HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

FHO Fachhochschule Ostschweiz

SCS
super computing systems

Spielerische Visualisierung von Optimierungsalgorithmen

Bachelorarbeit

Abteilung Informatik

Hochschule für Technik Rapperswil

Frühjahrssemester 2014

Autor(en):	Raphael Faes, Tobias Büchel
Betreuer:	Prof. Hansjörg Huser
Projektpartner:	SCS Zürich
Experte:	Stefan Zettel, Ascentiv Zürich
Gegenleser:	Prof. Dr. Luc Bläser

Danksagung

Wir danken folgenden Personen für Ihre Unterstützung während der Bachelorarbeit:

- Prof. Hansjörg Huser für die Betreuung unserer Bachelorarbeit
- Julian Heeb für die Betreuung unserer Bachelorarbeit
- Andreas Kägi für die Inputs bezogen auf die RESTful- Schnittstelle und Domain Analyse

Inhaltsverzeichnis

1 Aufgabenstellung

super computing systems

Aufgabenstellung Studienarbeit/Bachelorarbeit

„Spielerische Visualisierung von Optimierungsalgorithmen"

Die Planung und Optimierung von Produktionsabläufen und des Personaleinsatzes in den verschiedensten Industriezweigen und Dienstleistungsbranchen ist heute ohne den Einsatz raffinierter Algorithmen (=Lösungsstrategien) nicht mehr effizient durchführbar. Supercomputing Systems bietet unter anderem die Entwicklung solcher problemspezifischer Optimierungssoftware an. Um die Kommunikation mit Kunden (auch solchen mit einem anderen fachlichen Hintergrund) über dieses komplexe Thema zu erleichtern, wurde unter www.scs.ch/optimizer ein einfacher Prototyp einer Demonstrationssoftware (ein „Demonstrator") erstellt, der auf spielerische und graphisch ansprechende Weise die algorithmische Kompetenz von SCS im Bereich der Prozessoptimierung veranschaulicht. In diesen Arbeiten soll dieser Demonstrator weiterentwickelt und auf andere Technologien portiert werden.

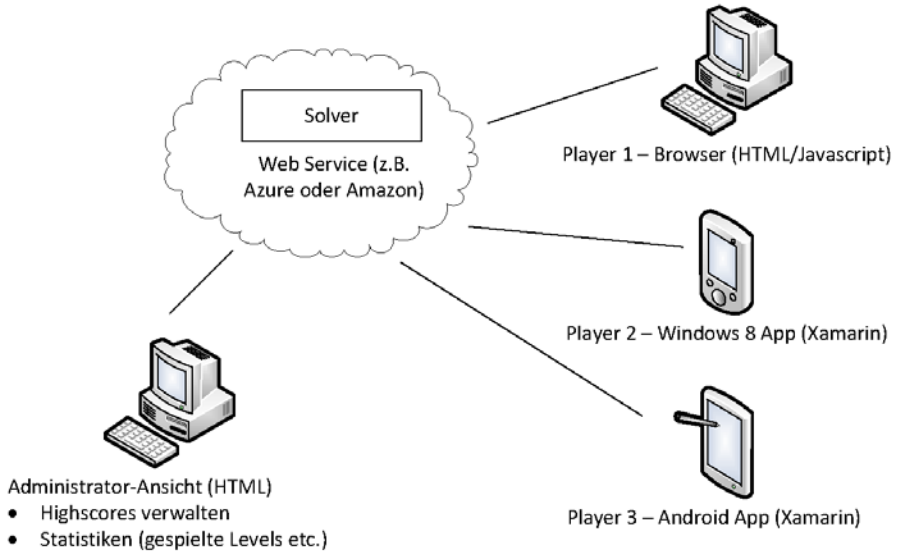
Liste der Teilaufgaben

	Priorität
Den (existierenden) Solver in die Cloud integrieren, so dass dieser beim Aufruf von „Solve" aufgerufen wird und die optimale Lösung an das Spiel zurückschickt (dieses zeigt dann die berechnete Lösung an) Dazu sind folgende Teilprobleme zu lösen: • Restful-Interface definieren, um Daten zwischen Endgeräten und Web-Service zu übertragen • Webservice aufsetzen, der das Restful-Interface implementiert, Solver in den Webservice integrieren	1
Mit Xamarin das Spiel auf Windows 8 und Android-Tablets portieren und testen	1
Weiterentwicklung des Spielkonzepts (Highscores, Countdown, mehr Flexibilität bei der Level-Generierung, Benutzer kann eigene Instanzen/Levels hochladen) *Optional:* Mehrspielermodus	1
Implementierung einer Administrator-Ansicht, welche die Highscore-Verwaltung und statistische Auswertungen erlaubt (z.B. welche Levels wurden wie häufig gespielt)	1
Harmonisierung der Eingabeschnittstellen des Browser-Spiels und des Solvers (allgemeinere Wechselkosten).	2

Weitere Anforderung:
Bei den oben beschriebenen Erweiterungen soll das Spiel bei fehlender Internetverbindung dennoch in beschränktem Umfang funktionieren (nur z.B. Highscores und Solver-Aufruf funktionieren dann nicht).

Supercomputing Systems AG • Technoparkstrasse 1 • CH-8005 Zürich • Phone +41 (0)43 456 16 00 • info@scs.ch • www.scs.ch

Das untenstehende Bild gibt einen Überblick über die angedachte Architektur des Systems.



2 Abstract

Der von der SCS entwickelte Solver bietet die Möglichkeit komplexe Produktionsabläufe und Personaleinsätze zu optimieren und somit Zeit und Geld zu sparen. Den Einsatz eines solchen Algorithmus zu veranschaulichen, speziell jemandem ohne technischen Hintergrund, ist eine schwierige und trockene Angelegenheit. Mittels Gamification kann das Potential des Solvers in spielender Form als Casual Game für jedermann demonstriert werden. Im Singleplayer-Modus beginnt der Spieler mit einfachen Aufgabenstellungen, welche von Runde zu Runde komplexer werden, und wird so langsam an die Problemdomäne herangeführt. Für die Langzeitmotivation wurde ein Multiplayer-Konzept ausgearbeitet und implementiert. In diesem Head-To-Head-Modus kann der Spieler sein erlerntes Know-How direkt gegen andere Spieler anwenden und sich weiter mit den Möglichkeiten des Algorithmus auseinander setzen.

Im zweiten Aspekt der Arbeit geht es um den Erfahrungsbericht gewisser Technologien und wie sich diese bewähren. Das Mobile App dient so ebenfalls als Demonstration von aktuellen Technologien und Konzepten. Um eine möglichst hohe Plattformunabhängigkeit für die unterschiedlichen mobilen Betriebssysteme zu erreichen wurde Xamarin für den Shared Code eingesetzt. Zum Schutz des Algorithmus wurde dieser in der Microsoft Azure Cloud abgekapselt und muss so nicht mit dem App ausgeliefert werden. Die Portabilität, welche sich von Xamarin in Kombination mit MvvmCross ergibt, stellt sich als sehr mächtig dar. Zum einen kann ohne Aufwand die Logik portiert werden, zum anderen vorhandenes C#-Know-How für iOS, Android und Windows Store App eingesetzt werden.

3 Management Summary

3.1 Ausgangslage

Einen komplexen Sachverhalt, wie die Optimierung von Produktionsabläufen oder des Personaleinsatzes, einer nicht involvierten Person zu erklären ist ein schwieriges Unterfangen. Unterschiede im Fachwissen betreffend einer Problemdomäne können Missverständnisse zwischen Sender und Empfänger hervorrufen und zu erhöhten Kosten auf beiden Seiten führen. Die Gefahr besteht, dass die Person das Interesse am Sachverhalt verliert oder diesen nicht richtig versteht. Die Lösung dieses Problems heisst Gamification. Gamification hat das Ziel, dem Benutzer auf spielerische Art einen komplexen Sachverhalt zu erklären, ohne dass dies als Lernprozess wahrgenommen wird.

3.2 Zielsetzung

Die vorliegende Bachelorarbeit hat das Ziel, das Einsatzgebiet des von der SCS entwickelten Solvers zu erklären. Der Solver optimiert einen Ablauf, so dass möglichst wenige Übergangswechsel vorhanden sind. Neben der Spielerischen Erklärung, Gamification genannt, des Sachverhalts sind auch eine moderne Architektur sowie moderne Technologien gefordert, so dass dieses Projekt zu demonstrationszwecken verwendet werden kann.

- Plattform unabhängiger Kern der Applikation
- Spielerische Erklärung der Problemdomäne des Solvers
- Moderne Architektur und Technologien zur Verwendung als Demonstrationsprojekt

3.3 Vorgehen

Um die Problembeschreibung zu lösen, haben wir uns für ein agiles Vorgehensmodell ähnlich zu „Rational Unified Process (RUP)“ entschieden.

Anfangs haben wir die Problemstellung analysiert und daraus mögliche Risiken und deren Auswirkungen eruiert. Um die Risiken abzudecken, wurde ein Prototyp entwickelt, welcher alle Komponenten der Arbeit verwendet. Da die Aufgabenstellung den Einsatz von Xamarin für eine Plattformunabhängige Entwicklung fordert, wurde eine Windows Store und eine Android Applikation entwickelt.

Der Einsatz von Microsoft Azure ist ein weiterer Teil der Problembeschreibung, welcher durch den Prototypen abgedeckt werden soll. Damit dieser umgesetzt werden kann, muss der von SCS entwickelte C++-Solver in der Cloud zum Laufen gebracht werden.

Die Anforderungen an das Spiel wurden vom Industriepartner grob vorgegeben und während dieser Arbeit verfeinert. Im Fokus steht dabei die spielerische Erklärung der Funktionsweise des Solvers, welche im bestmöglichen Fall dem Spieler unbewusst erklärt wird.

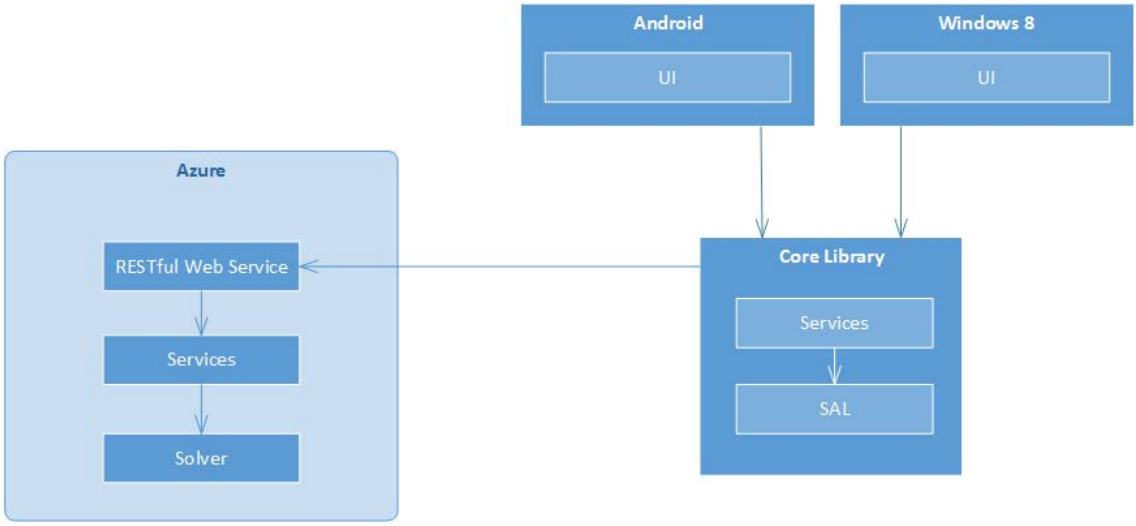


ABBILDUNG 1: PROTOTYPE-ARCHITEKTUR IN DER ERSTEN PHASE DES PROJEKTS

3.4 Resultat

Die Ergebnisse dieser Bachelorarbeit sind ein voll funktionsfähiges Spiel für den Windows Store, ein Android Prototyp als „Proof of Concept“ und der Solver, welcher in der Microsoft Azure Cloud läuft und über eine RESTful Schnittstelle angesprochen wird.

Im Spiel kann der Benutzer den Einsatzzweck des Solvers kennenlernen und versuchen die Abläufe so zu optimieren, dass eine optimale Anzahl von Wechsel erreicht wird. Für die Langzeitmotivation des Spielers sorgt ein globaler Highscore. Zusätzlich kann sich der Spieler im Direktvergleich mit dem Gegner und dem Solver messen.

Durch die Architektur des Clients, welche in einen Core Teil und in plattformspezifische Views aufgeteilt ist, ist dank Xamarin die gesamte Spiellogik auf Clientseite plattformunabhängig. Nur die Views und plattformspezifische Services müssen parallel für Android, Windows Store App und iOS erstellt werden. So werden nicht nur die Entwicklungskosten minimiert sondern auch die Sicherheit der Applikationen erhöht, da schneller auf allfällige Softwarelücken reagiert werden kann.

Diese Arbeit dient ebenfalls der SCS zur Demonstration der verwendeten Technologien. Durch das während dieser Bachelorarbeit gesammelten Wissen, ist die SCS nun fähig abzuschätzen, ob die verwendeten Technologien für sie interessant sind. Das Wissen, ob Xamarin oder Microsoft Azure eingesetzt werden sollten, ist für ein Unternehmen wertvoll, da so einerseits die entstehenden Initialkosten sowie die Personalkosten während der Überprüfung der Technologien gespart werden können.

Das Fazit für die Verwendung von Xamarin fällt sehr positiv aus. Die Lizenzen mit 1000 Franken pro Entwickler und pro Plattform sind zwar sehr hoch, doch diese Kosten werden durch die Zeitersparnis der plattformunabhängigen Entwicklung wieder kompensiert. Unsere Erfahrung zeigt, dass für das „portieren“ der Applikation, etwa 20 Prozent der ursprünglichen Zeit benötigt werden. Der Aufwand ist bei Business Applikation tendenziell tiefer, da diese in der Regel weniger graphische Spielereien voraussetzen und Standardkomponenten des Frameworks verwendet werden können.

Der grösste Vorteil von Windows Azure ist die Möglichkeit, dass ohne viel Aufwand horizontal sowie vertikal skaliert werden kann. Da die Notwendigkeit einer solchen Skalierung kurz nach der Veröffentlichung des Spiels noch nicht absehbar ist, ist das betreiben auf einem bereits bestehenden internen Server zu empfehlen.

3.5 Ausblick

Xamarin in Verbindung mit MvvmCross ermöglichen es, dass die Logik des Spiels nur einmal für IOS, Android und Windows Geräte geschrieben werden muss. Für die verschiedenen Plattformen muss die View und plattformspezifische Komponenten implementiert werden.

Die Fertigstellung des während dieser Arbeit erstellten Android Prototyps wäre eine mögliche zukünftige Arbeit. Zusätzlich wäre eine vollständige Implementierung für iOS Geräte in Betracht zu ziehen, womit die weitverbreitetsten Betriebssysteme auf dem Mobile-Markt abgedeckt wären.

Im Zuge dieser Bachelorarbeit wurden für die Generierung der dynamischen Spiel-Aufgaben (Matrizen) drei unterschiedliche Algorithmen entwickelt. Für jede Aufgabe erhält der Benutzer eine spezifische Zeit, in welcher er die Problemstellung zu lösen hat. Dieses Zeitfenster kann bei den Algorithmen eingestellt werden. Für die Bestimmung der idealen Grösse des Zeitfensters wurde eine graphische Unterstützung für die SCS erstellt. Nach einer Testphase, in welcher möglichst viele Spiele erfasst werden, kann das Zeitfenster mit einer Exponentialfunktion definiert werden, sodass sich ein Spiel-Flow einsetzt.

