manager besitzt, ist die Registrierung und das Management relativ einfach gehalten. Jeder State kann eine eigene Preload-, Create- und Update-Methode implementieren.

[GameStatesDiagramm]

Abbildung 8.1: Game-States-Diagramm



In diesem Projekt werden für das Produktions-Game die folgenden States verwendet:

boot Der Boot-State ist der erste State, der aufgerufen wird. In ihm werden die initialen Einstellungen (zum Beispiel das korrekte Scaling für das entsprechende Display) vorgenommen, um das Game starten zu können.

preload Der Preload-State wird anschliessend an den Boot-State aufgerufen und lädt alle Assets in das Game, so dass die Ladezeiten während des Game massiv verkürzt und das Game schneller gestartet werden kann. Sobald alle Assets geladen sind, wird der Menu-State aufgerufen.

menu Im Menu-State wird die JS_API Octospice gestartet. Während diese lädt, wird dem Spieler der Text 'Waiting for Game-Start' angezeigt. Sobald die JS_API Octospice bereit ist, wird der Play-State aufgerufen. Der Benutzer muss bis zum jetzigen Zeitpunkt noch keine Interaktion mit dem Spiel vornehmen.

gameLogic Im GameLogic-State befindet sich die eigentliche Spiellogik. Dieser State ist zuständig für alle spielrelevanten Aufgaben wie das initiale Erstellen der Baskets und eines Parts, das zur Verfügung stellen von Patterns, welche vom Spieler gebaut werden können und das anschliessende Auswechseln, sobald eines komplett ist. Hier befindet sich die Logik für das Handling von Benutzerinteraktionen

wie Drag&Drop eines Parts auf einen Basket (um es ins Zwischenlager zu verschieben) oder auf ein AssemblyPartItem (um das Pattern zu vervollständigen) oder das Tappen auf einen der Baskets, um ein Part aus dem Zwischenlager zu holen. Auch das Overlap-Handling wird in diesem State ausgeführt.

play Der Play-State erbt vom GameLogic-State, so dass der Play-State alle für die Spiellogik relevanten Operationen durchführen kann. Im Menu-State muss deshalb nur der Play-State aufgerufen werden. Im Play-State wird das Phaser-System gestartet, die Settings übernommen und die Methoden paused() und shutdown() definiert.

gameover Dieser State wird aufgerufen, sobald das Spiel beendet worden ist. Dem Spieler wird eine 'Game Over'-Nachricht angezeigt.

8.1.4 Phaser-Groups

In Phaser können mehrere DisplayObjects (zum Beispiel Sprites oder Images) zu einer sogenannten Group zusammengefasst werden. Wird eine Operation (Scaling, Moving, Rotating) auf einer Group ausgeführt, so wird sie automatisch auch auf allen Children der Group ausgeführt. Zusätzlich unterstützen Groups fast pooling und object recycling. Groups werden in Phaser auch als DisplayObjects erkannt und können somit ineinander verschachtelt werden. Ausserdem wird mittels Groups das Layering vereinfacht, da der Befehl 'bringToTop' auf eine ganze Group angewendet werden kann, ebenso wie 'destroy' die Group sowie alle zugehörigen Children zerstört. So kann sichergestellt werden, dass nicht einzelne Objekte vergessen werden.

8.1.5 Overlapping

Phaser stellt via Physics.Arcade eine Overlap-Methode zur Verfügung, die zwei Objekte (Sprites, Groups, etc.) und eine Callback-Methode, die im Falle eines Overlaps ausgeführt werden soll, annimmt. Die Methode ist jedoch nicht rekursiv, und testet nicht gegen Children von übergebenen Objekten. In diesem Fall ist aber ein Test eines Parts gegen jeden einzelnen Basket und jedes einzelne AssemblyPatternItem notwendig, die aber beide in der jeweiligen Gruppe (BasketGroup, AssemblyPattern) zusammengefasst sind. Um die gewünschte Überprüfung zu erhalten, hätte über beide Gruppen iteriert werden müssen.

Eine zweite Möglichkeit eine Überprüfung gegen jedes einzelne Objekt durchzuführen besteht in der Verwendung der Phaser-Methode 'getObjectsUnderPointer'. Dieser

Methode muss ein Pointer (ein Phaser-eigenes Objekt, welches einen Punkt markiert, in diesem Fall den Finger des Spielers), die zu testende Gruppe (hier die BasketGroup oder das AssemblyPattern), eine Callback-Methode und ein Kontext (in diesem Fall das entsprechende Part) mitgegeben werden. Die Methode überprüft, ob eines der Children der Group ein Overlapping mit den Koordinaten des Pointers aufweist. Somit wird die Group nicht gegen das bewegte Objekt (das Part) sondern gegen den Finger-Point des Benutzers abgeglichen, was aber zum gleichen Effekt führt, da sich das zu bewegende Objekt (das Part) immer unter dem Finger des Benutzers befindet.

8.1.6 Wichtigste Objekte

Das Mini-Game beinhaltet zwei interagierbare Game-Objects: ein bewegbares - das Part - und ein klickbares - den Basket. Zusätzlich befinden sich mehrere AssemblyItemParts auf dem Spielfeld, zusammengefasst zu einem AssemblyPattern. Nachfolgend sind diese vier wichtigsten Game-Objects kurz näher erläutert.

Basket

Basket symbolisiert den Eingang in die Zwischenlager und besitzt als Anzeigebild denselben Rhombus wie das Part, jedoch grösser skaliert. Basket erbt von der Phaser-eigenen Klasse Sprite, somit kann ein Basket einer Group hinzugefügt werden und besitzt eine eigenen Update-Methode. Ausserdem hat der Basket den Event 'onInputDown' registriert, so dass der Benutzer ein Part, das zuvor in den Zwischenspeicher verschoben wurde, mittels Touch auf den Basket auch wieder herausholen kann.

Part

Ein Part entspricht einem Material. Das Objekt kennt alle Baskets und alle Elemente des Patterns. Ein Part kann mittels Drag&Drop bewegt werden. Sobald die Drag-Geste beendet ist (das Part also "gedropped" wird) überprüft es, ob ein Overlap mit einem Basket oder dem Pattern besteht. Auch Part erbt von der Phaser-eigenen Klasse Sprite.

AssemblyPattern

Das AssemblyPattern entspricht dem Produkt. Es besteht aus verschiedenen AssemblyPatternItems, welche die einzelnen Positionen und Materialien innerhalb eines Produkts darstellen. Das AssemblyPattern berechnet die Position seiner AssemblyPatternItems selbst und kann auch überprüfen, ob es komplett zusammengebaut wurde.

AssemblyPatternItem

Das AssemblyPatternItem erbt ebenfalls wie der Basket und das Part von Sprite, der Phaser-eigenen Klasse für sichtbare und manipulierbare Game-Objects. Sobald ein Part auf ein AssemblyPatternItem gezogen wird, "locked" das AssemblyPatternItem das entsprechende Part auf sich fest, indem es dessen Farbe überprüft und falls nötig die Farbe des Parts annimmt.

8.2 Octocube JS API

Die Octocube JS API stellt die Schnittstelle zwischen den Games und dem Server (Backend) dar.

Diese verwendet selber verschiedene Libraries, welche hier erläutert werden. Beispiele zur Nutzung der JS-API werden unter Beispiel-Integration "Flappy Bird" als Produktionsspiel und Nutzung des Mini-Frameworks beschrieben, die Erklärungen zur API selbst findet sich im Architekturdokument unter JavaScript-API.

Amygdala

Amygdala ist eine Library, um REST-API von JavaScript aus einfacher anzusprechen können.

Amygdala, https://github.com/lincolnloop/amygdala/, wurde von LincolnLoop als OpenSource Software veröffentlicht.

q

q ist eine Implementation von Promises in JavaScript, die auch vor dem Erscheinen von EcmaScript 6 schon auf allen moderneren Browsern genutzt werden kann. Alle Funktionsaufrufe, welche tendenziell eine Verzögerung (vorallem AJAX) haben könnten, werden als Q-Promises zurückgegeben.

q, https://github.com/kriskowal/q, wurde von Kristopher Michael Kowal als OpenSource Software veröffentlicht.

EventEmitter

EventEmitter ist eine Library, welches es einfach ermöglicht, eigene Events in JavaScript zu senden und erlaubt eine einfacher Workflow für die Listener.

EventEmitter, https://github.com/Wolfy87/EventEmitter, wurde von Wolfy87 als Public Domain Software veröffentlicht.

Underscore

Underscore ist eine sehr bekannte und häufig benutzte JavaScript Helfer-Library, welche bei den gängigsten Manipulationen von Arrays und darüber hinaus oft verwendete JavaScript Patterns, wie zB throttle, zur Verfügung stellt.

Underscore, https://github.com/jashkenas/underscore, wurde von Jeremy Ashkenas, DocumentCloud and Investigative Reporters & Editors entwickelt und durch viele weitere Personen weiterentwickelt und ist als OpenSource Software veröffenlticht.