4 Ausgangslage

4.1 Gamification

Gamification ist mehr als nur ein Marketing-Modewort unserer Zeit. Einerseits kann mit Gamification gezielt ein Produkt oder ein Service beworben werden, andererseits können Benutzer mit Gamification trainieren, um komplexe Probleme in einem gegebenen System schneller zu lösen.

Die wichtigste Voraussetzung für den Erfolg von Gamification ist die Spielmotivation des Benutzers. Der Schwierigkeitsgrad muss über die Zeit zunehmen, damit häufige Spieler auch über längere Zeit motiviert bleiben. Um die Wiederspielbarkeit zu erhöhen, können die zu lösenden Aufgaben dynamisch erzeugt werden. Belohnungen der Errungenschaften ermöglichen dem Benutzer den Vergleich mit anderen Spielern, was zusätzlich motiviert.

Um eine Langzeitmotivation des Benutzers fördern zu können, muss der „Flow“ beachtet werden. Mihaly Csikszentmihalyi hat den Begriff „Flow“ in seiner Arbeit „Flow: The Psychology of Optimal Experience“ (1) definiert. Der Zustand des „Flows“ ist der optimale Zustand der intrinsischen Motivation, bei welcher die Person komplett in das Geschehen eintaucht und alles um sich herum vergisst. Dies geschieht durch das Gefühl von gänzlicher Absorption, bei welcher die Grundbedürfnisse ignoriert werden. Und an deren Stelle rücken das Gefühl von Erfüllung und das Streben nach Herausforderung.

4.2 Architekturaufbau

Als Architektur wird das Client-Server-Modell verwendet. Der Solver muss in die Cloud integriert werden. Das Spiel muss für Windows 8.1 sowie auf Android Geräten entwickelt werden. Die beiden Plattformen sollen dabei die gleiche Code Basis verwenden, so dass die Logik nur einmal implementiert werden muss.

4.3 Technologien

Die Bachelorarbeit soll zur Demonstration von modernen Technologien dienen. Die SCS will mit dieser Arbeit nicht nur ihr Produkt, den Solver, bewerben, sondern auch aufzeigen, was mit den aktuellen Technologien möglich ist und wo mögliche Probleme vorhanden sind.

Der von der SCS entwickelte Solver soll in der Microsoft Azure Cloud betrieben werden können. Azure bietet verschiedene Services von „Software as a service“ bis zu „Infrastructure as a service“, bei welcher ein virtueller Server in der Cloud betrieben wird, an. Der Solver ist ein in C++ Programm, welches nativ auf der Plattform läuft. Der Solver wird über eine RESTful Schnittstelle angesprochen.

Damit die Logik des Spiels unabhängig von der mobilen Plattform nur einmal implementiert werden muss, wird Xamarin und MvvmCross verwendet. Dies ermöglicht, dass die Schnittstelle zum Server, die Navigationslogik und die Viewmodels nur einmal implementiert werden müssen.

5 Problembeschreibung

5.1 Vision der Arbeit

5.1.1 Demoapplikation

Die während dieser Arbeit erstellte Anwendung soll der SCS einerseits zu Demonstrationszwecken des Solvers sowie der verwendeten Technologien dienen, andererseits werden die von uns gesammelten Erfahrungen für zukünftige Designentscheide einfliessen.

5.1.2 Gamification

Das Spiel soll die Langzeitmotivation des Benutzers fördern sowie die Funktionsweise des Solvers auf Spielerische Art und Weise beschreiben.

5.2 Stand der Technik

5.2.1 Browserspiel

Ein einfaches Browserspiel mit 6 Levels, lässt die Besucher der SCS Webseite (http://optimizer.scs.ch/) die Ablaufkoordination, welches der Solver löst, spielen.

Die Lösungen der 6 Levels sind fix in einem XML-File hinterlegt. Die Spielsteine welche der Benutzer verschieben kann werden bei jedem Spiel zufällig auf der Spielfläche verteilt.

ABBILDUNG 2: STATISCHER WEB-PROTOTYP DES OPTIMIZERS

5.2.2 Solver

Der Solver ist ein in C++ geschriebenes Programm, welches ein bestmögliches Resultat für die Ablaufkoordination einer gegebenen Problemstellung berechnet. Der Solver verwendet intern je nach Problemstellung unterschiedliche Algorithmen.

Das mathematische Problem auf welches die Problemstellung abgebildet werden kann ist das Traveling Salesman Problem.

Der Solver verwendet als Input und als Output XML-Files. Während der Bearbeitung der Problemstellung schreibt der Solver temporäre Files welche immer die gleichen Namen haben.

5.3 Anforderungen

5.3.1 UC

Das Bild unterhalb zeigt Use Cases für das System. Die wichtigsten (komplexeren) davon werden im nächsten Kapitel detaillierter beschrieben.

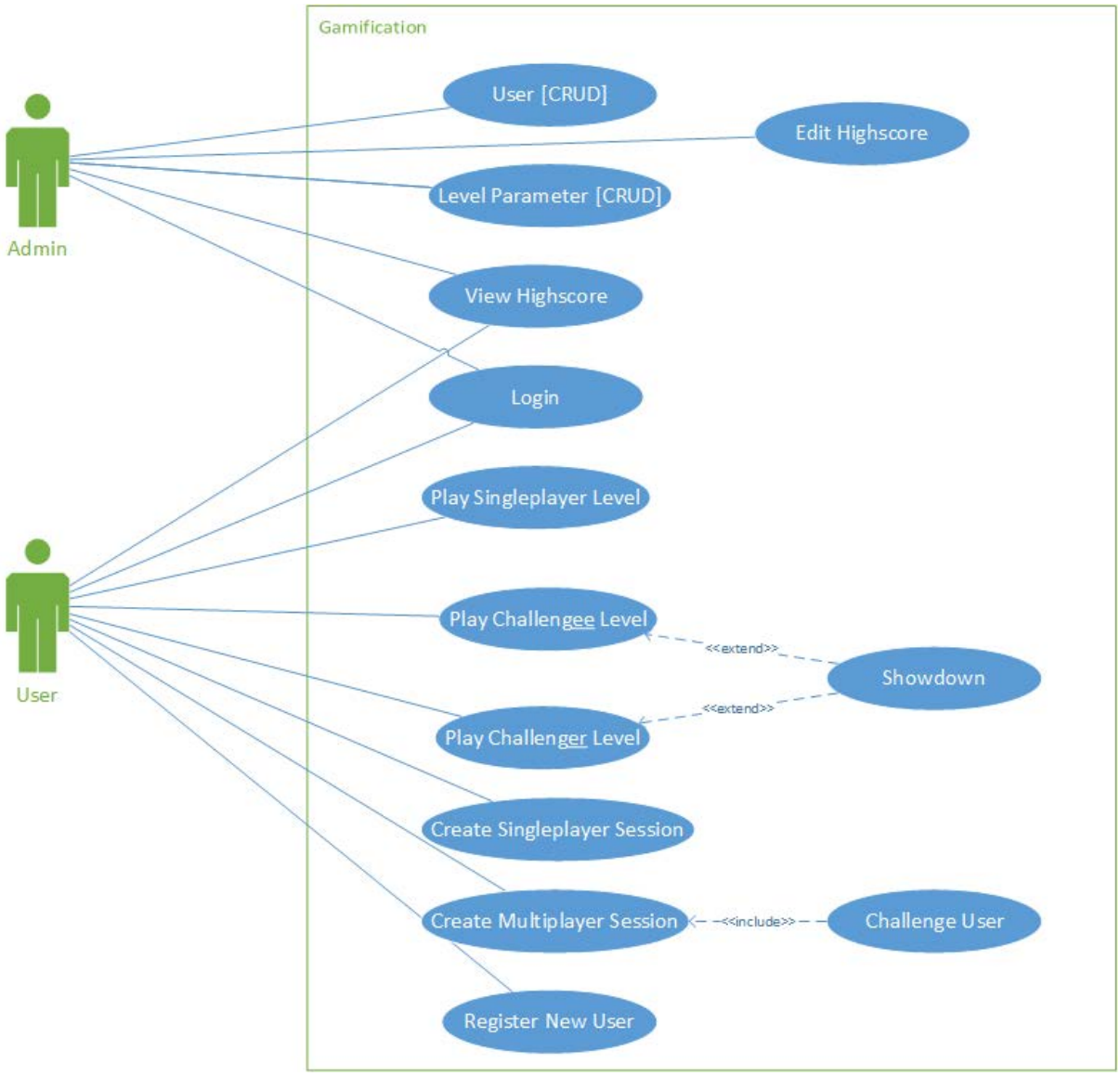


ABBILDUNG 3: USE CASE DIAGRAMM DES GESAMTSYSTEMS

Zahlen in Auflistungen (1., 2., 3., ...) geben chronologische Schritte an. Die Buchstaben (a., b., c., ...) hingegen beschreiben alternative Pfade. Bsp.:

1. System präsentiert „Datei überschreiben“-Dialog.
    a. Benutzer klickt „Ja“.
        1. Änderungen werden gespeichert.
        2. Applikation wird geschlossen.
    b. Benutzer klickt „Nein“.
        1. Dialog wird geschlossen.
        2. System zeigt aktuellen Inhalt

Register New User

Actor	Neuer Benutzer
Scope	Solver-Anwendung
Brief	Ein neuer Spieler möchte sich registrieren, um das Game spielen zu können.
Preconditions	• Es ist kein Spieler am System angemeldet.
Success Guarantees	• Benutzerinformationen (Username, Password,...) sind in der Datenbank gespeichert. • Benutzer ist angemeldet.
Main Success Szenario	1. Benutzer gibt Username und Passwort ein. 2. System informiert Benutzer über erfolgreiche Erstellung des Accounts und meldet diesen automatisch an. 3. Das System zeigt das Menu an.
Extensions	-

Create Multiplayer Session / Challenge User

Actor	User
Scope	Multiplayer Session
Brief	Der User möchte ein neues Spiel gegen einen anderen User beginnen.
Preconditions	• Der User ist angemeldet und befindet sich im Multiplayer-Menu.
Success Guarantees	• Eine Session ist für beide User eröffnet. • Der Herausgeforderte erhält eine Nachricht, dass eine neue Session verfügbar ist und sieht diese im Multiplayer-Menu.
Main Success Szenario	1. Der User eröffnet ein Multiplayer-Spiel und gibt den Username des Gegners ein. 2. Das System meldet, dass die Einladung erfolgreich abgeschickt wurde und man auf den ersten Zug des Herausgeforderten warten muss. 3. Der User erhält eine Übersicht über die aktuellen Sessions.
Extensions	2a. 1. Das System meldet, dass bereits eine Session mit diesem User existiert und deshalb keine Neue erstellt werden konnte. 2. Gehe zu Schritt 3. 2b. 1. Das System meldet, dass der eingetragene Benutzername nicht existiert. 2. Gehe zu Schritt 3.

Create Singleplayer Session

Actor	User
Scope	Singleplayer Session
Brief	Der User möchte ein neues Singleplayer-Spiel beginnen.
Preconditions	• Der User ist angemeldet und befindet sich im Haupt-Menu.
Success Guarantees	• Eine Session ist für den Spieler eröffnet. • Das System liefert einen neuen Level. *Crossreference: Play Level*
Main Success Szenario	1. Der User klickt auf den entsprechenden Button. 2. Das System zeigt den ersten Level der neuen Session an.
Extensions	2a. 1. Das System zeigt eine Warnung an, dass bereits eine offene Session existert und fragt nach, ob diese überschrieben (geschlossen) werden soll. a. User klickt „Ja“. 1. Gehe zu Schritt 2. b. User klickt „Nein“. 1. System zeigt das Menu an.

Play Singleplayer Level

Actor	User
Scope	Singleplayer Session
Brief	Der User möchte die nächste Aufgabe im Singleplayer-Modus lösen.
Preconditions	• Der User ist angemeldet und befindet sich im Singleplayer-Menu. • Eine Singleplayer Session befindet sich im offenen Zustand. *Crossreference: Create Singleplayer Session*
Success Guarantees	• Die Lösung des Spielers wird in der Datenbank gespeichert. • Der Status der Session (closed, finished, canceled, …) wird aktualisiert.
Main Success Szenario	1. Der User startet das Level. 2. Das System präsentiert (und generiert) die Aufgabenstellung (Matrix und verbleibende Zeit). 3. Der User optimiert die Matrix bis er die perfekte Lösung findet, bevor die Zeit abgelaufen ist. 4. Das System meldet den erfolgreichen Abschluss des Levels, speichert diesen, und präsentiert den Zeitbonus, den der Spieler in das nächste Level „mitnehmen“ darf. Die Optionen „Gehe zu Menu“ und „Nächstes Spiel starten“ werden angezeigt. 5. Der Spieler startet das nächste Spiel.
Extensions	3a. 1. Die Zeit ist abgelaufen bevor die Lösung erreicht wurde. 2. Das System speichert die aktuelle Lösung. 3. Das System zeigt die Lösung des Solvers an. 4. Das System meldet dem Benutzer, dass die Session beendet ist und der User eine neue Session starten kann. a. Gehe zu Schritt 5. b. System zeigt Menu an. 3b. 1. Der Benutzer möchte aufgeben und klickt auf die weisse Flagge. 2. Das System meldet dem Benutzer, dass die Session beendet ist und der User eine neue Session starten kann. a. Gehe zu Schritt 5. b. System zeigt Menu an. 5a. 1. Der Spieler geht zurück ins Menu.

Play Challengee Level

Actor	User = Challengee
Scope	Multiplayer Session
Brief	Der User befindet sich in einer Multiplayer-Session und ist in der entsprechenden Runde als erster am Zug.
Preconditions	• Der User ist angemeldet und befindet sich im Multiplayer-Menu. • Eine Multiplayer Session befindet sich im offenen Zustand und der User ist an der Reihe. *Crossreference: Create Multiplayer Session* • Der User ist in der aktuellen Runde als Challengee registriert.
Success Guarantees	• Die Lösung des Spielers wird in der Datenbank gespeichert. • Der Status der Session (closed, finished, canceled, …) wird aktualisiert. • Die Rollen Challenger und Challengee werden getauscht. • Der Gegenspieler wird über den Erfolg des Users mitgeteilt (nächste Runde oder Endresultat anzeigen).
Main Success Szenario	1. Der User klickt im Menu auf die entsprechende Session. 2. Das System fragt nach dem gewünschten Level. 3. Der User gibt das Level an und bestätigt. 4. Das System generiert das Level und präsentiert es dem User. 5. Der User optimiert die Matrix, bis er damit zufrieden ist und bestätigt sein Ergebnis. 6. Das System zeigt die Zeit und Changeovers der gelösten Aufgabe an und leitet anschliessend den User in das Multiplayer-Menu weiter.
Extensions	1a. 1. Nach dem Klick präsentiert das System den Showdown, falls dieser noch nicht angezeigt wurde. *Crossreference: Showdown* 2. Gehe zu Schritt 2.