8.3 Beispiel-Integration "Flappy Bird" als Produktionsspiel

Der Kurzbeschrieb zu Flappy Bird ist im Kapitel "Nutzung des Mini-Frameworks" im Abschnitt "Flappy Bird Beschreibung" zu finden.

Der generelle Vorgehensplan für die Integration des Spiels lautet:

0 (Spiel erstellen)

1 API anbinden

- **1.1** API JavaScript einbinden
- **1.2** Start, Stop und Pause von Server auslösen lassen
- **1.3** Produktion und Fehler registrieren

Die anschliessenden Schritte:

2 Bei Backend registrieren, um als mögliche Anwendung aufzutauchen

3 Deployen

sind im Kapitel Deployment zu finden.

1 API anbinden

Für eine Integration mit der JavaScript-API von Octocube sind Vereinfachungen nötig, da das Spiel nicht alle Anforderungen erfüllt (es gibt keinen separaten Sortier- und Produktionsprozess). Der einfachere Fall, dass das Spiel "nur" den Sortier-Prozess beschreibt, wird nicht beschrieben, da dieser trivial ist:

- Jede generierte Röhre stellt ein zu sortierendes Element dar

  - Erfolgreich sortiert entspricht dem Passieren der Röhren
  - Fehler bei Crash

Um den komplizierteren Fall des Produktionsprozesses zu beschreiben, müssen jedoch folgende Überlegungen zum Spielablauf gelten:

- Ein neues Röhren-Paar wird generiert, sofern eines vorhanden ist
  - Diese Produktion verbraucht nur ein Material-Element aus der globalen Queue
- Fehler (Crash) gelten nur für den Produktionsprozess
- Passieren von einer Röhre gilt als Produktions-Erfolg

1.1 API JavaScript einbinden

Zuerst muss das Spiel die JavaScript-API kennen. Dazu muss sie als Bower Dendency hinzugefügt werden:

```
// bower.json
{
  "name": "flappy-bird-reborn",
  "version": "0.0.0",
  "dependencies": {
    "phaser-official": "2.0.3", // Komma ergänzen
    "octocube-js-api": "git@bitbucket.org:octospice/octocube-js-api.git" // diese Zeile hinzufügen
  }
}
```

Dann auf der Konsole *“bower install”* ausführen und im “*index.html*” zwischen den vorhandenen Zeilen das Skript für die API einbinden:

```
/* index.html */
/* ... */
  <script src="js/phaser.js"></script>
  <script src="js/octocube-js-api.js"></script>
  <script src="js/game.js"></script>
/* ... */
```

Damit der Grunt-Befehl auch das API kopiert, muss dies im Grunt-File ergänzt werden:

```
// Gruntfile.js
// ...
    copy: {
      prod: {
        files: [
          // includes files within path and its sub-directories
          { expand: true, src: ['assets/**'], dest: 'dist/' },
          { expand: true, flatten: true, src: ['game/plugins/*.js'], dest: 'dist/js/plugins/' },
          { expand: true, flatten: true, src: ['bower_components/**/build/*.js'], dest: 'dist/js/' },
          // folgende Zeile Ergänzen
          { expand: true, flatten: true, src: ['bower_components/**/dist/*.js'], dest: 'dist/js/' },
          { expand: true, src: ['css/**'], dest: 'dist/' },
          { expand: true, src: ['index.html'], dest: 'dist/' }
        ]
```

```
      }
    },
// ...
```

Damit auf das API zugegriffen werden kann, muss dieses überall instanziiert werden, wo es verwendet wird:

```
// menu.js
// ...
Menu.prototype = {
 _oAPI: this._oAPI || new OctocubeAPI({
    'config': {
      'apiUrl': window.location.protocol + '//' + window.location.hostname + ':' + window.location.port
    }
  }),
  // ...


// play.js
// ...
Play.prototype = {
 _oAPI: this._oAPI || new OctocubeAPI({
    'config': {
      'apiUrl': window.location.protocol + '//' + window.location.hostname + ':' + window.location.port
    }
  }),
  // ...
```

1.2 Start, Stop und Pause von Server auslösen lassen

Als erstes muss die Funktion, dass das Spiel automatisch pausiert, deaktiviert werden, wenn es den Fokus verliert. Dazu muss lediglich eine Zeile im "*preload.js*" eingefügt werden:

```
// preload.js
//...
 preload: function() {
//...
    this.load.audio('score', 'assets/score.wav');
    this.load.audio('ouch', 'assets/ouch.wav');
    this.load.bitmapFont('flappyfont', 'assets/fonts/flappyfont/flappyfont.png', 'assets/fonts/flappyfont/flappyfont.fnt');
// folgende Zeile hinzufügen
    //don't stop update loop when loosing focus
    this.game.stage.disableVisibilityChange = true;
// ...
 }
```

Damit der Start nicht mehr durch den Spieler ausgelöst werden kann, sondern über den Server erfolgt, müssen folgende Zeilen aus Schritt 6 vom "*menu.js*" geändert werden:

```
// menu.js
// ...
   /** STEP 6 **/
    //  create an oscillating animation tween for the group
    this.game.add.tween(this.titleGroup).to({y:115}, 350, Phaser.Easing.Linear.NONE, true, 0, 1000, true);

// Änderungen:
   // Start-Button entfernen und mit einem Text ersetzen
    this.waitForStart = this.game.add.text(this.game.width/2, 250, 'Waiting for Start');
    this.waitForStart.anchor.setTo(0.5,0.5);
   // surrender start event handling to server
    var that = this;
   // auf den Event "started" hören, um das Spiel zu starten und wenn das Spiel nicht auf stopped ist, das spiel starten
    this._oAPI.on('startChanged', function(shouldStart){
```

```
      if ((that._oAPI.stopped() !== true) && (shouldStart === true)) {
        that.game.state.start('play');
      }
    });
// ...
```

Zur Vereinfachung wird der Instruktionsteil im “*play.js*” des Spiels nicht mehr angezeigt:

```
// play.js
// ...
    this.scoreText = this.game.add.bitmapText(this.game.width/2, 10, 'flappyfont',this.score.toString(), 24);
// Zeilen löschen:
    //this.instructionGroup = this.game.add.group();
    //this.instructionGroup.add(this.game.add.sprite(this.game.width/2, 100,'getReady'));
    //this.instructionGroup.add(this.game.add.sprite(this.game.width/2, 325,'instructions'));
    //this.instructionGroup.setAll('anchor.x', 0.5);
    //this.instructionGroup.setAll('anchor.y', 0.5);
// Ende Zeile löschen

// ...
  startGame: function() {
    if(!this.bird.alive && !this.gameover) {
      this.bird.body.allowGravity = true;
      this.bird.alive = true;
      // add a timer
      this.pipeGenerator = this.game.time.events.loop(Phaser.Timer.SECOND * 1.25, this.generatePipes, this);
      this.pipeGenerator.timer.start();

// folgende Zeile auch löschen
      //this.instructionGroup.destroy();

    }
  },
//...
```

Ein paar Zeilen mehr werden benötigt, um die Start-, Stop- und Pause-Events im “*play.js*” zu beschreiben:

```
// play.js
// ...
  create: {
  // ...
    // add keyboard controls
    this.flapKey = this.game.input.keyboard.addKey(Phaser.Keyboard.SPACEBAR);
  // folgende Zeilen löschen:
  // this.flapKey.onDown.addOnce(this.startGame, this); // löschen
  // auf den Event "started" hören, um das Spiel zu starten
  var that = this;
    this._oAPI.on('startChanged', function(shouldStart){
      if (shouldStart === true) {
        that.game.state.start('play');
      }
    });

    this.flapKey.onDown.add(this.bird.flap, this.bird);
    // add mouse/touch controls
  // diese zeile kann gelöscht werden
  //this.game.input.onDown.addOnce(this.startGame, this); // löschen

    this.game.input.onDown.add(this.bird.flap, this.bird);
  // ...
  // neue Methode aurufen
  this.addListeners();
  },
// Neue Methode hinzufügen, um auf die API zu hören
  addListeners: function(){
    var that = this;
  if (this._oAPI.stopped() === true) {
      that.game.state.start('menu');
    }
    if (this._oAPI.started() === true) {
```

```
      that.startGame();
    }
    if (this._oAPI.paused() === true) {
      //TODO: add a nice pause sprite
      that.game.paused = true;
    }
    this._oAPI.on('startChanged', function(shouldStart){
     if (shouldStart === true) {
        that.startGame();
      }
    });
    this._oAPI.on('stopChanged', function(shouldStop){
      if (shouldStop === true) {
        that.game.state.start('menu');
      }
    });
        this._oAPI.on('pauseChanged', function(shouldPause){
      that.game.paused = shouldPause === true;
    });
    this._oAPI.on('startChanged', function(shouldStart){
      that.startGame();
    })
  },

//...
```