Play Challenger Level

Actor	User = Challenger
Scope	Multiplayer Session
Brief	Der User befindet sich in einer Multiplayer-Session und muss in der entsprechenden Runde auf die Vorgaben (Zeit und Anzahl Changeovers) reagieren.
Preconditions	• Der User ist angemeldet und befindet sich im Multiplayer-Menu. • Eine Multiplayer Session befindet sich im offenen Zustand und der User ist an der Reihe. *Crossreference: Create Multiplayer Session* • Der User ist in der aktuellen Runde als Challenger registriert, d.h. es existiert eine Vorgabe der Aufgabenstellung durch den Challengee.
Success Guarantees	• Die Lösung des Spielers wird in der Datenbank gespeichert. • Der Status der Session (closed, finished, canceled, …) wird aktualisiert. • Die Rollen Challenger und Challengee werden getauscht. • Der Gegenspieler wird über den Erfolg des Users mitgeteilt (nächste Runde oder Endresultat anzeigen).
Main Success Szenario	1. Der User klickt im Menu auf die entsprechende Session. 2. Das System präsentiert dem User die gleiche Aufgabenstellung (Matrix) wie dem Challengee. 3. Das System limitiert die verfügbare Zeit zum Lösen der Aufgabe anhand der Vorgabe durch den Challengee. 4. Der User optimiert die Matrix, bis er damit zufrieden ist und bestätigt sein Ergebnis. 5. Das System zeigt die Zeit und Changeovers der gelösten Aufgabe an. 6. Der Showdown wird angezeigt. 7. Das System leitet anschliessend den User in das Multiplayer-Menu weiter.
Extensions	4a. 1. Die Zeit des Users ist abgelaufen und die Runde wird mit der aktuellen Lösung (Matrix) beendet.

5.3.2 Nonfunctional Requirements

Security

Client-Server Connection

Die Kommunikation zwischen dem Spiel und der Web-API-Schnittstelle muss abhörsicher sein, da sich die Clients über Benutzer/Passwort authentifizieren. Um dies zu garantieren wird eine TLS-Verbindung mit Server-Zertifikat eingesetzt.

Performance

Antwortzeiten

Die Antwortzeiten sind auf eine Verbindung zwischen Server und Client mit mindestens 10 Mbit bezogen.

Verbindungsaufbau inkl. TLS, Anmeldung	<= **2 Sekunden**
Senden einer Lösung	<= **3 Sekunden**
Neues Spiel erstellen	<= **3 Sekunden**
Anzeige refreshen	<= **3 Sekunden**

Reaktionszeiten

Die Antwortzeiten beziehen sich auf ein Surface mit Windows 8.1 RT.

Spiel starten	**<= 5 Sekunden**
Training Level **starten**	**<= 1 Sekunde**

Stabilität

Keine Internetverbindung

Ohne Internetverbindung muss der Training Modus verwendet werden können.

Portierbarkeit

Core Library für Android, Windows 8

Die Applikation ist so aufgebaut, dass die Core Library, welche die Spiellogik sowie die Komponenten zur Kommunikation mit dem Server beinhaltet, mit Xamarin Android verwendet werden kann.

6 Lösungskonzept

6.1 Benutzeranalyse

6.1.1 Zielgruppe

Die Zielgruppe der Applikation sind Benutzer ohne Kenntnisse über die Funktionsweise oder die Problemdomäne des Solvers.

6.1.2 Interview

Ethnographisches Interview

Für die Benutzeranalyse wird ein ethnographisches Interview (2) verwendet.

Interview-Partner

X

Der Interviewpartner spielt regelmässig verschiedene Computerspiele und hat keinerlei Kenntnisse über den Solver.

Einleitung

Dem Interviewpartner wurde kurz das Ziel dieses Interviews erklärt.

Fragen und Antworten

Wieso spielst du Computerspiele?

Weil diese mir Spass machen. Ich spiele am liebsten Multiplayerspiele, in welchen ich mich mit anderen messen kann. Spannend finde ich, wenn Allianzen mit anderen Spielern geschlossen werden können, um besser zu gewinnen.

Spielst du mehrmals gegen die gleiche Person in einem Multiplayerspiel?

Ja. Denn einmal gewinnt er, und einmal ich. Es ist viel spannender wenn ich im Direktvergleich sehe, dass ich im Gegensatz zu meinem Gegner besser geworden bin (Interviewpartner lacht).

Das heisst dir ist wichtig, dass du weisst wo im Ranking du stehst?

Natürlich! So kann ich sehen ob ich besser geworden bin oder nicht. Und ich kann auch die besseren Spieler sehen und gegen diese versuchen zu gewinnen.

Was ist dir bei einem Spiel wichtig?

Das Spiel muss eine Herausforderung sein. Wenn es zu einfach ist, spiele ich es 5 Minuten und anschliessend nie wieder.

Was auch noch wichtig ist, ist das es keine Fehler hat. Es stört mich nicht mehr, als wenn das Spiel während dem Spielen abstürzt oder sonst eine Macke hat.

Und die riesigen Updates stören mich auch. Wenn ich das Spiel starte, dann will ich spielen und nicht noch eine halbe Stunde warten müssen, bis das Spiel auf dem aktuellen Stand ist.

Das ich mit meinem Gegner kommunizieren kann ist mir auch wichtig, dass wenn ich gewonnen habe, ich ihnen mitteilen kann, wie schlecht sie sind.

Fazit

- Direktvergleich mit anderen Spielern
- Richtiger Schwierigkeitsgrad, nicht zu einfach, nicht zu leicht
- Gute Bedienung, damit für sich das Spiel wie erwartet verhält. Sonst wird dies für ein Bug gehalten
- Schnelle Startzeit

6.2 Big Picture

Wie in Abbildung 4 zu erkennen ist, kann das System grob in zwei Komponenten eingeteilt werden. Auf der linken Seite sehen wir den Server-Teil, die Pfeile zeigen dabei die Richtung der Abhängigkeit an. Die erste Schnittstelle (MVC) ist über den Browser zu erreichen und dient der Administration des Systems. Dort können verschiedene Reports mit Statistiken zu den Spielen angesehen oder Einstellungen verändert werden. Via eines RESTful-Web-Services wird die Business-Logik dem Client zur Verfügung gestellt. Beide Schnittstellen bauen dabei auf dem Service-Layer auf, welcher Zugriff auf die Daten (DAL), den Solver und die Domain-Logik besitzt.

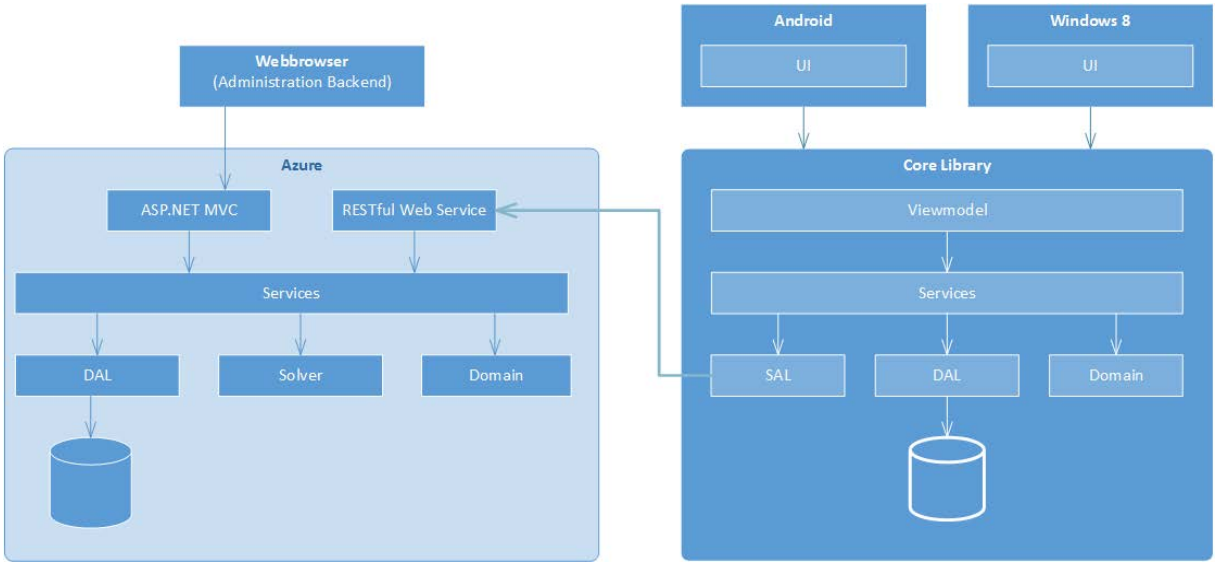


ABBILDUNG 4: BIG PICTURE

Der Client auf der rechten Seite kann selbst wieder in zwei Teile gegliedert werden. Unterhalb sehen wir die Core Library, die möglichst viel Funktionalität plattformunabhängig für die benötigten Betriebssysteme zur Verfügung stellt. Wie in der Server-Komponente werden auf dem Client Services angeboten, welche Infracture- und Domain-Subsysteme verwenden. Der Service Access Layer (SAL) ist dabei für die RESTful Anbindung an die Server-Komponente verantwortlich. Viewmodels in der Core Library bieten den plattformspezifischen Komponenten (Android, Windows 8) die Schnittstellen zwischen UI und Businesslogik.

6.3 Spielkonzept

6.3.1 Ziel des Spiels

Unser Ziel als Entwickler der Software ist es, den Spielern zu vermitteln, welche Art von Problem der Solver-Algorithmus der SCS lösen kann. Der Spieler schlüpft dabei in die Rolle des Solvers und muss sich gegen diesen behaupten. Dabei gibt es zwei verschiedene Spielmodi:

- Singleplayer (inkl. Demo-Modus)
- Multiplayer

Das Prinzip der zu lösenden Aufgaben ist in beiden Modi das selbe:

> Minimiere die Anzahl der Farbwechsel innerhalb einer Spalte!

Als Beispiel sehen wir in Abbildung 5 eine zweidimensionale Matrix, die mit unterschiedlichen Symbolen gefüllt ist. Oberhalb der Spalte sehen wir Pfeile, die für jede Spalte die Farbwechsel, auch Changeovers genannt, anzeigen. Insgesamt sind also bei dieser Aufgabe sechs Changeovers vorhanden.

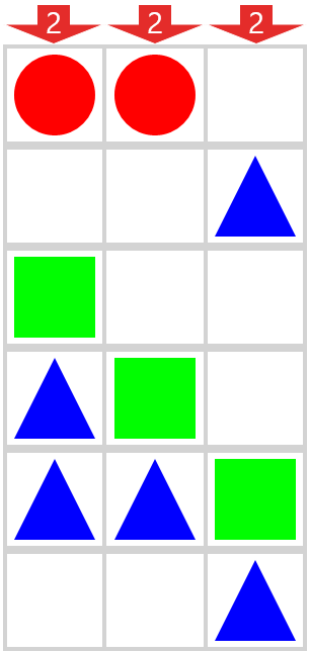


ABBILDUNG 5: TYPISCHE AUFGABENSTELLUNG DES OPTIMIZER-SPIELS

Zur Visualisierung der Changeovers wurden in Abbildung 6 Pfeile (1-6) zur Kennzeichnung der Farbwechsel eingefügt. Wie zu sehen ist, verursachen leere Felder kein zusätzlicher Wechsel, aber auch kein Auflösung eines Changeovers – sie werden einfach übersprungen.

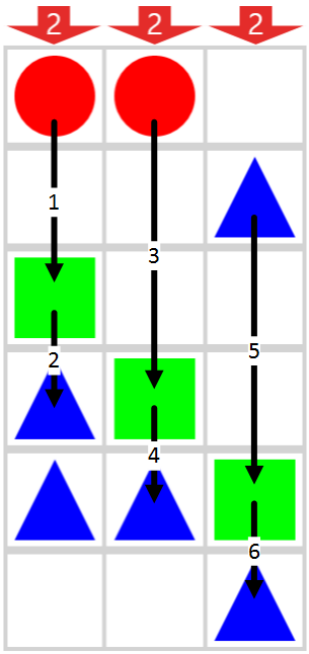


ABBILDUNG 6: AUFGABENSTELLUNG MIT VISUALISERTEN CHANGEOVERS

Die Symbole, auch Spielsteine genannt, können innerhalb der eigenen Reihe beliebig verschoben oder mit anderen Steinen getauscht werden. In Abbildung 7 sehen wir eine optimale Lösung der vorherigen Aufgabenstellung.

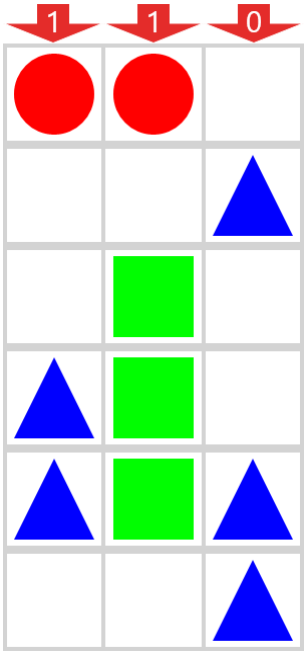


ABBILDUNG 7: OPTIMIERTE MATRIX MIT MÖGLICHST WENIGEN CHANGEOVERS

Es ist zu erkennen, dass nur noch zwei Changeovers, in den ersten zwei Spalten nach den roten Spielsteinen, vorkommen. Der Spieler kann die Formen verschiedenartig anordnen um diese Anzahl von Farbwechsel zu erreichen. Die Position der Spielsteine spielt für das erfolgreiche Abschliessen der Aufgabe keine Rolle, solange die minimale Anzahl an Changeovers erreicht wurde.

In fortgeschrittenen Runden (Levels), können die Reihen als ganze Einheit verschoben werden. Dabei ist das Verrücken eines einzelnen Steines in eine andere horizontale Ebene nicht zulässig. Die Changeovers werden dabei wie üblich gezählt.

6.3.2 Singleplayer

In diesem Modus tritt der Benutzer direkt gegen den Solver an. Der Spieler bekommt anfänglich leichtere Aufgaben zu lösen, die in jedem Durchgang schwieriger werden, bis der Benutzer scheitert. Kann er das gegebene Problem nicht mehr lösen, wird ihm die Lösung und ultimativ die Überlegenheit des Solvers präsentiert.

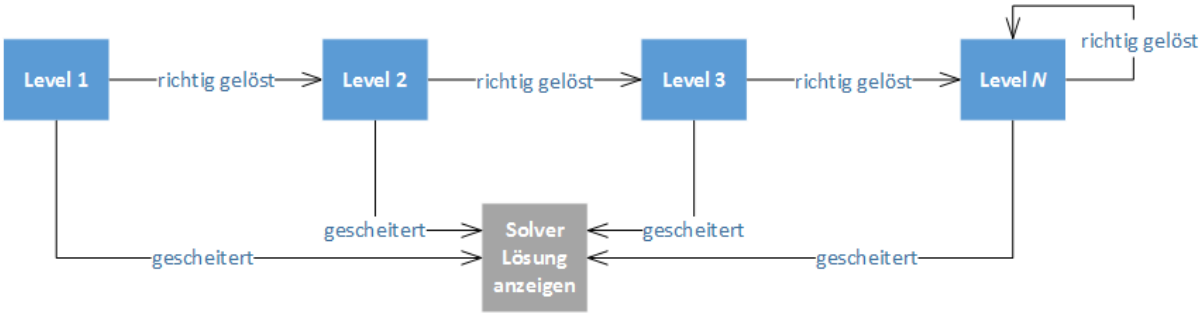


ABBILDUNG 8: LEVEL-SYSTEM DES SINGLEPLAYER-MODUS

Die Spiel-Session ist zu diesem Zeitpunkt zu Ende und muss wieder neu gestartet werden, d.h. von vorne bei Level 1 beginnen.

Der Demo-Modus entspricht dem Singleplayer-Modus, aber mit statischen Aufgaben, die bereits auf mit der App ausgeliefert werden. In diesem Modus kann auch ohne Internet-Verbindung, z.B. an einer Messe, eine Demonstration des Spiels gezeigt werden.

Singleplayer- und Demo-Levels sind zeitliche eingeschränkt, das heisst, es steht nur eine bestimmte Dauer zur Lösung der Aufgabe bereit. Zeit, welche nach erfolgreichem Abschluss einer Runde übrig ist, wird in die Nächste übertragen. Das folgende Szenarion ist als Beispiel zu betrachten:

1. Der Spieler Max startet das erste Level und hat 1 Minute Zeit dieses zu lösen.
2. Nach 20 Sekunden findet er die optimale Lösung (hätte also noch 40 Sekunden zur Verfügung gehabt) und wird in das zweite Level weitergeleitet.
3. Im zweiten Level bekommt jeder Spieler zwei Minuten Zeit dieses zu lösen. Max hat insgesamt nun 2 Minuten und 40 Sekunden um das zweite Level fertig zu stellen.

Über das Back-End (siehe Kapitel 7.3.4) kann die Formel für die Zeit der Singleplayer-Runden definiert werden.

6.3.3 Multiplayer

Der Multiplayer-Modus lässt zwei Spieler gegeneinander antreten. Pro Runde erhalten beide Benutzer die gleiche Problemstellung, allerdings gibt der beginnende Spieler, das Level (Schwierigkeitsgrad des Problems) und die zur Verfügung stehende Zeit vor. Haben beide Spieler die Aufgabe abgeschlossen, wird ein Showdown angezeigt, der die beiden Lösungen der Spieler untereinander vergleicht. Die optimalere (bessere) Lösung erhält einen Punkt. Zusätzlich wird im Showdown noch die Lösung des Solvers präsentiert – allerdings hat diese keinen Einfluss auf die Punktevergabe, sondern dient ausschliesslich der Präsentation des Solver-Algorithmus‘.

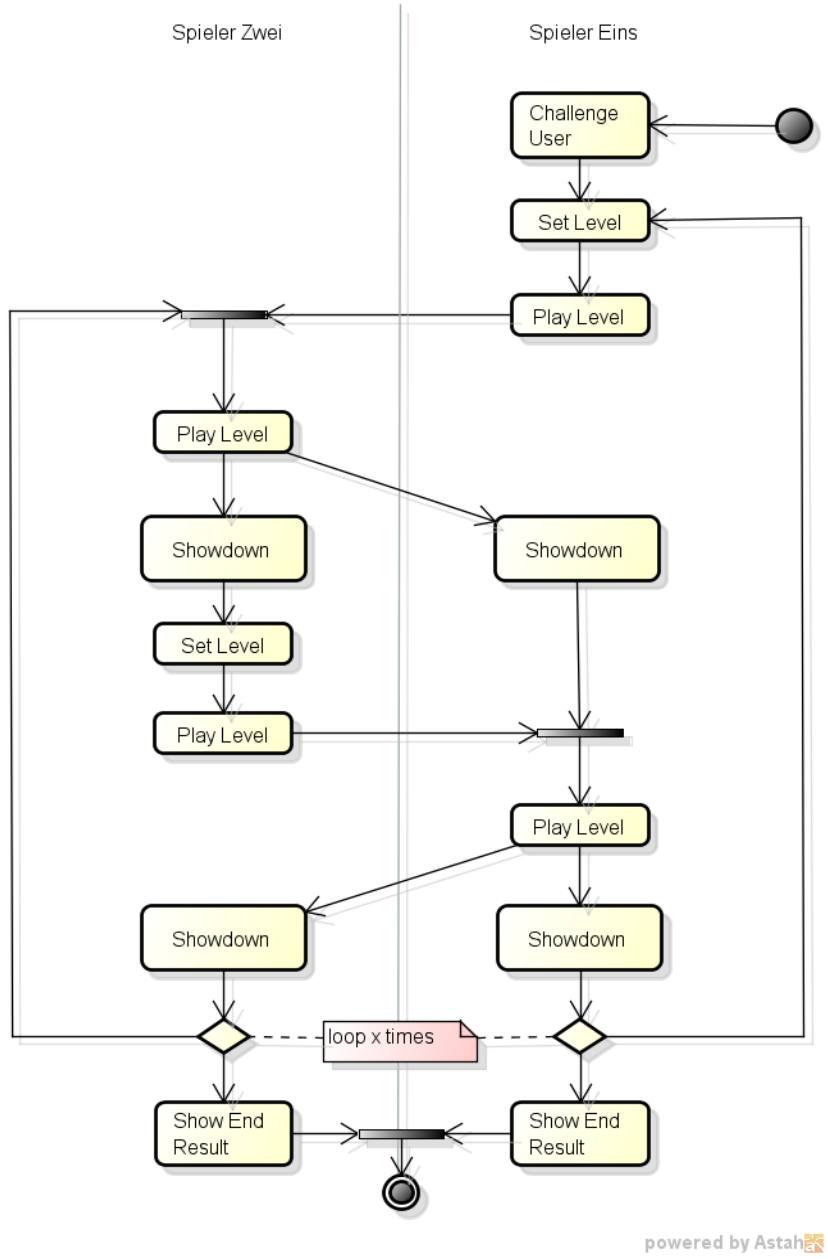


ABBILDUNG 9: SPIELABLAUF DES MULTIPLAYER-MODUS

Wie in Abbildung 9 zu sehen ist, setzt der Herausforderer das Level, nachdem er den Gegner ausgesucht hat (*Challenge User*). Der beginnende Spieler hat keine Zeiteinschränkung zum Lösen des Problems – allerdings wird die benötigte Zeit gestoppt und dient dem zweiten Spieler als Zeiteinschränkung. Nach dem Showdown werden die Rollen getauscht. Sind eine definierte Anzahl Runden durchgespielt (*loop x times*) wird das Endresultat präsentiert.

6.3.4 Mockups

Das Spiel bietet 3 verschiedene Spielmodi, Singleplayer, Multiplayer und ein Training Modus, welcher gleichzeitig als Demo Modus für Ausstellungen verwendet werden kann.

Startmenü

Das Startmenü ist der Einstiegspunkt des Spiels. Hier muss sich der Benutzer für einen der drei Spielmodi entscheiden.

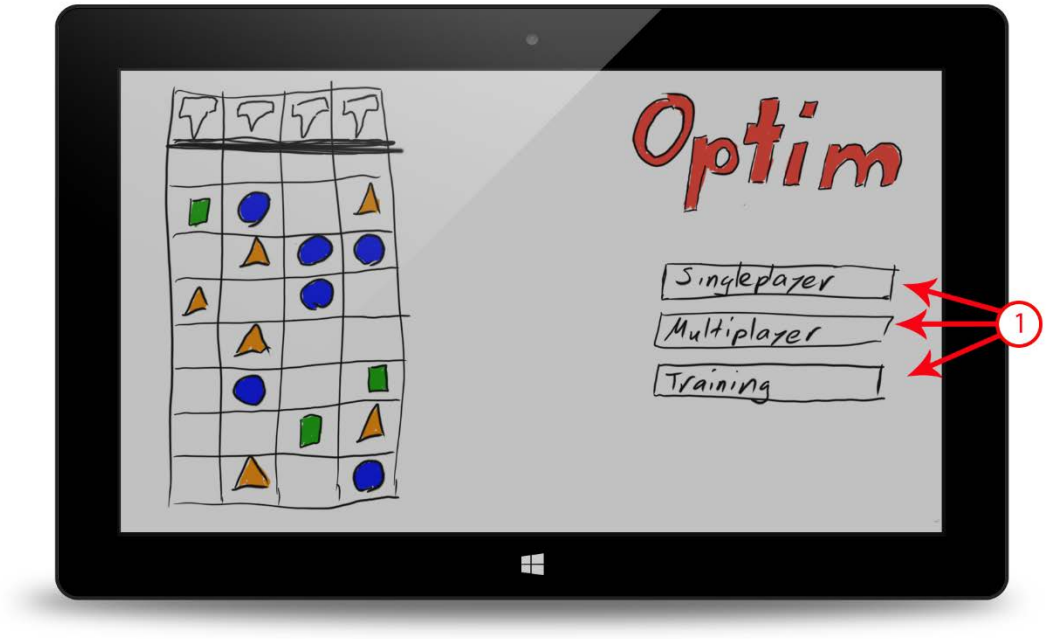


ABBILDUNG 10: MOCKUP DES HAUPTMENUS

1. Verschiedene Spielmodi

Singleplayer

Singleplayermenü

Sobald im Startmenü der Singleplayermodus ausgewählt wurde, wechselt der Screen auf den Singleplayer Startscreen. Hier kann der Benutzer ein neues Singleplayer Spiel starten oder ein bereits bestehendes wieder aufnehmen. Der Highscore gibt dem Benutzer an, wer die besten Spieler sind und an welcher Stelle er steht. Es werden so viele Benutzer wie möglich vor dem Spieler, beginnend bei Position 1, angezeigt.

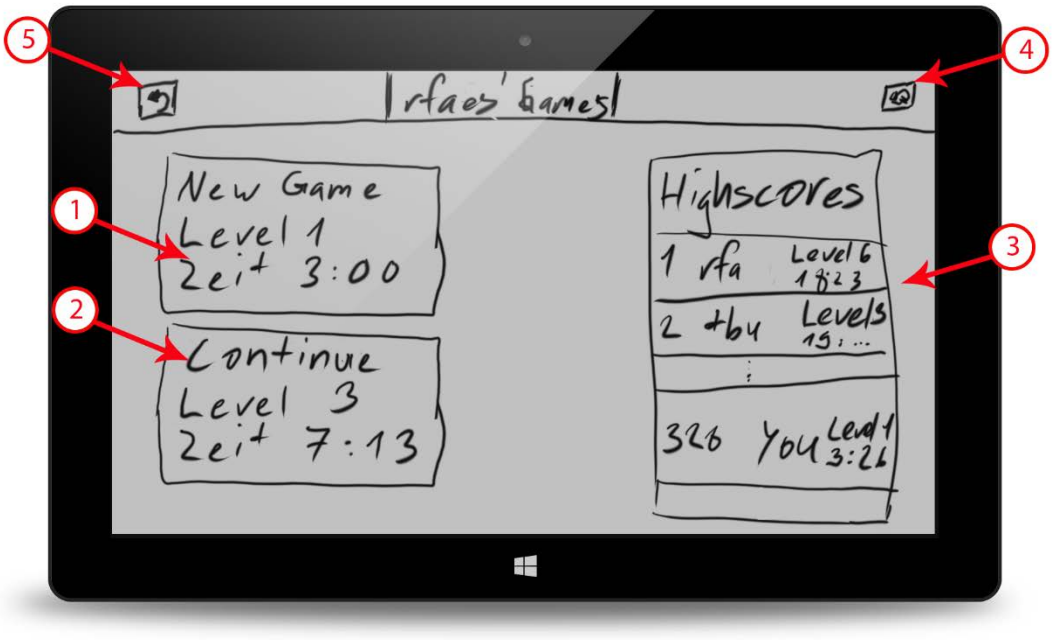


ABBILDUNG 11: MOCKUP DES SINGLEPLAYER-MENUS

1. Ein neues Spiel wird gestartet. Falls ein Spiel weitergespielt werden könnte, wird dieses mit dem neuen Spiel überschrieben.
2. Das zuletzt gespielte Spiel kann weitergespielt werden.
3. Der weltweite Highscore, die obersten Spieler und das beste eigene Spiel werden immer angezeigt.
4. Refresh Button
5. Zurück zum Singleplayer Menü

Spiel

Ein Singleplayer Spiels läuft immer gleich ab, das Spiel zeigt den Level und die verfügbare Zeit an. Sobald der Level gelöst ist, wird die gewonnene Zeit für den nächsten Level angezeigt und der Benutzer kann entscheiden, ob er den nächsten Level gleich oder zu einem späteren Zeitpunkt spielen will. Das Level ist gelöst, wenn die minimale Anzahl von Changeovers (Solver-Lösung) erreicht wurde.

Level Information

Das Spiel zeigt in einem Popup den Level und die verfügbare Zeit an.

ABBILDUNG 12: SPLASH-SCREEN VOR DEM BEGINN EINES SINGLEPLAYER-TURNS

Lösen des Levels

Der Benutzer sieht anschliessend zum Popup die Herausforderung. Hier muss der Spieler die Optimale Lösung innerhalb der gegeben Zeit erreichen. Die Zeit die übrig bleibt, wird dem nächsten Level gut geschrieben. Falls der Spieler die gegebene Aufgabe nicht lösen kann oder will, kann er jederzeit das Spiel abbrechen, indem er auf den entsprechenden Button drückt und muss somit nicht die verbleibende Zeit abwarten.

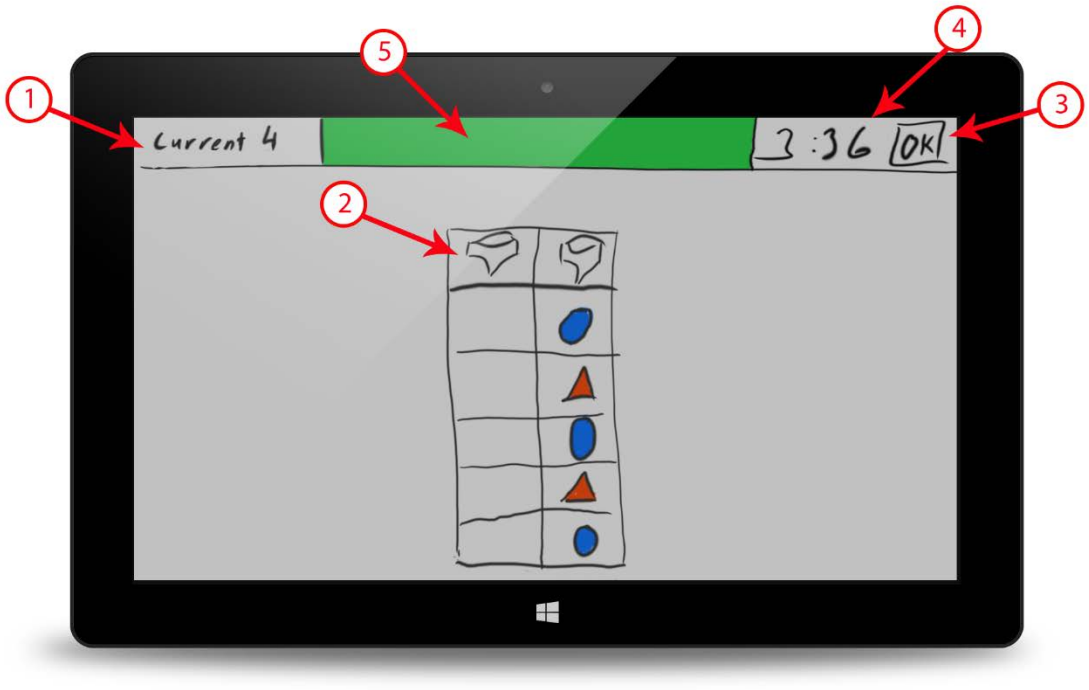


ABBILDUNG 13: MOCKUP EINES EINFACHEN SINGLEPLAYER-LEVELS

1. Optimale Anzahl der Wechsel und die eigene Anzahl
2. Die zu lösende Aufgabe
3. Aufgeben (white flag)
4. Noch verfügbare Zeit
5. Noch verfügbare Zeit in Form eines Balkens, Balken wird kleiner und ändert die Farbe zu einem rot

So sieht der Screen nach verstrichenen 1.5 Minuten aus. Ist die minimale Anzahl von Changeovers erreicht, wird das Spiel automatisch beendet.