1.3 Produktion und Fehler registrieren

Nun bleibt nur noch der Pipe-Generator und der Death-Handler zum Anpassen. Der Death-Handler ist eine Zeile zusätzlich:

```
// play.json
//...
  deathHandler: function(bird, enemy) {
// fire and forget
    this._oAPI.queues.putItem('production-error').done();
//...
```

Aufwändiger ist es die Logik für eine neue Pipe zu implementieren, dafür kommt eine Zeil die Methode *startGame* und die *generatePipes* Methode bekommt die erweiterte Funktionalität dazu:

```
// play.json
//...
  startGame: function() {
    if(!this.bird.alive && !this.gameover) {
      this.bird.body.allowGravity = true;
      this.bird.alive = true;
      // add a timer
      this.pipeGenerator = this.game.time.events.loop(Phaser.Timer.SECOND * 1.25, this.generatePipes, this);
      this.pipeGenerator.timer.start();
     // folgende Zeilen kommt dazu:
      this.pipeExists = false;
    }
  },
// ...
   generatePipes: function() {
    var that = this;
    if(this.pipeExists === false) {
    // nur eine pipe per Bild
      this.pipeExists = true;
      // neues pattern holen, falls eines existiert, sonst warten, bis es wieder eines gibt
      this._oAPI.queues.getNext()
        // am Ende, wenn das holen veruscht wurde
        .fin(function(){})
        .then(
      // im Erfolgsfall pipe generieren
```

```
          function(){
          that._oAPI.queues.getNextPattern()
            .then(function(patternColors) {
              var pipeY = that.game.rnd.integerInRange(-100, 100);
              var pipeGroup = that.pipes.getFirstExists(false);
              if(!pipeGroup) {
                pipeGroup = new PipeGroup(that.game, that.pipes);
              }
              pipeGroup.reset(that.game.width, pipeY);
            });
        },
    // Im Fehlerfall nochmals versuchen
        function(){
          // try again later
          that.pipeExists = false;
        });
    }
  }
//...
```

8.4 Ausblick

Das Framework kann im jetztigen Stadium bereits genutzt werden, und einer Integration von neuen Spielen steht nichts im Weg. Dennoch gibt es einige Aufgaben, welche als nächstes in Angriff genommen werden sollten, bevor der Prototyp des Frameworks ausgereift ist.

Vom Entwicklerteam werden als nächstes folgende Schritte vorgeschlagen, um den Prototypen näher zur Produktreife zu bringen:

- Automatisierte Tests der JavaScript-API, der Mini-Games und des Backends
  - Eventuell Testsuite für neue Games, für externe Gameentwicklung
- Trennung vom spezifischen Game und der allgemeinen Nutzung
- Erweiterung der zur Verfügung gestellten Dashboard-Elemente und Auswertungsverfeinerungen
- Admin Interface schöner gestalten und Usability-Tests durchführen
- Websockets mit einer Library wie zB Socket.io nutzen und JavaScript-API verfeinern, so dass push Events vom Server empfangen werden können

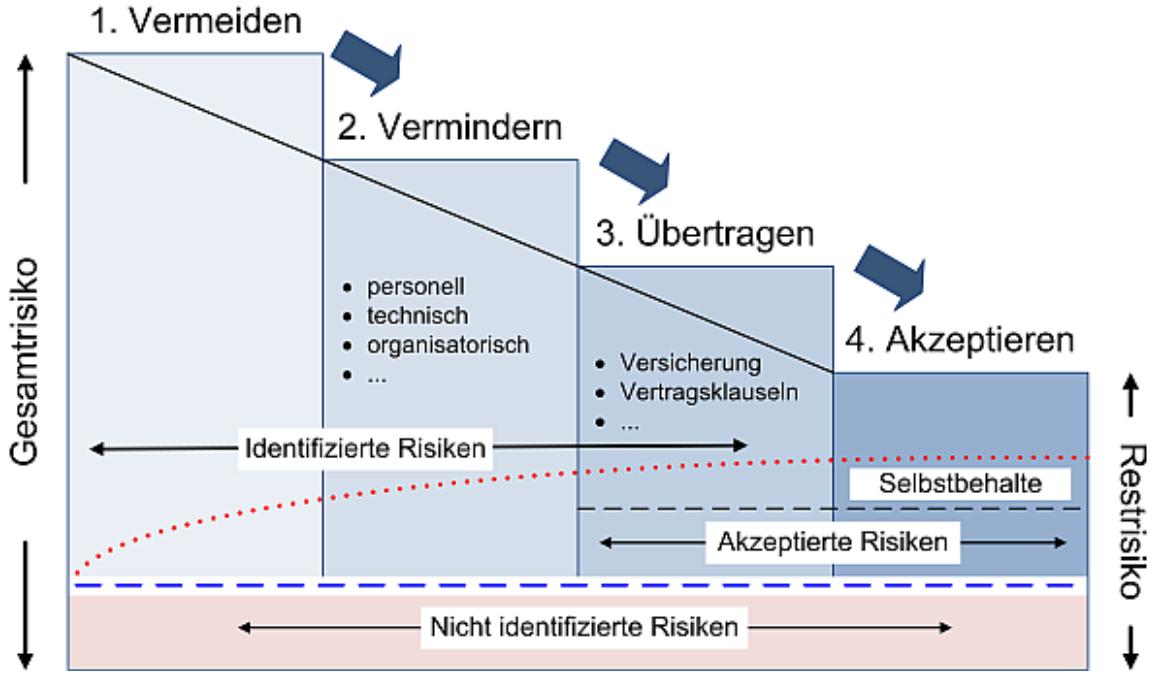
Kapitel 9

Risiken

9.1 Strategie zur Minimierung der Risiken und Schäden

Die Behandlungsstrategie für anfallende Risiken wird in der Abbildung 9.1 *Behandlungsstrategie* erläutert.

Abbildung 9.1: Behandlungsstrategie aus [3]



1 Risiken sollen weitmöglichst im Vornherein ausgeschlossen oder deren Eintrittswahrscheinlichkeit vermindert werden können.

2 Können Risiken nicht vermieden werden, so sollen Auswirkungen kompensiert oder minimiert werden.

3 Das Risiko soll zusätzlich an Dritte übertragen werden (typischerweise durch Versicherungen oder spezielle Verträge). Da dies im Rahmens einer Bachelorarbeit nur bedingt möglich ist (zum Beispiel durch eine Aufgabenstellung), musste in diesem Projekt auf diesen Schritt grösstenteils verzichtet werden.

4 Die übrig gebliebenen Risiken verbleiben im Projekt. Diese müssen dokumentiert und während des gesamten Projektverlaufs überwacht werden.

9.1.1 Risikoprozess nach Peter Johann

In der Abbildung 9.2 *Risikoprozess* wird der Risikoprozess nach Peter Johann[3] gezeigt.

Die Schritte sind im Folgenden genauer erläutert:

0 In diesem ersten Schritt wird festgelegt, wie das Risikomanagement durchgeführt werden soll. Dies beinhaltet auch das Definieren von Methoden, Dokumenten und Tools, die während des Projektverlaufs zum Einsatz kommen sollen.

1 Während der Risiko-Identifikation werden mögliche Risiken identifiziert und als solche in einer Auflistung oder Tabelle festgehalten.

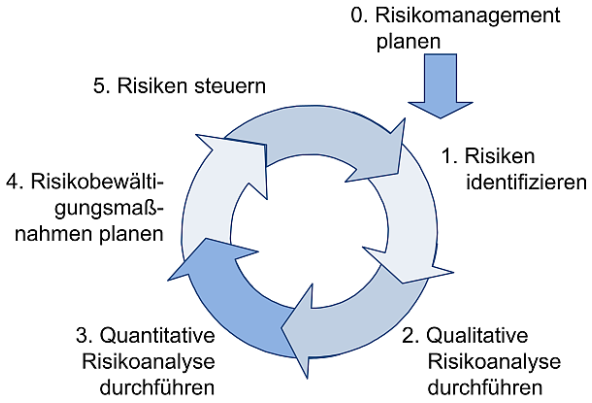
2 Anschliessend werden die Auswirkungen der erkannten Risiken auf das Projekt untersucht, wobei während der qualitativen Risikoanalyse nur Auswirkungen, welche keine Geldwerte darstellen, betrachtet werden.

3 Als Ergänzung der qualitativen Risikoanalyse werden die Auswirkungen der Risiken monetär durchgerechnet. Da im Rahmen dieser Bachelorarbeit das Zeitbudget und folglich auch die monetären Ressourcen fix vorgegeben sind, erübrigt sich eine quantitative Risikoanalyse.

4 In diesem Schritt wird eine Art “Massnahmenplan” erstellt, wie mit dem eventuellen Eintreffen eines jeden identifizierten Risikos umgegangen werden soll.