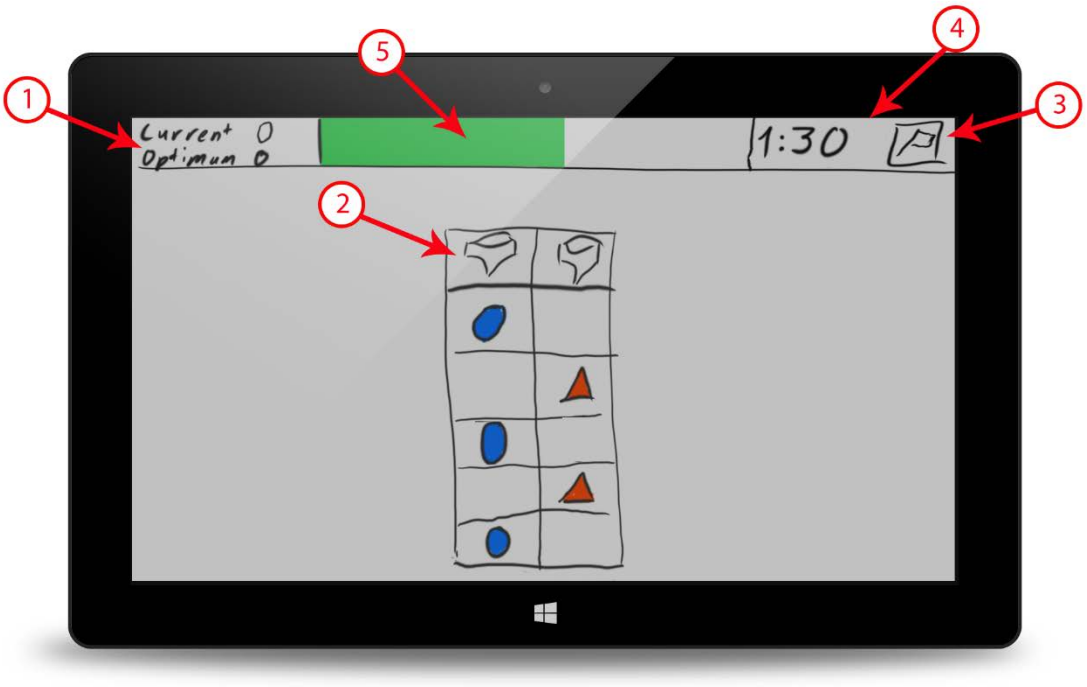


ABBILDUNG 14: IN-GAME MOCKUP DES SINGLEPLAYER-MODUS

Fertigstellung der Aufgabe

Sobald der Spieler die optimale Lösung erreicht hat, erscheint ein Popup, welches ihn über die „gewonnene“ Zeit informiert. Die verbleibende Zeit wird in das nächste Level übertragen.

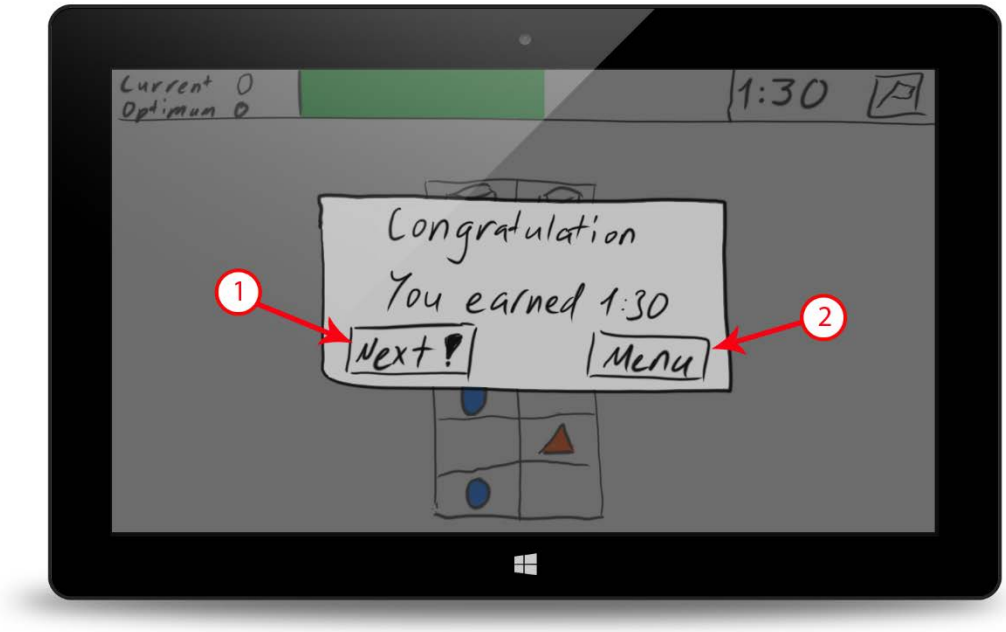


ABBILDUNG 15: MOCKUP EINES ERFOLGREICH GELÖSTEN SINGLEPLAYER-SPIELS

1. Nächste Aufgabe direkt starten
2. Zurück zum Menü. Die nächste Aufgabe kann vom Menü aus zu einem späterem Zeitpunkt gestartet werden

Aufgabe nicht in der Zeit gelöst

Sollte die Aufgabe nicht innerhalb der gegebenen Zeit gelöst werden können, kann wird dem Spieler ein Popup mit dem zuletzt fertig gestelltem Level und den zu diesem Zeitpunkt verfügbaren Zeit.

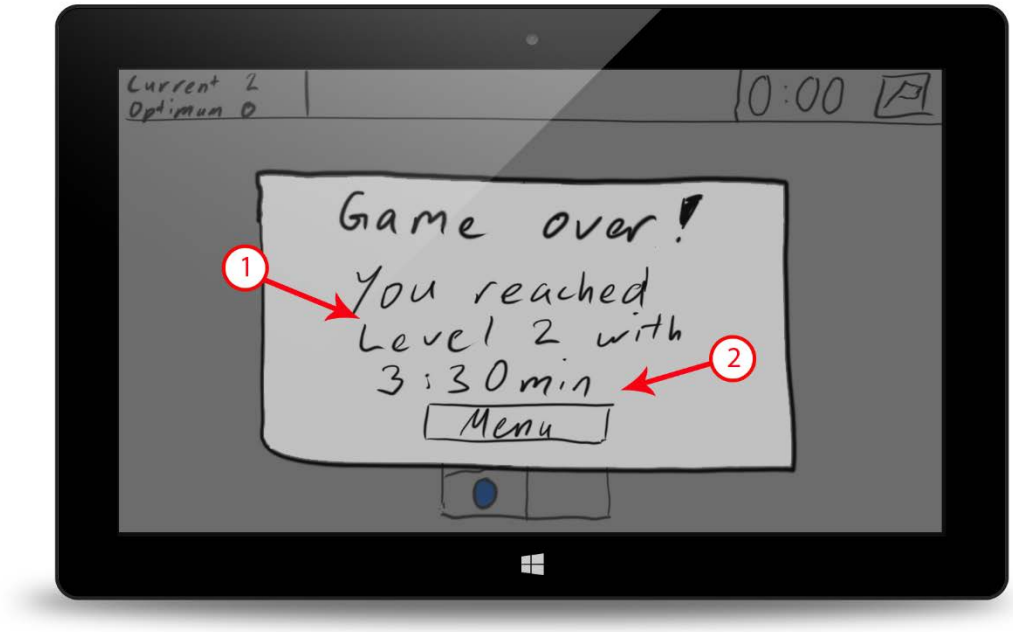


ABBILDUNG 16: GAME-OVER-DIALOG DES SINGLEPLAYERS

1. Fertig gespielter Level und die zur Fertigstellung verfügbaren Zeit
2. Zurück zum Menü

Multiplayer

Sobald im Startmenü der Multiplayermodus ausgewählt wurde, wechselt der Screen auf den Multiplayer Startscreen.

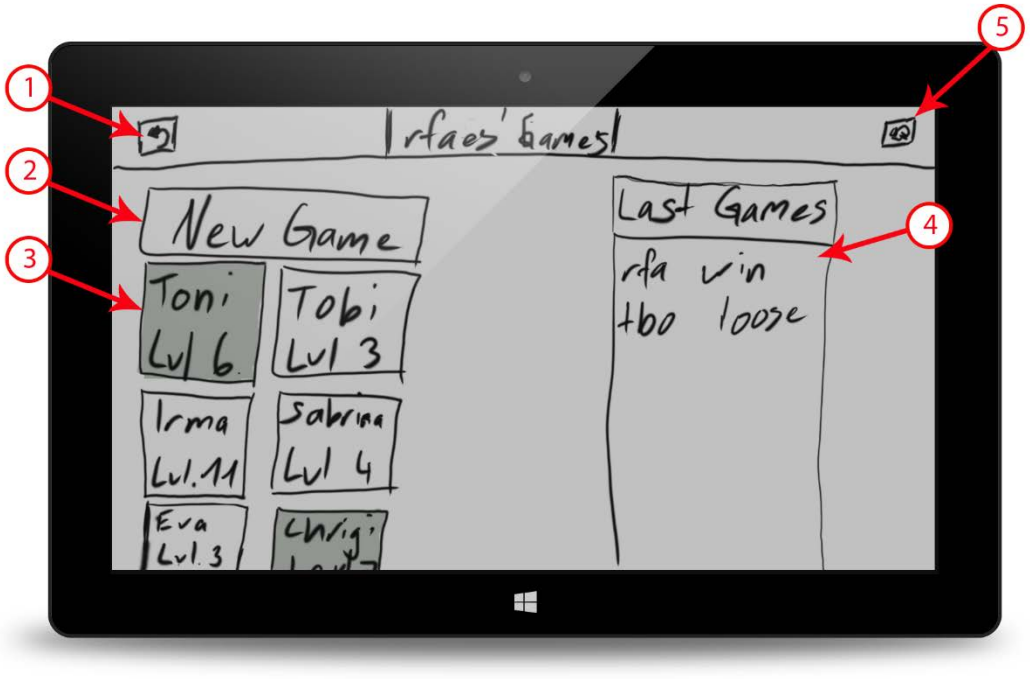


ABBILDUNG 17: MOCKUP DES MULTIPLAYER-MENUS

1. Zurück Button zum Menü
2. Startet ein neues Spiel
3. Noch laufende Spiele, ausgegraute Spiele sind solche die momentan noch auf Antwort des Spielpartners warten
4. Historie der bereits gespielten Partien
5. Refresh Button

Neues Spiel

Alice möchte gerne gegen Bob spielen:

1. Alice sucht nach „Bob“ und fordert ihn heraus.
2. Alice kann dabei das erste Level frei wählen, und kann direkt den Level spielen. Dieser hat kein Zeitlimit.
3. Alice schliesst ihr Spiel nach 3:36 Minuten ab. Sie muss dabei nicht die optimale Lösung einreichen.
4. Bob bekommt nun die exakt gleiche Aufgabenstellung, für welche er maximal 3:36 Minuten (Alices Zeit) zur Verfügung hat. Wenn er schneller ist, hat das keine Auswirkungen auf die Bewertung, es zählt ausschliesslich die Anzahl Wechsel (Changeover) für die Endwertung.
5. Nachdem beide Spieler ihre Lösung eingegeben haben, findet der Showdown zwischen Alice, Bob und dem Solver statt.
6. Schritte 2 bis fünf werden 6 Mal wiederholt, die Rolle des Herausforderers wird aber abgewechselt.

Herausforderer - Herausgeforderten auswählen

Der Herausforderer kann den Herausgeforderten auswählen. Dieser kann eine beliebige oder eine spezifisch gesuchte Person sein.

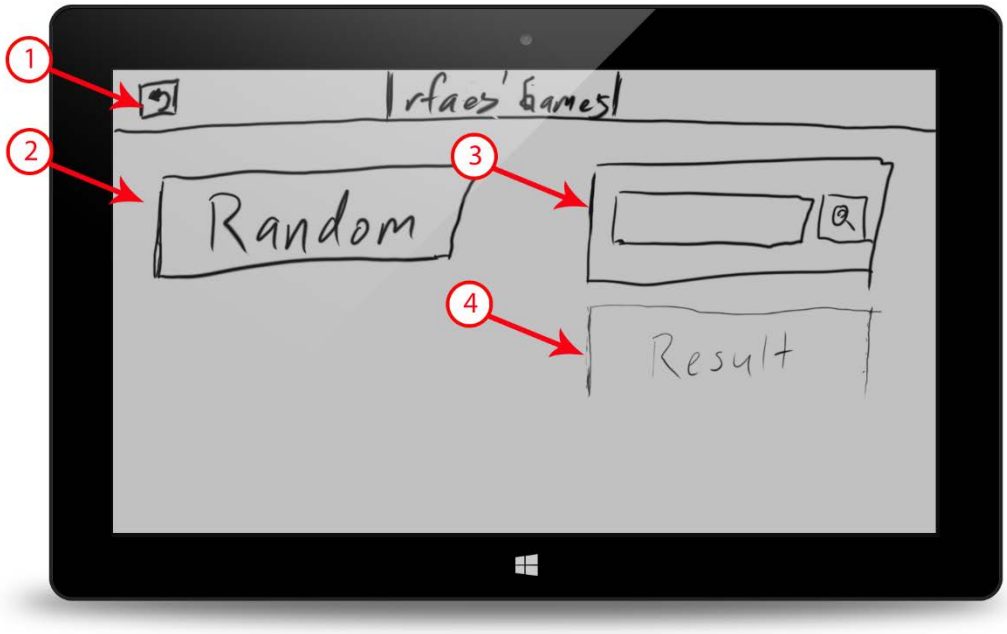


ABBILDUNG 18: MOCKUP DER GEGNER-SUCHE

1. Zurück Button zum Menü
2. Beliebigen Spieler auswählen
3. Spielersuchfeld
4. Resultate

Level Auswahl

Sobald der Gegner ausgesucht wurde, kann der zu spielende Level ausgesucht werden.

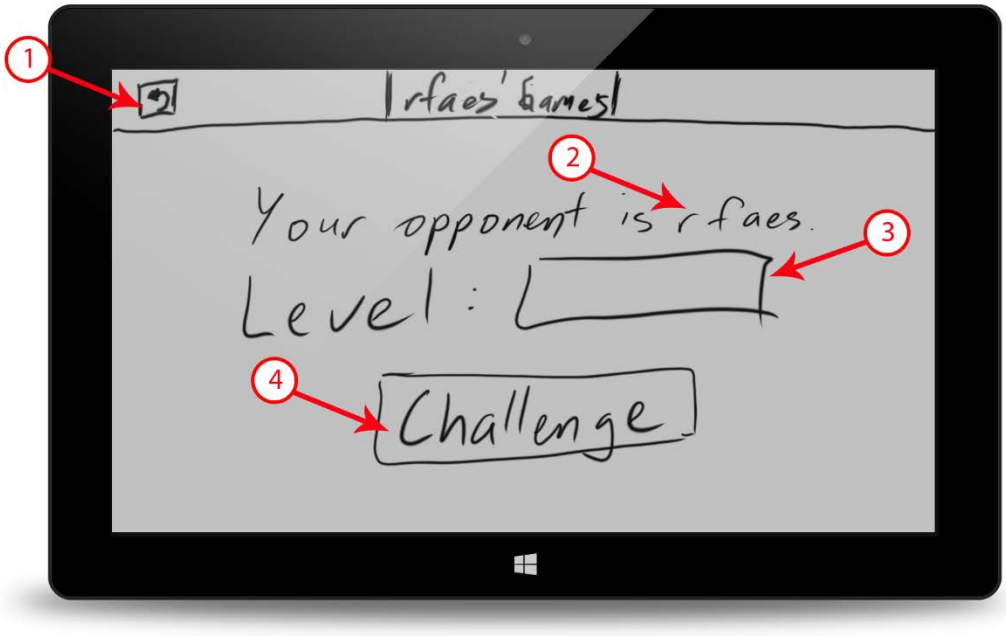


ABBILDUNG 19: AUSWAHL DES LEVELS DER ZU ERSTELLENDEN CHALLENGE

1. Zurück Button zur Gegnerauswahl
2. Gegner
3. Level Eingabefeld
4. Challenge Startknopf

Aufgabe lösen

Der Herausforderer kann die Zeit setzen, welche der Herausgeforderte einhalten muss. Hier kann der Herausforderer sich für eine der zwei möglichen Strategien entscheiden – entweder er löst das Level perfekt und braucht dafür mehr Zeit oder die Lösung ist nicht optimal, aber dafür wurde das Level in kürzer Zeit gelöst.