5 Der in Schritt 4 erstellte Massnahmenplan wird im Laufe des Projekts dauernd angepasst und ergänzt. Das Reporting der Risiken und das gezielte Eingreifen im Bedarfsfall gehört ebenfalls zu diesem Prozessschritt.

Abbildung 9.2: Risikoprozess



9.2 Analyse

Ein Risiko besteht aus dem zu erwartenden Schadensausmass in Abhängigkeit von der Eintrittswahrscheindlichkeit. Das Schadensausmass wird in 6 verschiedene Wertebereiche eingeteilt und entsprechend das erwartete Ausmass (in Anzahl verlorener Tage) sowie die erwartete Auswirkung auf das Projekt dokumentiert. Die Einteilung kann aus der Tabelle 9.1 *Skala Schadensausmass* entnommen werden.

Tabelle 9.1: Skala Schadensausmass

Wert	Ausmass	Auswirkung auf Projekt
1	Sehr geringer Schaden, maximaler Zeitverlust 1 Tag	Schaden unbedeutend
2	Geringer Schaden, maximaler Zeitverlust 2 Tage	Schaden gering
3	Mittlerer Schaden, maximaler Zeitverlust 5 Tage	Schaden spürbar
4	Grosser Schaden, maximaler Zeitverlust 12 Tage	Schaden nur mit enormem zusätzlichen Aufwand tragbar
5	Sehr grosser Schaden, maximaler Zeitverlust 3 Wochen	Schaden kritisch für Projekt
6	Grösster anzunehmender Schaden, Zeitverlust liegt über 3 Wochen	Projektabbruch

Tag Ein Tag entspricht einem Arbeitstag auf das gesamte Team bezogen (120% Arbeitswoche)

In einer Tabelle werden die möglichen Risiken und deren Eintrittswahrscheinlichkeit und Schaden aufgelistet und während des Projektverlaufs immer wieder nachgeführt und ergänzt. Die im Verlauf der Arbeit identifizierten Risiken können aus der Tabelle 9.2 *Identifizierte Risiken* entnommen werden. Zusätzlich zu den identifizierten Risiken sind in der Tabelle 9.3 *Nicht Identifizierte Risiken* die ungeplanten, nicht vorhergesehenen aber eingetroffenen Risiken aufgelistet.

Tabelle 9.2: Identifizierte Risiken

Risiko-Nr.	Bezeichnung	EW	Schaden
1	Featurezeitschätzung wird um mehr als 10% überschritten	20%	3
2	System kann vorgegebene Last (Anzahl Mitteilungen) nicht bewältigen	40%	4
3	Performance-Bottlenecks nicht korrekt identifiziert	40%	4
4	WLAN-Latency erweist sich als zu gross	30%	3
5	Ein Teammitglied fällt aus (zB Krankheit)	100%	2
6	Konkretisierung der Aufgabenstellung komplexer als erwartet	100%	3
7	Zeitmarge für Kundenwunschmanagement zu klein	100%	2
8	Umfangreiche Risikoabdeckung verlängert die Vorstudienarbeit	100%	3

EW Eintrittswahrscheinlichkeit

Tabelle 9.3: Nicht Identifizierte Risiken

Risiko-Nr.	Bezeichnung	EW	Schaden
9	Zu niedrig geschätzter Aufwand für parallele Backend- und Frontendentwicklung, Hardwarebeschaffung und Inbetriebnahme	100%	2
10	Sich widersprechende Anforderungen an den Inhalt der Dokumentation	100%	3
11	Externer Firmenpartner kann die Bachelorarbeit nicht mehr betreuen	100%	3

EW Eintrittswahrscheinlichkeit

9.2.1 Massnahmen

Anhand der Auflistung der identifizierten Risiken werden Massnahmen geplant, um die Eintrittswahrscheinlichkeit für ein Risiko zu senken oder das erwartete Schadensausmass zu minimieren. In der Tabelle 9.4 *Massnahmen* findet sich eine Aufstellung

der ergriffenen Massnahmen, um Eintrittswahrscheinlichkeit und Schadensausmass der zuvor identifizierten Risiken zu minimieren.

Tabelle 9.4: Massnahmen

Risiko-Nr.	Massnahme
1	Zeitschätzung wird fortlaufend überprüft und falls notwendig angepasst
2	Frühzeitig entsprechende Loadtests durchführen, falls notwendig alternative technische Lösungen evaluieren.
3	Architektur-Prototyp über alle Schichten und Layers erstellen.
4	Einführung und Verwendung vom HTML 5 Localstorage und Projektumfang entsprechend verkleinern.
5	Durch tägliche Standup-Meetinges weiss jedes Mitglied immer, wo das Projekt aktuell steht. Es wird ein Zeitpuffer eingeplant. Falls notwendig, muss eine Reduktion der geplanten Features in Betracht gezogen werden.
6	Aufgabenstellung mit den Vorstellungen und Wünschen des Kunden und der Dozenten immer wieder abgleichen und wenn nötig entsprechend anpassen.
7	Projektumfang verkleinern und zeitlich nicht mehr umsetzbare Change-Requests als optionale Features dokumentieren.
8	Projektumfang verkleinern.
9	Pufferzeit einberechnen und Projektumfang verkleinern, Hardwarebeschaffung und Inbetriebnahme aus dem Projektscope entfernen und nur die entsprechende Dokumentation erstellen.
10	Zusätzliche Pufferzeit berechnen oder Projektumfang verkleinern.
11	Neues Projekt (Aufgabenstellung) mit neuem Partner finden.

9.2.2 Bewältigung und Auswirkungen der eingetroffenen Risiken

Risiko 5

Beide Teammitglieder waren jeweils um eine Woche versetzt eine Woche lang krank. Durch Kommunikation via Slack war das kranke Mitglied jeweils zwar auf dem neuesten Projekt-Stand, konnte jedoch aufgrund der Krankheit nicht am Projekt weiterarbeiten. Da bereits für das Finden eines neuen Projekts und Industriepartners die

Pufferzeit grossenteils aufgebraucht war, blieb für das Risiko des Krankheitsfalls nicht mehr allzu viel übrig und das Projekt geriet in Verzug.

Risiko 6

Ein Teil der Aufgabenstellung beinhaltete das Konkretisieren der Aufgabenstellung gemeinsam mit den Betreuern und dem Kunden. Für diese Aufgabe wurde keine Pufferzeit eingeplant, da sie mit zur Aufgabenstellung gehört und deshalb in der anfangs zur Verfügung gestellten Zeit erledigt werden soll. Die Zeit, welche für das Konkretisieren gebraucht wurde, fehlte anschliessend in der Entwicklungszeit.

Risiko 7

Das Kundenwunschmanagement beinhaltet Sitzungen mit dem Kunden, Besprechen von Wünschen bezüglich der Features der Softwarelösung, etc. Für diese Tätigkeiten wurde zu wenig Zeit eingeplant. Die zusätzlich benötigte Zeit ging zu Lasten der Entwicklungszeit. Der Projektumfang wurde auf ein Minimum reduziert und alle Features, auf welche der Kunde verzichten konnte, gestrichen. Da von Anfang an klar war, dass das Zeitbudget für diesen Projektumfang eher eng war, wurde schon zu Beginn mit einer Minimalen Version geplant, so dass nicht mehr allzu viele Features gestrichen werden konnten.

Risiko 8

Um alle anfangs identifizierten Risiken - vorallem bezüglich des Netzwerks - abzudecken, wurden zwei umfangreiche Architektur-Prototypen erstellt (einer für den Frontend-Bereich und einer für den Backend-Bereich). Dies verlängerte die Vorstudienarbeit erheblich, so dass entschieden wurde, Code der Prototypen in das endgültige Projekt zu übernehmen, da der Projektumfang nicht mehr verkleinert werden konnte.

Risiko 9

Da der voraussichtliche Liefertermin der Hardware auf ein Datum nach der offiziellen Abgabe der Bachelorarbeit fällt, wurde in Absprache mit dem Kunden die Inbetriebnahme der Hardware aus dem Projektscope entfernt.

Risiko 10

Da von den Betreuern nicht abgestimmte Anforderungen an die Dokumentation und Arbeitsweise gestellt wurden, war lange nicht ganz klar, was letztendlich in die Dokumentation muss. Dieses Risiko wurde erst gar nicht erkannt und anschliessend unterschätzt. So wurde anfangs vieles im Doppel umgesetzt anstatt Klarheit zu schaffen, welche Anweisung gilt. Die dort verlorengegangene Zeit konnte nicht mehr durch Reduzierung des Projektumfangs kompensiert werden, da dieser bereits minimal war.

Risiko 11

Der Industriepartner für die ursprüngliche Arbeit erklärte in der ersten Arbeitswoche des Projekts, dass im Moment keine Ressourcen vorhanden seien, um eine Bachelorarbeit zu betreuen. In gegenseitigem Einverständnis wurde darauf die Zusammenarbeit aufgelöst. Es musste ein neuer Industriepartner und ein neues Projekt gesucht werden. Diese Aufgabe verbrauchte fast die gesamte Pufferzeit des Projekts (konkret ca 1/7 der Gesamtzeit). Anschliessend musste eine Aufgabenstellung erarbeitet und ein Projektumfang definiert werden, was nochmals andersweitig verplante Zeit in Anspruch nahm.

9.2.3 Conclusion

Insgesamt sind wesentlich mehr Risiken eingetroffen als in einem Software-Projekt dieser Grösse üblich ist. Im Normalfall reicht eine Pufferzeit von 20% um die Risiken ab zu fangen. Beinahe die gesamte Pufferzeit des Projekts wurde in den Projektwechsel in der ersten Woche investiert. Das neue Projekt bot sich trotz des umfangreichen Scopes als Alternativ-Projekt an, da der Kunde an der HSR selbst ist und somit Wegzeiten für Meetings wegfallen und das Projekt viel Lernpotenzial enthält. Anschliessend kostete die Konkretisierung der Aufgabenstellung und das Definieren eines Projektumfangs zusätzliche Zeit, welche nicht für diese Tätigkeiten eingeplant war. Der Projektumfang wurde bewusst klein gewählt, da sich alle Beteiligten der knappen übrig gebliebenen Zeit bewusst waren. Nachdem beide Teammitglieder kurz nacheinander für jeweils eine Woche krank waren, hat das Projekt ein minimales Feature-Set erreicht. Die Akkumulation aller eingetroffenen Risiken verlangten nach der Entscheidung ob ein Projektabbruch oder das Leisten von Überzeit in Kauf genommen wird. Normalerweise bestünde in so einem Fall zusätzlich die Möglichkeit mehr Zeit zu budgetieren (mehr

Geld zu investieren), mehr Projektmitarbeiter[1] einzustellen oder Teile des Projekts auszulagern. Dies ist bei dieser Bachelorarbeit mit Lieferung eines Softwareprodukts an einen Kunden nicht möglich.

[1]Dies macht das Projekt erst langsamer, da neue Mitarbeiter Ressourcen für Einarbeitung, Koordination und Administration benötigen. Diese Ressourcen fehlen während dieser Zeit im Projekt. Nach der Einarbeitungsphase gewinnt das Projekt durch die zusätzlich gewonnenen Ressourcen wieder an Geschwindigkeit. Wenn ein Projekt noch in der Anfangsphase ist, kann die anfängliche Verlangsamung durch die nachher gewonnene Geschwindigkeitssteigerung abgefangen werden und die Massnahme hat Erfolg.

Kapitel 10

Testing

10.1 Usability Tests

10.1.1 Paperprototyping

Während der Entwicklungsphase der ersten Prototypen wurden diverse Tests mit Paperprototypen für die einzelnen Views durchgeführt. Diese wurden je nach angestrebter Zielgruppe mit Studenten und dem zukünftigen Manager durchgeführt. Dabei wurde evaluiert, ob die Menu-Führung verständlich ist und ob die Anmeldung intuitiv durchführbar ist.

Es wurde festgestellt, dass sich Paperprototyping gut eignet, um beispielsweise zu überprüfen, ob die Menu-Führung oder Abläufe innerhalb einer Applikation für die Benutzer verständlich aufgebaut sind. Einer der grössten Vorteile von Paperprototyps ist, dass bereits festgestellt werden kann, ob die Benutzer sich im angedachten Design zurecht finden, bevor etwas implementiert wird.

Um Fragen bezüglich der Themen Touch-Feeling, Flow, Spass oder Design zu klären, eignen sich User-Tests besser, da in einem Paperprototyp zum Beispiel nicht evaluiert werden kann, ob die verwendeten Farben gut zu unterscheiden sind.

Die Ergebnisse der Paperprototyp-Tests und die daraus resultierenden Redesigns und Entscheidungen sind in der Vorstudie unter dem Punkt “Paperprototyping” genauer beschrieben.

10.1.2 User-Tests

Mittels den User-Tests wurde überprüft ob das Touch-Feeling ansprechend ist, die Test-Personen in einen sogenannten Flow gelangen, grundsätzlich Spass an der gestell-

ten Aufgabe haben und das Design ansprechend gestaltet ist. Dafür wurden während der Implementation wiederholt Test-Personen gebeten das Spiel zu spielen und Rückmeldung zu verschiedenen Themen zu geben. Die Tests und deren Ergebnisse finden sich in der Tabelle 10.1 zusammengefasst und anschliessend genauer erläutert.

Da das Spiel intuitiv gespielt werden können soll, wurden nicht allzu viele Tests im Voraus vorbereitet, viel mehr sollten die Test-Benutzer einfach spielen und dabei laut vorsagen was sie als nächstes machen wollen und wie sie gedenken, dies zu bewerkstelligen. Anhand dieser Rückmeldungen wurden Probleme im Userinterface-Design aufgedeckt, die anfangs gar nicht als Probleme erkannt wurden. Die darauf erkannten Probleme und deren Behebung findet sich in der Tabelle 10.2 zusammengefasst.

Tabelle 10.1: User-Tests

Test-Nr.	Was wurde getestet?	Datum	Resultat/Erkenntnis
1	Drag-Feeling	23.09.2014	Dragging fühlt sich ausreichend natürlich an, flicking fehlt jedoch
2	Drag&Drop-Feeling	23.09.2014	Drag&Drop fühlt sich natürlich an
3	Button-Press-Feeling	18.11.2014	visuelle Rückmeldung an User fehlte
4	visuelle Rückmeldung an User, wenn Button gedrückt wurde	02.12.2014	Animation musste etwas verkürzt werden

Test-Nr. 1

Da Dragging eine der wichtigsten Gesten in dieser Applikation ist, wurde durch Testbenutzer verifiziert, dass sich das Draggen von Objekten gut und vor allem gewohnt anfühlt. Dabei wurde Wert darauf gelegt, dass das Objekt nicht unter dem Finger wegrutscht und bei schnellen Bewegungen sich nicht zu weit vom Finger entfernt. Leider wird die Flick-Geste, mit welcher ein Objekt angestossen werden kann, nicht unterstützt. Nach kurzer Angewöhnungszeit fällt dies jedoch nicht mehr auf.

Test-Nr. 2

Drag&Drop wird in der Produktions-View für das Produzieren von Produkten und das Einsortieren von Materialien in die Zwischenlager verwendet. Bei den Tests wurden

die Test-Benutzer aufgefordert, das Verhalten des Objekts, welches sie fallen lassen, zu bewerten. Die Tatsache, dass die meisten Test-Benutzer nicht sagen konnten, wie es sich denn falsch anfühlen könnte, zeigt, dass Drag&Drop - welches standardmässig von Phaser zur Verfügung gestellt wird - sich natürlich und gewohnt anfühlt.

Test-Nr. 3

Da in der Abnahme-View aufgrund der mangelnden Anpassbarkeit keine Phaser-eigene Buttons verwendet wurden, musste das Button-Press-Feeling, welches selber implementiert wurde, auf seine Natürlichkeit geprüft werden. In einem ersten Test wurde keine visuelle Rückmeldung des Buttons implementiert. Die Test-User waren nach dem Klicken des Buttons jeweils unsicher, ob das Spiel nun verstanden hat, dass sie ein Produkt ordern wollten und ob die Bestellung abgeschickt wurde. Die Idee einer kleinen Messagebox oder in der View integrierten Nachricht wurde schnell verworfen, da es zu viel Platz eingenommen hätte. So wurde entschieden, den Button kurz sein Alpha-Wert ändern zu lassen um dem Benutzer zu signalisieren, dass der Klick registriert wurde.

Test-Nr. 4

Die aufgrund Test-Nr. 3 implementierte visuelle Rückmeldung ist anschliessend wieder von Test-Benutzern bewertet worden. Die Test-Benutzer bemerkten, dass die Animations-Dauer zu lange gewählt war und für einen angenehmen Spielfluss verkürzt werden sollte. Diese wurde daraufhin ohne nochmaligen User-Test verkürzt.

Tabelle 10.2: Usability Probleme

Problem-Nr.	Problem	Datum	Behebung
1	die Farben Grau und Violett konnten im Produkt nur ungenügend unterschieden werden	04.11.2014	die Alpha-Werte wurde verändert
2	User verstand nicht, warum kein Material mehr aus dem (leeren) Zwischenlager kam	18.11.2014	jedem Zwischenlager wurde ein Counter mit der aktuellen Anzahl Materialien hinzugefügt
3	User verstand nicht, warum kein Material mehr ins (volle) Zwischenlager verschoben werden kann	18.11.2014	eine Animation wurde hinzugefügt, falls ein Spieler Materialien in ein bereits volles Zwischenlager verschieben will
4	Dauer zwischen Loslassen des Materials (Drag&Drop in Zwischenlager) und Verschwinden des Materials je nach Netzwerk-Verbindung zu gross	02.12.2014	"Warte-" Animation wurde eingeführt, welche dem User anzeigt, dass etwas am geschehen ist und er einfach ein wenig Geduld haben muss

Problem 1

Die aufgehellten Farben Grau und Violett welche in das Produkt-Pattern eingebaut sind, konnten von den Test-Benutzern nur ungenügend voneinander unterschieden werden. Um dieses Problem zu beheben wurden die jeweiligen Alpha-Werte verändert, so dass die Farben weniger blass und deshalb wieder einfacher zu unterscheiden sind.