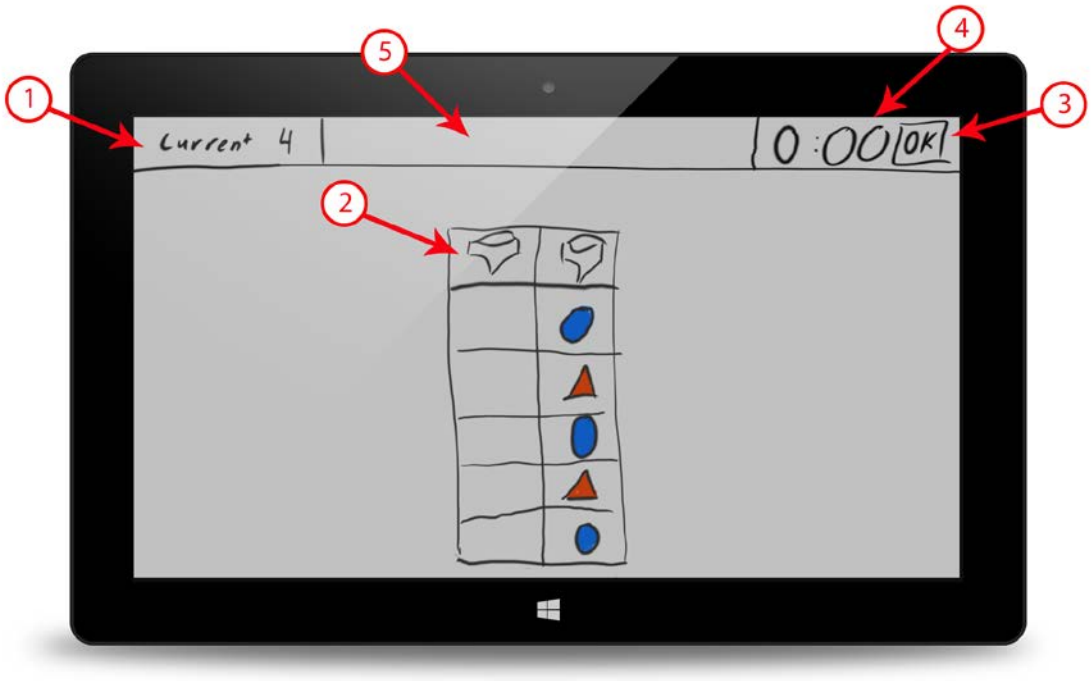


ABBILDUNG 20: MOCKUP DER CHALLENGER-ROLE IM SPIEL

1. Anzeige der momentanen Wechsel
2. Die zu lösende Aufgabe
3. OK-Button für das absenden der erstellten Lösung
4. Zeit welche aufwärts zählt
5. Bleibt leer, da keine Zeit abwärts gezählt werden muss

Aufgabe absenden

Sobald der Herausforderer mit seiner Lösung zufrieden ist, kann dieser die Lösung mit der benötigten Zeit absenden.

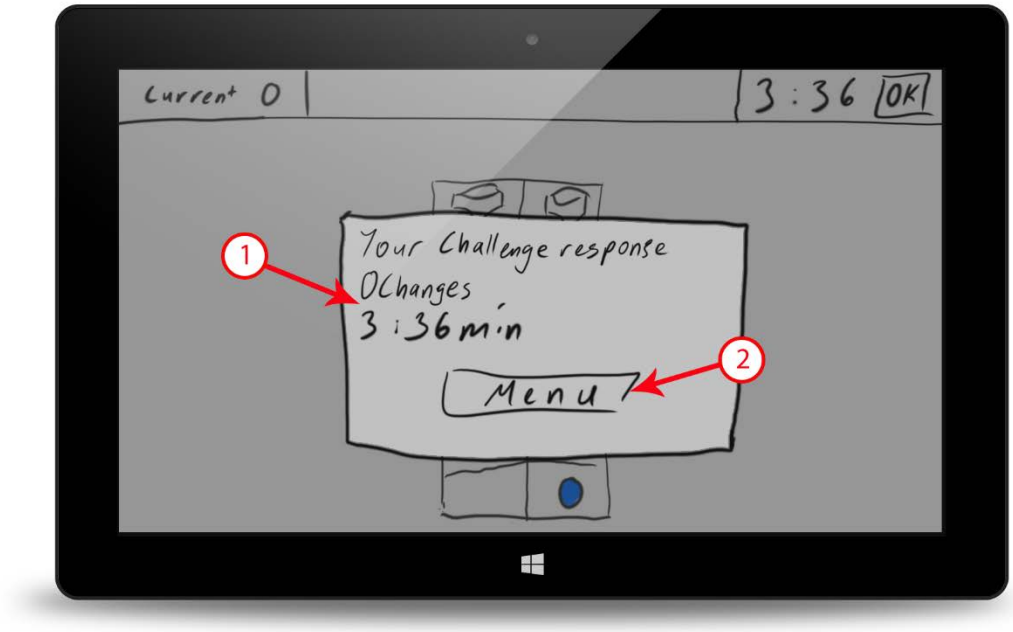


ABBILDUNG 21: MOCKUP DES VERSENDENS EINER CHALLENGE

1. Anzahl der benötigten Wechsel sowie die benötigte Zeit
2. Menü Knopf um in das Multiplayermenü zu gelangen

Herausgeforderter – Herausforderung

Der Herausgeforderte sieht in Multiplayermenü das neue Spiel. Sobald er dieses auswählt, wechselt der Screen auf die Zusammenfassung der Herausforderung.

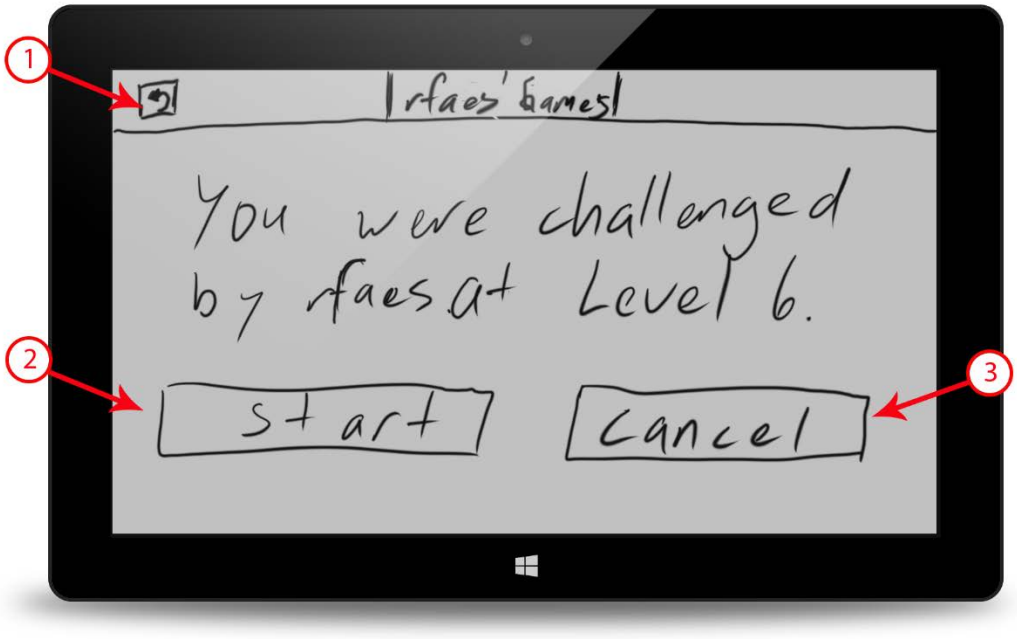


ABBILDUNG 22: ANNAHME DER CHALLENGE

1. Zurück Button zur Multiplayermenü
2. Starten der Herausforderung
3. Herausforderung ablehnen

Herausgeforderte – Lösen des Levels

Nun muss der Herausgeforderte das Level in gegebener Zeit möglichst gut lösen. Die Anzahl der Wechsel des Herausgeforderten sieht er nicht. Sobald die Zeit abgelaufen ist, oder der Herausforderer auf den Button OK drückt, kommt es zum Showdown.

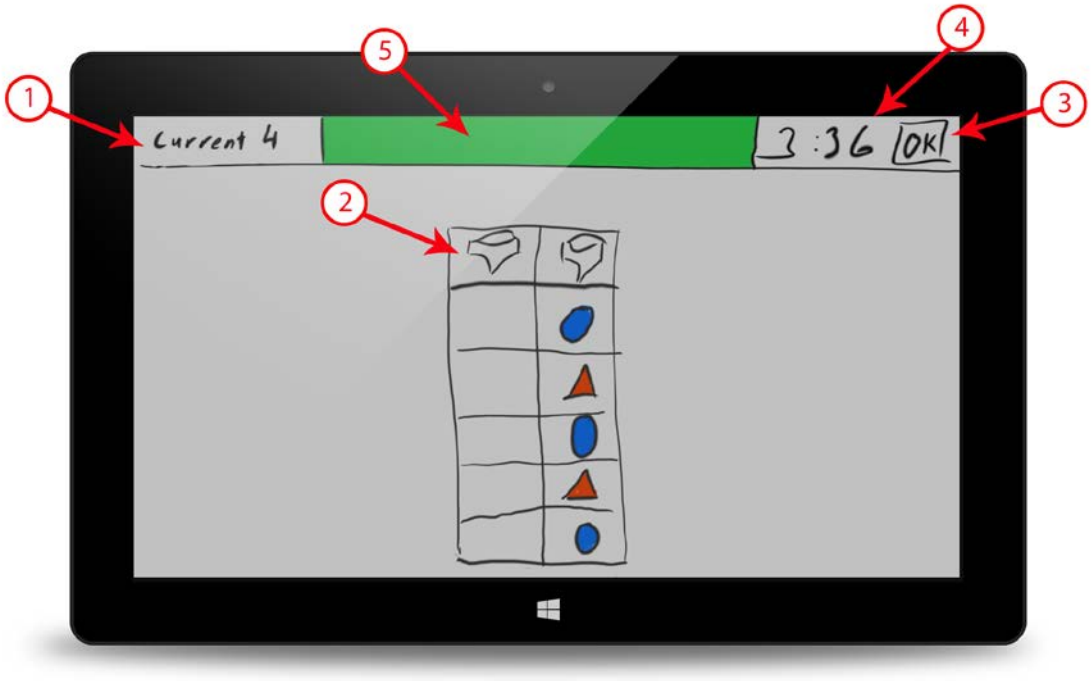


ABBILDUNG 23: MOCKUP DER CHALLENGEE-ROLE IM SPIEL

1. Anzeige der momentanen Wechsel
2. Die zu lösende Aufgabe
3. OK-Button für das absenden der erstellten Lösung
4. Noch verfügbare Zeit
5. Noch verfügbare Zeit in Form eines Balkens, Balken wird schmaler und ändert Farbe bei verstreichender Zeit

Showdown

Der Showdown wird jedes Mal angezeigt, sobald die Resultate für einen Level des Herausforderers und des Herausgefordertem vorliegen. Im Showdown werden die beiden Lösungen der Lösung des Solvers gegenüber gestellt.

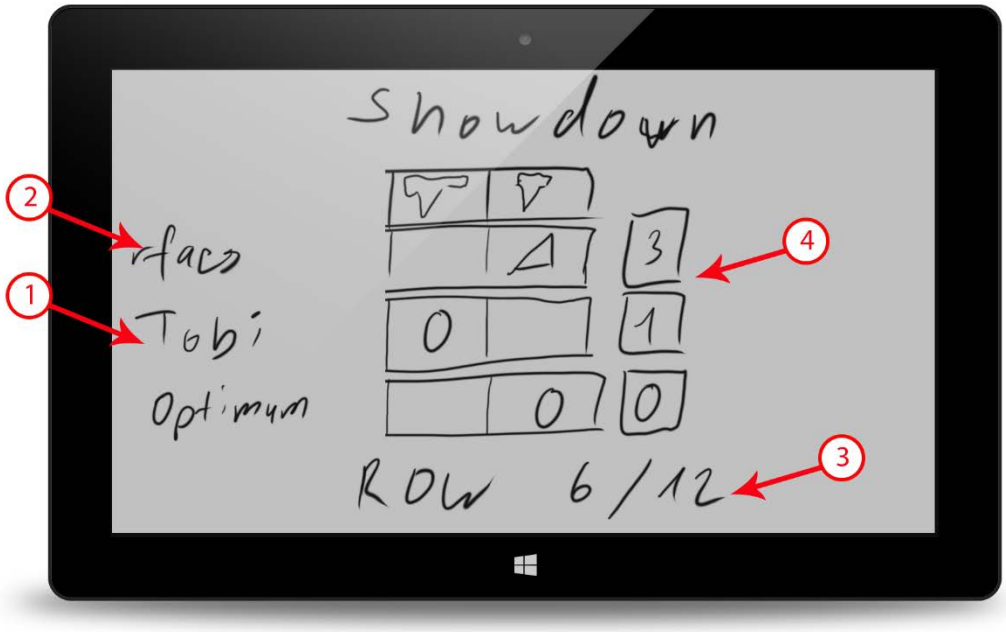


ABBILDUNG 24: MOCKUP DES SHOWDOWNS

1. Herausforderer
2. Herausgeforderter
3. Momentan angezeigte Zeile und die maximale Anzahl der Zeilen
4. Anzahl der Changeovers

Training

Der Spielmodus Training dient zum Kennenlernen der Spielweise. Die Levels sind statisch auf dem Gerät hinterlegt, es wird keine Internetverbindung benötigt.

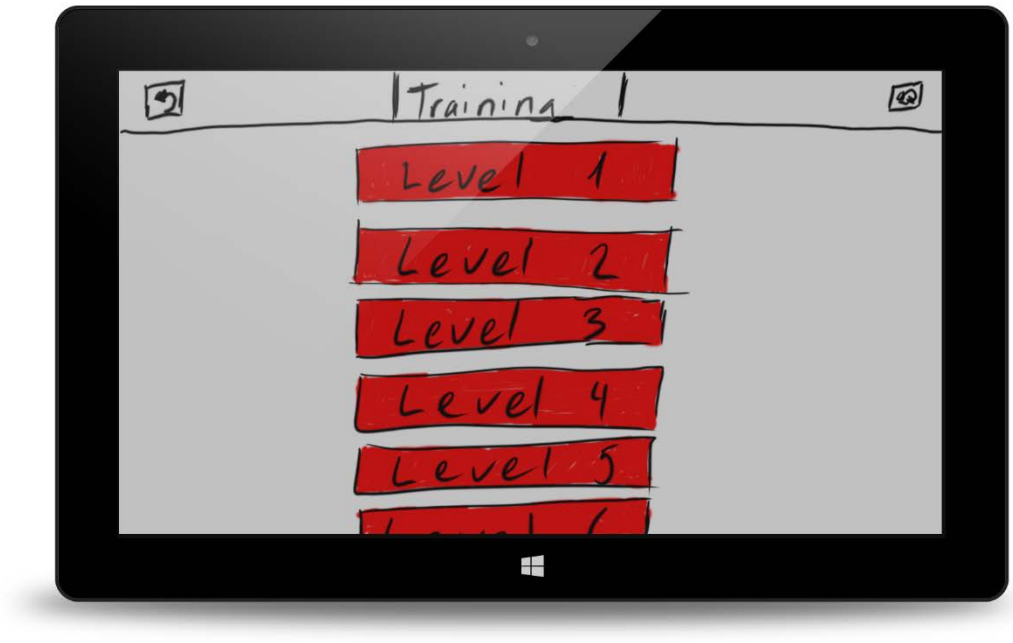


ABBILDUNG 25: LEVELAUSWAHL DES TRAININGSMODUS, AUCH DEMOMODUS GENANNT

6.4 Usability Tests

Startpunkt für die Szenarien ist jeweils die geöffnete Windows 8 Applikation. Als Gerät wird ein Microsoft Surface eingesetzt.

6.4.1 Testszenarien

Verstehen des Spielmechanik

Ziel

Der Benutzer soll ohne Anleitung erkennen mit welchen Userinterface Elementen der Benutzer interagieren kann und was das Ziel des Spiels ist.

Vorgehen

1. Touch auf „Demo“
2. Benutzer sieht die optimale und die momentan Anzahl von Changeovers und versteht das Ziel
3. Benutzer realisiert dass die Spielsteine verschoben werden können.
4. Touch auf Spielstein und durch ziehen des Steins in eine Zelle einer anderen Zeile woraufhin der Stein an die Ursprungsposition zurück springt.
5. Benutzer merkt, dass das Wechseln von Zeilen nicht möglich ist.
6. Benutzer spielt die ersten 4 Levels durch.

Erkennen der Spielzeit

Ziel

Der Spieler merkt, dass die Zeit abläuft.

Vorgehen

1. Touch auf Demo
2. Spieler spielt das Spiel.
3. Spieler nach dem Spielen von 2 Leveln fragen, ob im der Zeitbalken im oberen Bereich des Spiels aufgefallen ist.

Verschieben von Zeilen

Ziel

Der Benutzer merkt, dass neue Elemente vorhanden sind und dass er mit diesen interagieren kann.

Vorgehen

1. Touch auf „Demo“
2. Touch auf das Element neben einer Zeile und durch ziehen die Reihenfolge der Matrix verändern.

6.4.2 Durchführung

Nutzer

Nutzer 1 – Y

Y wohnt in Wagen St.Gallen und studiert an der HTW Chur.

An Schultagen entstehen täglich Reisezeiten von über zwei Stunden. Diese Zeit wird entweder zum Lernen oder zum Spielen verwendet. Als Spielgerät dient ein iPhone, da dieses immer dabei ist.

Y fällt somit in die Kategorie der Casual Gamer, welche weniger Zeit und Interesse am Spielen haben als Hardcore Gamer.

Nutzer 2 – Martin Plangger

Martin Plangger wohnt in Mannheim Deutschland und studiert in Mannheim Chemie.

Da die Schule in Deutschland nur bis Nachmittags geht, sind genügend Stunden für das Spielen vorhanden. Er spielt verschiedene MMORPG am Computer oder verschiedene Spiele auf Spielkonsolen.

Martin Plangger ist ein Hardcore Gamer, welcher mehrere Stunden täglich mit Spielen verbringt.

Instruktionen für Usability Test

Testumgebung

Die Probanden erhalten ein Microsoft Surface mit Windows 8.1 RT. Auf dem Gerät ist das Spiel SCS Optimizer installiert. Den Probanden werden Aufgaben gestellt, welche sie erfüllen sollen.

Die Applikation SCS Optimizer ist jeweils gestartet und offen.

Testbedingungen

Den Probanden wird mitgeteilt sobald diese mit dem Test beginnen können. Sollte für ein Testszenario längere Zeit benötigt werden, können Fragen an die Betreuer des Tests gestellt werden.

Kontext

Der Benutzer hat Lust auf ein spannendes Spiel um sich zu entspannen.

Durchführung

Aufgabe 1 – Testperson 1

Benutzer hat geöffnete Applikation vor sich

Benutzer wählt Demo Modus aus

Benutzer versteht nicht was Changeovers sind und wie diese gezählt werden

Betreuer erklärt was und wie die Changeovers gezählt werden.

Benutzer versteht das Ziel nicht.

Betreuer erklärt das Ziel nochmals und löst gleichzeitig den ersten Level zusammen mit dem Benutzer.

Benutzer spielt den Level 2 selbständig. Und versucht weiterhin die Zeile der Spielsteine zu verschieben, was nicht funktioniert.

Benutzer versteht nicht warum dies nicht möglich ist, denn die Steine können an die entsprechende Position verschoben werden.

Aufgabe 1 – Testperson 2

Benutzer hat geöffnete Applikation vor sich

Benutzer wählt Demo Modus aus

Benutzer versteht nicht was Changeovers sind und wie diese Zustande kommen

Betreuer erklärt was und wie die Changeovers gezählt werden.

Benutzer versteht das Ziel und verschiebt die Steine.

Aufgabe 2 – Testperson 1

Benutzer spielt die ersten 4 Levels.

Betreuer fragt ob dem Benutzer die Zeitbalken aufgefallen sind, was der Benutzer verneint.

Aufgabe 2 – Testperson 2

Benutzer spielt die ersten 4 Levels.

Betreuer fragt ob dem Benutzer die Zeitbalken aufgefallen sind, woraufhin der Benutzer dies bejaht mit dem Kommentar „Ja klar, da die Anzahl von Zügen egal ist, muss etwas anderes wichtig sein, die Zeit“.

Aufgabe 3 – Testperson 1

Benutzer spielt den Level und versucht die optimale Anzahl von Changeovers zu erreichen. Als dies nach einiger Zeit nicht gelingt, fragt der Benutzer den Betreuer ob der Level überhaupt lösbar ist. Daraufhin wird der Benutzer auf die Elemente neben der Zeile hingewiesen mit dem Kommentar dass damit die Zeilen verschoben werden können.

Benutzer klickt auf die Pfeile des Zeilenverschiebers, woraufhin nichts geschieht.

Der Benutzer fragt den Betreuer wie er den nun die Zeilen verschieben können. Der Betreuer erklärt die Funktionsweise.

Der Benutzer löst den Level

Aufgabe 3 – Testperson 2

Dem Benutzer fallen die Elemente neben der Zeile auf und drückt auf die Pfeile woraufhin nichts geschieht. Der Benutzer fragt den Betreuer was das sei. Der Betreuer erklärt, dass der Benutzer damit die Zeilen mit dem Finger verschieben könne.

Der Benutzer verschiebt die Zeile mit dem Finger und löst den Level

Zusammenfassung der Probleme

Die Spielmechanik ist nicht ohne Erklärung dem Benutzer verständlich.

Die beiden Pfeile des Zeilenverschiebers verwirren die Benutzer. Hier muss ein anderes Symbol gesucht werden. Hier wurde die Regel „Consistency and standards“ von Nielsen (3) verletzt. Dies kann gelöst werden, indem anstelle von Pfeile die üblichen 3 horizontalen Striche verwendet werden.

Dass die Steine bei einem ungültigen Verschieben direkt zurück springen reicht nicht aus damit die Benutzer verstehen, dass die Steine nur horizontal verschoben werden können.

Folgerungen aus den Problemen

Die beiden Pfeile der Zeilenverschieber müssen durch die üblichen horizontalen Striche ersetzt werden.

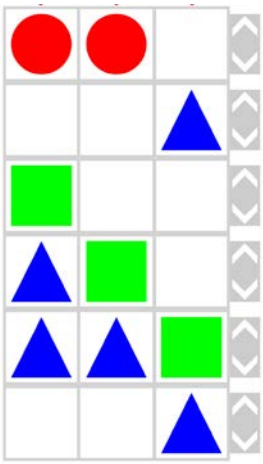


ABBILDUNG 26: SPIELMATRIX MIT VERSCHIEBBAREN ROWS

Die Tokens sollten sich am Anfang des Spiels kurz bewegen, damit der Benutzer merkt, dass die Spielsteine verschoben werden können.

Die Anzahl der Changeovers in einer Spalte anzeigen.

7 Umsetzung

Die Umsetzung ist in drei Teile gegliedert:

- Common: Aspekte der Software, welche das gesamte System, im speziellen die Kommunikation, betreffen.
- Client: Die (mobile) Spiel-App
- Server: Persistierung, Cloud, Administration

7.1 Common

7.1.1 Lizenzen

Produkt	Lizenz
Xamarin	Lizenzpflichtig
MvvmCross	Microsoft Public License (MS-PL)
RavenDB	Lizenzpflichtig für kommerzielle Nutzung
Spielmusik - 8bit Dungeon Level	Creative Commons: Attribution 3.0
Grafiken	Creative Commons: Attribution-NoDerivs 3.0
ASP.NET MVC 5	Apache License 2.0
ASP.NET Web API 2.1	Apache License 2.0
Bootstrap	MIT License
jQuery	MIT License
nvd3.js	Apache License 2.0
d3.js	Copyright (c) 2012, Michael Bostock All rights reserved. Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met: • Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer. • Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution. • The name Michael Bostock may not be used to endorse or promote products derived from this software without specific prior written permission.

7.1.2 Models

Nachdem das Spielkonzept ausgearbeitet, die Komponenten und der Ablauf des Spiels definiert wurden, konnte das Domain Model zum System erstellt werden. Es dient als Diskussionsgrundlage mit dem Auftraggeber und Betreuer der Arbeit, sowie als Grundlage für das Klassendiagramm und der RESTful-Schnittstelle.

Statische Objekte – Daten, welche sich weniger häufig verändern, z.B. Stammdaten – sind in Abbildung 27 unten links angeordnet. Das heisst, je weiter sich eine Klasse in der oberen rechten Ecke befindet, desto mehr wird diese zur Laufzeit des System mutiert. Speziell zu beachten ist hierbei die **Matrix**-Klasse, die in der Klasse Turn (startMatrix, optimalMatrix) und UserSolution (userMatrix) vorkommt. Per Definition besitzt das Value Object keine Identität weist aber eine komplexere Struktur als ein einfacher Datentyp auf.

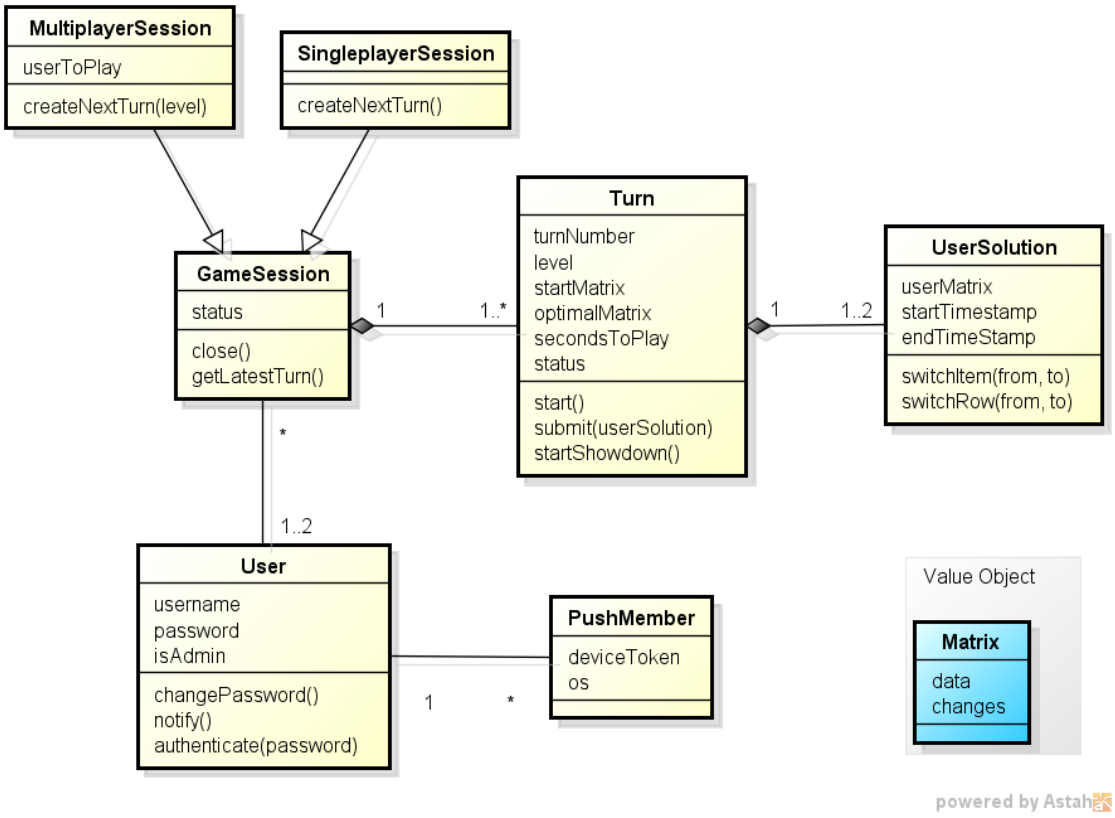


ABBILDUNG 27: DOMAIN MODEL

Die Klasse **User** und **PushMember** werden als Stammdaten der Applikation betrachtet. In PushMember können für einen Benutzer mehrere Geräte eingetragen werden, die dann als Push-Notification eine Nachricht erhalten, dass sie mit dem Spielzug an der Reihe sind. Die Klasse User dient sowohl innerhalb der Infrastruktur als Quelle für die Authorisierung der Services, als auch der Zugehörigkeit eines Benutzers zu seinen Spielen (GameSession).

Eine **GameSession** kann als Single- oder Multiplayer auftreten und fasst mehrere Runden zusammen. Im Multiplayer-Modus treten zwei Spieler Head-To-Head gegeneinander an. Zusätzlich unterscheidet sich die MultiplayerSession darin, dass für die nächste Runde (Turn) das Level in *CreateNextTurn()* angegeben werden muss, da dieser im Gegensatz zum Singleplayer vom Spieler bestimmt werden kann.

Der **Turn** könnte auch als „Aufgabenstellung“ oder „Problem“ bezeichnet werden. Das Attribut *startMatrix* enthält dabei die Aufgabenstellung und *optimalMatrix* die Lösung des Solvers. Je höher das *level*, desto komplexer ist die zu lösende Matrix.

Zu jedem Turn werden die Lösungen als **UserSolution** abgebildet. Falls man sich im Multiplayer-Modus befindet, wird für jeden User je eine Solution erstellt. Manipulationen auf der Matrix während des Spiels werden in der UserSolution über die *switchItem()* und *switchRow()* Methoden durchgeführt.

7.1.3 Kommunikationsablauf für Singleplayer-Modus

Die einzelnen Schritte aus Abbildung 28 werden in der Tabelle unterhalb beschrieben.

Schritt	Methode	Beschreibung
1	CreateSingleplayerSession()	Neue Singleplayer-Session wird erstellt, falls bereits eine Offene besteht, wird diese defaultmässig überschrieben.
2	GetSingleplayerSession()	Die aktuelle (Status = Open) Singleplayer-Session wird zurück gegeben. Es kann nur eine offene Singleplayer-Session pro Benutzer existieren.
3	CreateNextTurn()	Die nächste Runde der aktuellen Session wird erstellt. Silent-Fail falls eine offene Runde (Turn) bereits existiert.
4	GetLatestTurn()	Gibt die letzte offene Runde zurück.
5	Start()	Startet das Spiel lokal auf dem Client und übermittelt den aktuellen TimeStamp an den Server.
6	Submit()	Überträgt die eigene Lösung. Der Server schliesst die Session, falls die Optimierung schlechter wie die vom Solver ist (=> 7: turn failed). Sind die Turn-Bedingungen erfüllt, kann der nächste erstellt werden (=> 3: CreateNextTurn)
7	ShowOptimizerSolution()	Zeigt lokal die Lösung vom Optimizer-Algorithmus aus der Cloud an.
8	Close()	Schliesst die Session lokal ab.