Problem 2

Ist das Zwischenlager leer, können keine Materialien mehr daraus geholt werden. Die Test-Benutzer verstanden nicht, dass das Zwischenlager leer ist und wunderten sich daher, dass keine Materialien mehr bezogen werden können. Um dem User mitzuteilen, wie viele Materialien sich aktuell gerade im Zwischenlager befinden, wurde jedem Zwischenlager ein entsprechender Counter hinzugefügt.

Problem 3

Ist das Zwischenlager voll, können keine Materialien mehr hinein verschoben werden. Die Test-Benutzer verstanden nicht, dass das Zwischenlager voll ist und wunderten sich daher, dass keine Materialien mehr eingelagert werden können. Eine erste Idee war den Counter so zu erweitern, dass der Benutzer sah, wie viele Plätze der maximalen Anzahl Plätze bereits durch Materialien besetzt sind. Durch die zusätzliche Information wäre jedoch die Schrift zu klein geworden, so dass entschieden wurde, eine Animation hinzuzufügen, welche dem Spieler anzeigt, dass das Material nicht eingelagert werden kann.

Problem 4

Die Dauer zwischen dem Loslassen des Materials (Drag&Drop in Zwischenlager) und dem effektiven Verschwinden des Materials im Zwischenlager war je nach Lag in der Netzwerk-Verbindung zu gross. Um diese Wartezeit zu überbrücken, wurde eine “Warte-” Animation eingefügt, welche dem User anzeigt, dass das Spiel verstanden hat, dass er den Auftrag gegeben hat, das Material einzulagern.

10.2 Loadtests

Um zu überprüfen, ob die System- und Netzlast im tragbaren Bereich liegen, wurden zwei verschiedene Loadtest durchgeführt. Der erste Test erfolgte lokal. Mittels des Programms Siege wurden Requests generiert, welche der Server abarbeiten sollte. Der zweite Test wurde mit 28 Testpersonen durchgeführt, welche alle von ihren Geräten aus durch Klicken eines Buttons Requests für den Server generierten. Auf diese Weise konnte sichergestellt werden, dass das verteilte System die vorgegebene Anzahl paralleler Nutzer bewältigen kann.

10.2.1 Local Loadtest

Ein erster Test sollte prüfen, ob die Systemlast tragbar ist.

Tabelle 10.3: Loadtests

	Elapsed Time	Data transferred	Response time	Transaction rate	Throughput	Concurrency
Run 1	9.86	1.01	0.05	1014.2	0.1	49.85
Run 2	9.81	1.01	0.05	1019.37	0.1	49.83
Run 3	9.81	1.01	0.05	1019.37	0.1	49.81
Run 4	9.81	1.01	0.05	1019.37	0.1	49.84
Run 5	9.81	1.01	0.05	1019.37	0.1	49.84
Run 6	9.81	1.01	0.05	1019.37	0.1	49.82
Run 7	9.82	1.01	0.05	1018.33	0.1	49.86
Run 8	9.82	1.01	0.05	1018.33	0.1	49.85
Run 9	9.88	1.01	0.05	1012.15	0.1	49.75
Run 10	9.8	1.01	0.05	1020.41	0.1	49.86
Ø	9.823	1.01	0.05	1018.027	0.1	49.831
Masseinheit	[s]	[MB]	[s]	[transactions/s]	[MB/s]	

Der lokale Loadtest wurde auf einem Laptop mit folgenden Spezifikationen ausgeführt:

- Prozessor: i7-4710HQ
- Festplattentyp: SSD
- Datenbank: Postgresql (gefüllt mit bereits über 100'000 Event-Entries)

Als Server wurde Flask verwendet. Mittels des Loadtest-Programms Siege wurden insgesamt 10'000 Requests an den Server geschickt, aufgeteilt auf 50 parallele simulierte Worker. Dies ergibt 200 Requests pro simuliertem Worker. In der Tabelle 10.3 wird ersichtlich, dass der Server die geforderte Parallelität (Concurrency) während

des Tests fast komplett aufrecht erhalten konnte. Um aussagekräftige Daten zu gewinnen, wurde der Test 10 Mal durchgeführt. Dabei zeigt die Tabelle 10.3, dass in jedem Durchgang für die Abarbeitung der 10'000 Requests ungefähr dieselbe Zeit benötigt wurde. Anhand der in jedem Durchgang gleich grossen übertragenen Datenmenge (Data transferred) kann abgeleitet werden, dass alle Requests erfolgreich bearbeitet wurden. Die Antwortzeit des Servers ist ebenfalls in jedem Testdurchgang identisch. Die Spalte Transaction-Rate zeigt dass durchschnittlich 1018 Transaktionen pro Sekunde abgearbeitet werden konnten.

Der Siege-Aufruf für die Ausführung des Tests wurde folgendermassen formuliert:

```
siege -d0 -r200 -c50 "localhost/api/v1/events/ POST event_type_id=1&name=siegetest&user=siege"
```

Dabei wurden die folgenden Parameter mitgegeben:

-d Verzögerung der Instantiierung der simulierten Benutzern: 0 (es ist keine Verzögerung erwünscht, da alle Benutzer auf einmal instantiiert werden sollen)

-r Anzahl Requests pro simuliertem User: 200

-c Anzahl parallelle simulierte User: 50

"localhost/api/v1/events/" URL, die zu testen ist

POST Request-Typ: Post-Request

event_type_id=1&name=siegetest&user=siege Payload

event_type Foreign-Key im Event

name Bezeichnung des Tests

user Benutzer-Id des Users

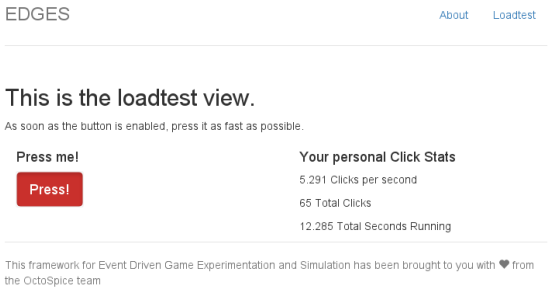
Für einen hohen Realitätsbezug wurden Requests mit kleiner Datenmenge, dafür in umso grösserer Anzahl verwendet.

Der vollständige Log des Tests findet sich im Anhang.

10.2.2 Netzlast-Test

In einem zweiten Test wurde überprüft, ob die Netzlast tragbar ist.

Abbildung 10.1: Klickload-Test Screenshot



Dafür wurde eine Website mit einem Button zur Verfügung gestellt. Durch Klicken auf den Button können Requests erzeugt werden, welche wiederum vom Server verarbeitet werden. Der Test wurde während einer Unterrichtslektion mit 28 Studenten durchgeführt. Die Studenten bekamen die Anweisung, zur Website zu navigieren und auf das mündliche Startkommando so schnell und so oft wie möglich auf den Button zu klicken.

Abbildung 10.2: Klickload-Test mit Studenten

10.3 Browser Tests

Um zu überprüfen, in welchen Browsern das Spiel gespielt werden kann, werden Tests mit verschiedenen Browsern durchgeführt. Das Hauptaugenmerk liegt dabei auf der Spielbarkeit der verschiedenen Spiel-Views. Dabei wurden folgende Testkriterien überprüft:

- wird der Inhalt aufgerufenen Seiten korrekt dargestellt?
- können die Buttons der Abnahme-View und der Einkaufs-View geklickt werden?

Tabelle 10.4: Browser-Test

Browser	Gerät/OS	Version	Test-Datum	Ergebnis Spiel	Ergebnis Dashboard
Firefox	Windows 7	34.0	16.12.2014	OK	Nicht OK
Chrome	Windows 7	39.0. 2171.95	16.12.2014	OK	OK
Opera	Windows 7	26.0. 1656.32	16.12.2014	OK	OK
IE	Windows 7	11.0.9600. 17420	16.12.2014	OK	Nicht OK
Safari	OS X 10.9.5	7.1	16.12.2014	OK	OK
Safari Mobile	iOS 8.1		16.12.2014	OK	Nicht OK
Chrome (Mobile)	Android 5.0	39.0. 2171.93	16.12.2014	OK	OK
Chrome (Mobile)	Android 4.4.2	31.0. 1650.59	16.12.2014	OK	OK
Android Browser Internet	Android 4.4.2	1.5 .28	16.12.2014	OK	OK

- können Objekte in der Produktions-View ohne Ruckeln bewegt werden?
- ist die gesamte erwartete Funktionalität vorhanden?
- kann das Spiel auch nach dem Ändern der Fenstergrösse noch gespielt werden?

Ausserdem wird separat das Dashboard in den verschiedenen Browsern getestet. Dabei werden folgende Kriterien getestet:

- wird das Dashboard angezeigt?
- können im Dashboard Elemente ein- und ausgeblendet werden?

Firefox

Testumgebung Firefox Version 34.0 unter Windows 7

Resultat Alle Spiel-Views des Spiel lassen sich einwandfrei verwenden. Das Dashboard lässt Firefox in manchen Fällen nach einer gewissen Zeit einfrieren.

Chrome

Testumgebung Chrome Version 39.0.2171.95 unter Windows 7

Resultat Alle Spiel-Views lassen sich einwandfrei verwenden. Auch das Dashboard bereitet Chrome keine Schwierigkeiten und lässt sich sehr angenehm benutzen.

Opera

Testumgebung Opera Version 26.0.1656.32 unter Windows 7

Resultat Alle Spiel-Views lassen sich einwandfrei verwenden. Auch das Dashboard bereitet Opera keine Schwierigkeiten und lässt sich sehr angenehm benutzen.

Internet Explorer

Testumgebung IE Version 26.0.1656.32 unter Windows 7

Resultat Alle Spiel-Views lassen sich einwandfrei verwenden. Das Dashboard wird jedoch nicht dargestellt, zu sehen ist nur eine weisse Fläche.

Safari

Testumgebung Safari Version 7.1 unter OS X 10.9.5

Resultat Alle Spiel-Views lassen sich einwandfrei verwenden. Das Dashboard wird nicht korrekt angezeigt und kann nicht verwendet werden.

Safari (Mobile)

Testumgebung Safari Mobile unter iOS 8.1

Resultat Alle Spiel-Views lassen sich einwandfrei verwenden. Auch das Dashboard bereitet Safari keine Schwierigkeiten und lässt sich sehr angenehm benutzen.

Chrome (Mobile)

Testumgebung Chrome Version 31.0.1650.59 laufend unter Android 4.4.2

Resultat Alle Spiel-Views lassen sich einwandfrei verwenden. Auch das Dashboard bereitet Chrome keine Schwierigkeiten und lässt sich sehr angenehm benutzen.

Android Browser Internet (Mobile)

Testumgebung integrierter Android Browser 1.5.28 laufend unter Android 4.4.2

Resultat Alle Spiel-Views lassen sich einwandfrei verwenden. Auch das Dashboard bereitet dem Android Browser keine Schwierigkeiten und lässt sich sehr angenehm benutzen.

Kapitel 11

Deployment

11.1 Voraussetzungen

Als Betriebsystem für den Server wird Linux eingesetzt. Das Setup wurde unter Ubuntu 14.04 getestet. Um sicher zu stellen, dass auch andere Linux Distributionen und Versionen betrieben werden können, gibt es im Abschnitt Manuelles Deployment eine detailliertere Beschreibung, wie die Umgebung aufgesetzt werden kann und was die benötigten Komponenten sind.

Grundvoraussetzungen sind:

- Linux
- Internet Zugang
- Zugriff auf die Octospice Repositories
- Zugriff auf den Remote Server mit Sudo-Rechten
- Public Key vom Server auf den deployt werden soll ist auf den benötigten Repositories eingetragen
- Lokal ausgechecktes octocube-backend mit installiertem fabric[1]
- Python 2.7+ (nicht mit Python 3 kompatibel)

[1] Siehe auch Lokale Entwicklung

Abhängigkeiten

Die Abhängigkeiten zu System-Libraries des Linux Systems werden beim automatisierten Deployment nach Möglichkeit gleich installiert. Bei der lokalen Installtion ist dies meistens nicht erwünscht. Für diesen Fall, oder wenn es nicht erwünscht ist oder fehl schlägt, sind die Abhängigkeiten hier separat aufgelistet:

- git
- redis-server
- redis-tools
- python-dev (Development-Header von Python)
- python-virtualenv (Virtualenv Umgebung für Python
- libevent-dev (Development-Header der (lib)event Library)
- postgresql
- postgresql-server-dev-all (Postgres Development-Header)
- build-essential
- supervisor
- nginx-full

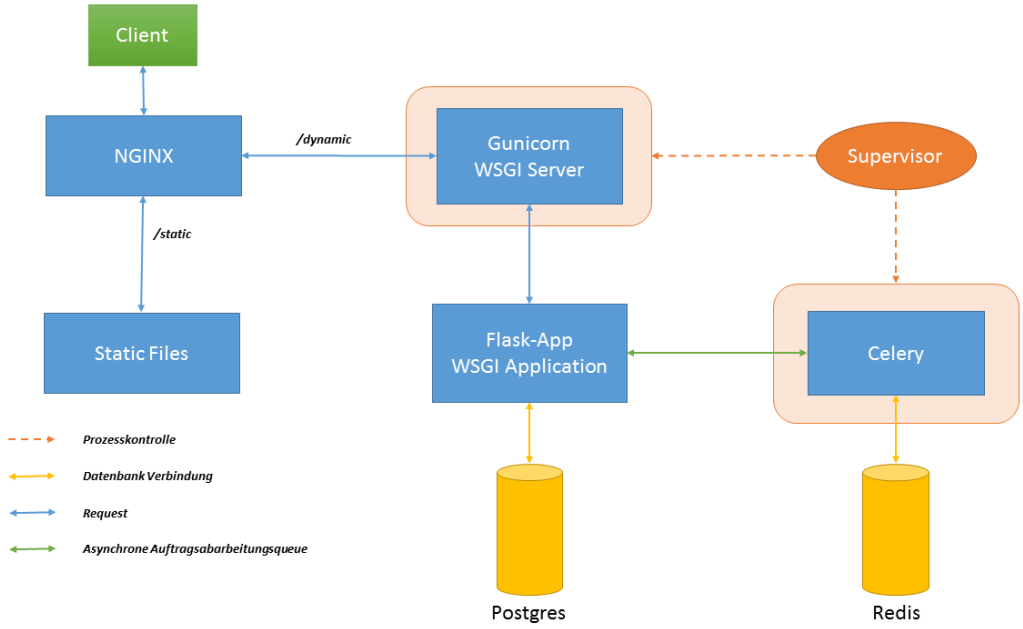
11.2 Automatisiertes Deployment

Für das automatisierte Deployment wird Fabric verwendet. Dies ist, etwas verallgemeinert gesagt, ein in Python geschriebenes, Skript-Basiertes Tool, um Remote (und lokal) repetitive Task zu automatisieren. Fabric selbst hat eine sehr gute Dokumentation, welche unter http://docs.fabfile.org/en/ gefunden werden kann.

Um das automatisierte Deployment nutzen zu können, muss die Datei ___init___.py im Verzeichnis “fabfile” des “octocube-backend” angepasst werden. Die Ausführungen in der Datei sind selbsterklärend. Wenn der <server-ip-oder-name> dort ergänzt wurde, können folgende Befehle verwendet werden:

- fab -H <server-ip-oder-name> bootstrap

Abbildung 11.1: Übersicht Backend-Zusammenspiel



  - Setzt den Server auf: Konfiguriert die Datenbank, Installiert alle Abhängigkeiten, holt alle angegebenen Repositories, erzeugt die benötigten Konfigurationsdateien, startet die benötigten Services, kreiiert einen Admin-User

- fab -H <server-ip-oder-name> deploy

  - updated alle git repos und started die Services neu

- fab -H <server-ip-oder-name> create_superuser

  - Erstellt einen neuen Admin Benutzer

11.3 Manuelles Deployment

Ein manuelles Deployment ist mehr Arbeit, grundsätzlich ist es sehr ähnlich zum automatisierten Prozess, nur nicht automatisch.

Zuerst müssen alle unter “Abhängigkeiten” aufgelisteten Libraries installiert werden. Anschliessend muss eine Datenbank erstellt werden, hier wird auf das Manual von Postgres selber verwiesen.

Der Checkout aller benötigten Repositories in ein Verzeichnis der Wahl folgt als nächster Schritt. Im ausgecheckten octocube-backend ordner muss eine Datei ".env" erstellt werden. Die sieht etwa so aus:

```
# .env
DATABASE_URL='postgres://<db_nutzer>:<db_passwort>@<db_host>:<db_port>/<db_name>'
CELERY_BROKER_URL='redis://localhost:6379'
CELERY_RESULT_BACKEND='redis://localhost:6379'
APP_SETTINGS='config.ProductionConfig'
```

Anschliessend muss eine Nginx Konfiguration angelegt werden, ein Beispiel für eine Mögliche Konfiguration findet sich unter fabfile/templates/nginx.conf.

Wenn diese angepasst ist, darf eine weitere Konfigurationsdatei angelegt werden, nämlich diejenige für den Prozessüberwacher "Supervisor". Auch für diese findet man ein Beispiel unter fabfile/templates/supervisor_template.conf.txt.

Wenn die Nginx Konfiguration aktiviert ist und der Supervisor die Prozesse starten konnte, steht einem erfolgreichen Start nichts mehr im Wege.

11.4 Lokale Entwicklung

Backend Entwicklung

Um eine lokale Entwicklungsumgebung aufzusetzen, braucht es eine Linux Umgebung mit den installierten Abhängigkeiten und einer Datenbank, und ein entsprechende .env Datei, wie beim manuellen Deployment. Dann müssen zwei Services gestartet werden (zwei Konsolenfenster öffnen), beide im Verzeichnis des checkouts des backends mit aktiviertem virtualenv:

- gunicorn -k flask_sockets.worker event_framework:app -b '0.0.0.0:8000'
- celery -A event_framework.celery worker -B -l debug

Der erste Befehl startet den lokalen Server und der zweite sorgt dafür, dass der Background Worker mitsamt Heartbeat läuft.

Game- oder API-Entwicklung

Für eine Game- oder API-Entwicklung wird auch ein Backend benötigt, jedoch muss dieses nur "laufen". Entsprechend kann zB eine VirtualBox mit Ubuntu aufgesetzt werden und das automatisierte Deployment dorthin gemacht werden, und bei der Entwicklung verwendet werden.

11.5 Ein neu entwickeltes Spiel einbinden

Sobald ein neues Spiel mit der API Verbunden ist kann es eingebunden werden.

Dies kann in zwei Schritten erreicht werden:

1. Repository in die Deployment-Datei eintragen
2. Deployen

Das neue Spiel wird in der Datei “static_deployments.py” im Verzeichnis “fabfile” bei den schon vorhandenen Spielen eingetragen. Die Ergänzung ist:

```
{
    'name': 'verzeichnis-name-und-url-name-ohne-leerzeichen',
    'title': u'Link Titel bei den Games und entsprechender Name, Leerzeichen sind erlaubt',
    'repository': 'Pfad zum entsprechenden git-Repository', # zB. git@bitbucket.org:octospice/octocube-flappybird-example.git
    'branch': 'master', # kann geändert werden, wenn gewünscht
    'path': 'dist' # pfad zu den "distributed" files, meistens automatisch generierte, minifizierte Version des Spiels
},
```

Dann kann ein Deploment mit “fab -H <server-oder-ip> deploy” erfolgen und das neue Spiel ist eingebunden.

Literaturverzeichnis

[1] Ralf Bruns and Jürgen Dunkel. *Event-Driven Architecture: Softwarearchitektur für ereignisgesteuerte Geschäftsprozesse*. Berlin, Heidelberg : Springer Berlin Heidelberg, 2010. ISBN 978-3-642-02439-9.

[2] Roy Thomas Fielding. *Architectural Styles and the Design of Network-based Software Architectures*. PhD thesis, University of California, 2000. AAI9980887.

[3] Peter Johann. Risikomanagement in Projekten, 09 2014. URL `http://www.peterjohann-consulting.de/index.php?menu-id=risk`.

Anhang A

Log

A.1 Raw Daten Siege Loadtest

```
test with 10000 connections
Wed Nov 7 10:26:03 CET 2014
** SIEGE 3.0.8
** Preparing 50 concurrent users for battle.
The server is now under siege...      done.

Transactions:              10000 hits
Availability:             100.00 %
Elapsed time:               9.86 secs
Data transferred:           1.01 MB
Response time:              0.05 secs
Transaction rate:        1014.20 trans/sec
Throughput:             0.10 MB/sec
Concurrency:               49.85
Successful transactions:       10000
Failed transactions:               0
Longest transaction:            0.14
Shortest transaction:           0.02


test with 10000 connections
Wed Nov 7 10:26:43 CET 2014
** SIEGE 3.0.8
** Preparing 50 concurrent users for battle.
The server is now under siege...      done.

Transactions:              10000 hits
Availability:             100.00 %
Elapsed time:               9.81 secs
Data transferred:           1.01 MB
Response time:              0.05 secs
Transaction rate:        1019.37 trans/sec
Throughput:             0.10 MB/sec
Concurrency:               49.83
Successful transactions:       10000
Failed transactions:               0
Longest transaction:            0.08
Shortest transaction:           0.01


test with 10000 connections
Wed Nov 7 10:27:23 CET 2014
** SIEGE 3.0.8
** Preparing 50 concurrent users for battle.
The server is now under siege...      done.

Transactions:              10000 hits
```

```
Availability:              100.00 %
Elapsed time:                9.81 secs
Data transferred:            1.01 MB
Response time:               0.05 secs
Transaction rate:         1019.37 trans/sec
Throughput:              0.10 MB/sec
Concurrency:                49.81
Successful transactions:       10000
Failed transactions:               0
Longest transaction:            0.08
Shortest transaction:           0.01


test with 10000 connections
Wed Nov 7 10:28:03 CET 2014
** SIEGE 3.0.8
** Preparing 50 concurrent users for battle.
The server is now under siege...      done.

Transactions:               10000 hits
Availability:              100.00 %
Elapsed time:                9.81 secs
Data transferred:            1.01 MB
Response time:               0.05 secs
Transaction rate:         1019.37 trans/sec
Throughput:              0.10 MB/sec
Concurrency:                49.84
Successful transactions:       10000
Failed transactions:               0
Longest transaction:            0.07
Shortest transaction:           0.01


test with 10000 connections
Wed Nov 7 10:28:43 CET 2014
** SIEGE 3.0.8
** Preparing 50 concurrent users for battle.
The server is now under siege...      done.

Transactions:               10000 hits
Availability:              100.00 %
Elapsed time:                9.81 secs
Data transferred:            1.01 MB
Response time:               0.05 secs
Transaction rate:         1019.37 trans/sec
Throughput:              0.10 MB/sec
Concurrency:                49.84
Successful transactions:       10000
Failed transactions:               0
Longest transaction:            0.08
Shortest transaction:           0.01


test with 10000 connections
Wed Nov 7 10:29:23 CET 2014
** SIEGE 3.0.8
** Preparing 50 concurrent users for battle.
The server is now under siege...      done.

Transactions:               10000 hits
Availability:              100.00 %
Elapsed time:                9.81 secs
Data transferred:            1.01 MB
Response time:               0.05 secs
Transaction rate:         1019.37 trans/sec
Throughput:              0.10 MB/sec
Concurrency:                49.82
Successful transactions:       10000
Failed transactions:               0
Longest transaction:            0.08
Shortest transaction:           0.01


test with 10000 connections
Wed Nov 7 10:30:02 CET 2014
```

```
** SIEGE 3.0.8
** Preparing 50 concurrent users for battle.
The server is now under siege...      done.

Transactions:              10000 hits
Availability:             100.00 %
Elapsed time:               9.82 secs
Data transferred:           1.01 MB
Response time:              0.05 secs
Transaction rate:        1018.33 trans/sec
Throughput:             0.10 MB/sec
Concurrency:               49.86
Successful transactions:       10000
Failed transactions:               0
Longest transaction:            0.08
Shortest transaction:           0.01


test with 10000 connections
Wed Nov 7 10:30:42 CET 2014
** SIEGE 3.0.8
** Preparing 50 concurrent users for battle.
The server is now under siege...      done.

Transactions:              10000 hits
Availability:             100.00 %
Elapsed time:               9.82 secs
Data transferred:           1.01 MB
Response time:              0.05 secs
Transaction rate:        1018.33 trans/sec
Throughput:             0.10 MB/sec
Concurrency:               49.85
Successful transactions:       10000
Failed transactions:               0
Longest transaction:            0.08
Shortest transaction:           0.00


test with 10000 connections
Wed Nov 7 10:31:22 CET 2014
** SIEGE 3.0.8
** Preparing 50 concurrent users for battle.
The server is now under siege...      done.

Transactions:              10000 hits
Availability:             100.00 %
Elapsed time:               9.88 secs
Data transferred:           1.01 MB
Response time:              0.05 secs
Transaction rate:        1012.15 trans/sec
Throughput:             0.10 MB/sec
Concurrency:               49.75
Successful transactions:       10000
Failed transactions:               0
Longest transaction:            0.09
Shortest transaction:           0.01

test with 10000 connections
Wed Nov 7 10:32:02 CET 2014
** SIEGE 3.0.8
** Preparing 50 concurrent users for battle.
The server is now under siege...      done.

Transactions:              10000 hits
Availability:             100.00 %
Elapsed time:               9.80 secs
Data transferred:           1.01 MB
Response time:              0.05 secs
Transaction rate:        1020.41 trans/sec
Throughput:             0.10 MB/sec
Concurrency:               49.86
Successful transactions:       10000
Failed transactions:               0
Longest transaction:            0.07
Shortest transaction:           0.01
```