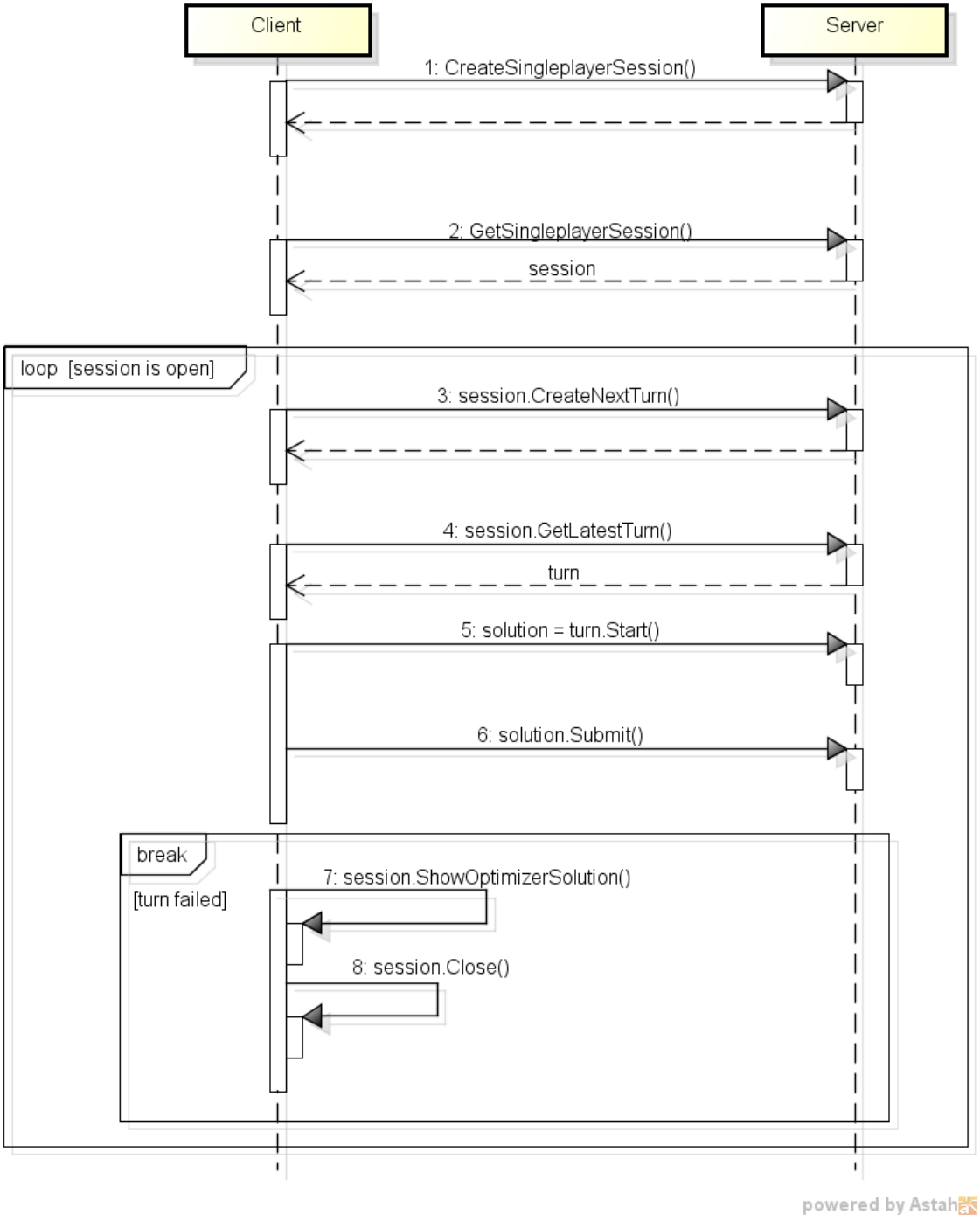


ABBILDUNG 28: SEQUENZDIAGRAMM FÜR SINGLEPLAYER-KOMMUNIKATION

7.1.4 Kommunikationsablauf für Multiplayer-Modus

Die einzelnen Schritte aus Abbildung 28 werden in der Tabelle unterhalb beschrieben.

Schritt	Methode	Beschreibung
1	GetMultiplayerSessions(userID)	Holt sich sämtliche Spiele (ohne Detail-Informationen) des aktuellen Benutzers für die Anzeige im Menu.
2	Get(sessionId)	Benutzer wählt lokal ein Spiel aus. (Keine neue Session)
3	CreateNextTurn(level)	Der Herausforderer bestimmt das Level des nächsten Turns. Server erstellt anschliessend zufällig generierte Matrix mit definiertem Schwierigkeitsgrad (Level). Der Schritt wird ausgelassen, falls man sich in der Rolle des Challengee befindet (siehe Kapitel 5.3.1 Play Challengee Level).
4	GetLatestTurn()	Der Spieler erhält den aktuellen Turn zu seiner Session.
5	Start()	Das Spiel wird lokal gestartet und der aktuelle TimeStamp an den Server übermittelt.
6	Submit()	Die Lösung (Matrix) wird an den Server übermittelt.

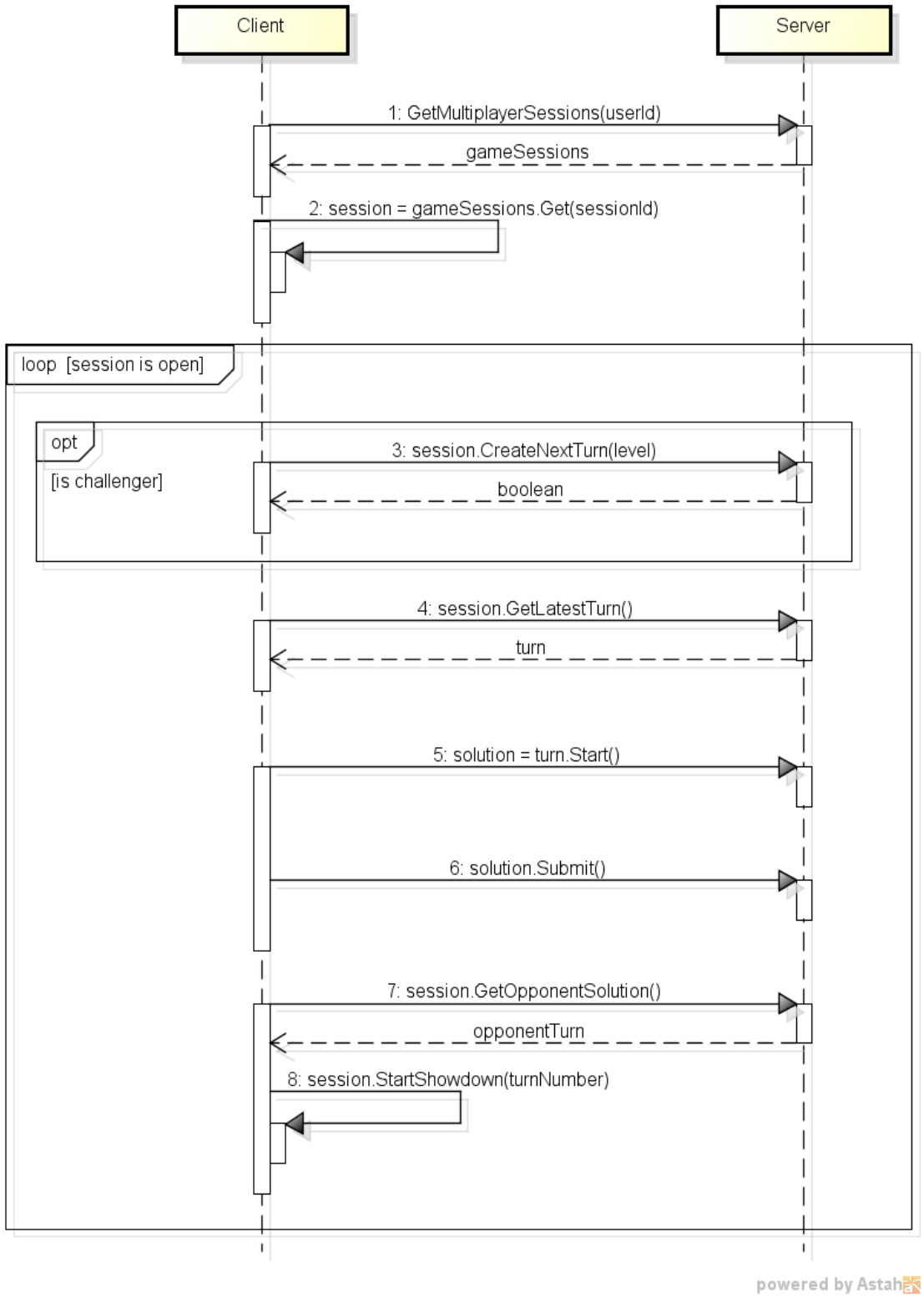


ABBILDUNG 29: SEQUENZ-DIAGRAMM FÜR MULTIPLAYER-KOMMUNIKATION

7.1.5 Authentifizierung über Basic Authentication

Da die Verbindung zwischen Client und Server mittels TLS (Server-Zertifikat) verschlüsselt wird, können die Credentials uncheffriert im HTTP-Header nach folgendem Muster hinzugefügt werden:

```
Authorization: Basic Base64(<username>:<password>)
```

Username und Passwort warden mit Doppelpunkt getrennt und mittels Base64 kodiert. Als Beispiel mit *Username = scs und passwort = pwd*:

```
Authorization: Basic c2NzOnB3ZA==
```

7.1.6 Onion Architecture

Der Client und der Server sind nach der „Onion Architecture“, welche von Jeffrey Palermo erläutert wurde (4), aufgebaut. Diese Architektur ist in verschiedene Schalen aufgeteilt, das Domain Model steht dabei immer im Zentrum der Architektur, da die Domain Logik das langlebigste der Applikation sein sollte. Die äusseren Schalen in der „Onion Architecture“ dürfen nur auf dieselbe Schale oder die Schalen in Richtung Kern verwenden.

Die „Onion Architecture“ ist auch unter anderen Namen bekannt, wie Hexagonale Architektur. Das Prinzip ist bei allen gleich, die äusseren Schichten dürfen nur auf die gleiche Schicht oder auf die Inneren zugreifen. Die Architektur verwendet das „Inversion of Control“-Pattern, welches vorsieht, dass die inneren Schalen die Funktionalität der äusseren Schalen mit Interfaces definiert. Die inneren Schalen greifen nie auf die Implementierungen sondern ausschliesslich auf die Interfaces zu. Die Implementierungen werden mithilfe von (Constructor-) Injection zur Verfügung gestellt.

7.1.7 Anzahl von Visual Studio Projekte

Ein Visual Studio Projekt ist als „Deployment Unit“ (Auslieferungseinheit) zu betrachten (5), was zur Folge hat, dass jeweils ein Client und ein Server Projekt vorhanden ist. Aufgrund der plattformunabhängigen Realisierung durch Xamarin, muss der Client in ein Core Projekt, welche die Logik der Applikation beinhaltet, und ein plattformspezifisches Projekt für das Userinterface beinhaltet. Da während dieser Arbeit eine Windows Store App sowie eine Android Applikation sind zusätzlich zum Core Projekt noch zwei weitere Projekte vorhanden, was zu einer Gesamtzahl von 3 Projekten seitens Client führt.

7.1.8 JSON Array

Der ECMA-404 Standard (6) sieht eine Übertragung von Arrays vor, welche wie folgt aussieht:

```
[
    {"Firstname":"John", "Lastname":"Doe"},
    {"Firstname":"Anna", "Lastname":"Smith"},
    {"Firstname":"Peter", "Lastname": "Jones"}
]
```

Dies funktioniert bei Listen mit Inhalt, doch sobald eine leere Liste zu JSON umgewandelt wird, würde das Ergebnis folgendermassen aussehen:

```
[]
```

Dies führt bei der in dieser Bachelorarbeit verwendet Version der JSON.Net-Library zu einem Fehler. Die Library erwartet ein Key-Value Paar, welches aber in dem gegebenen Beispiel nicht vorhanden ist.

Daher wird nicht direkt eine Liste, sondern ein Objekt, welches die Liste als Property hält übertragen. Das Ganze sieht folgendermassen aus:

```
{"People":[] }
```

Durch die Kapslung der Liste erübrigt sich die Überprüfung ob die Liste leer ist.

7.1.9 Mehrdimensionales Array anstelle von Jagged Array

Für das Übertragen und Persistieren der Matrix wird ein zweidimensionales Array verwendet anstelle eines Jagged Arrays, welches ein Array ist und für jede Zeile ein Array beinhaltet. Die Vorteile des zweidimensionalen Arrays sind das die Überprüfung der Längen der einzelnen Zeilen entfällt und das die Speicherallozierung auf einmal geschehen kann, was sich positiv auf die Perfomance auswirkt.

7.2 Client

7.2.1 Persistenz Client

Der Client persistiert nur Benutzername, Passwort und Spieleinstellungen, wie zum Beispiel Lautstärke, dies erspart eine Synchronisationslogik welche den Rahmen dieser Bachelorarbeit gesprengt hätte.

7.2.2 Error Handling REST-Client

Der während dieser Arbeit geschriebene REST-Client wandelt HTTP-Fehlercodes in Exceptions um. Diese werden an den Aufrufer der Funktionen weitergeleitet, so dass der Aufrufer je nach Methode unterschiedlich reagieren kann.

7.3 Server

7.3.1 RavenDB/Aggregates

Wie in der Domain-Modellierung festgestellt wurde, benötigt das System relativ wenige Klassen, welche sich gut in eine kleine Anzahl von Aggregation zusammen fassen lassen.

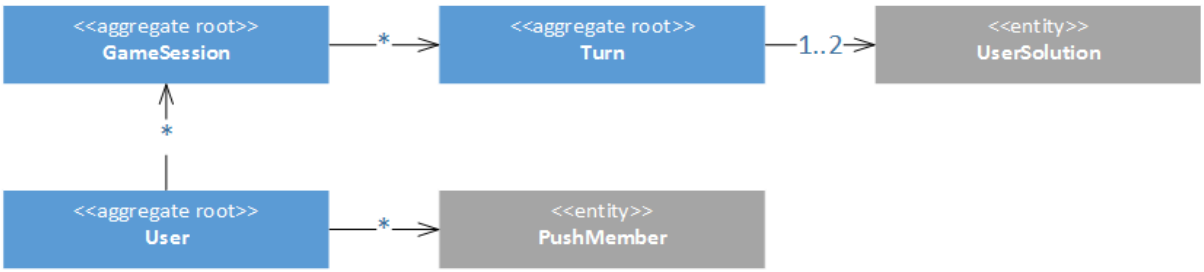


ABBILDUNG 30: AGGREGATES FÜR DIE PERSISTIERUNG DER DATEN AUF DEM SERVER

Einfache Relationen zwischen den Klassen/Objekten legen eine Dokumentenbasierte Datenbank nahe. Ein positiver Effekt ist dabei die Performance, da NoSQL-DBs als sehr schnell gelten und im Projekt auf das Entity Framwork verzichtet werden kann. Der Maintenance-Aufwand gegenüber einer relationalen Datenbank, wie MSSQL, ist ebenfalls geringer. RavenDB ist in C# programmiert und bietet einfache Schnittstellen, die dem Entity Framework nachempfunden sind und somit für .NET-Entwickler nach kurzer Einarbeitungszeit vertraut sind. Ausserdem bietet RavenDB praktische Features, wie die Unterstützung für *System.Linq.IQueryable* innerhalb von Abfragen und *System.Transactions.TransactionScope* um auf höheren Layers Transaktionen zu gliedern.

7.3.2 Microsoft Azure

Der Solver der SCS ist eine nativ 32 Bit Applikation, was zur Folge hat, dass die Applikation direkt auf der Hardware läuft; die Hardware muss zwingend 32 Bit Applikationen unterstützen. Microsoft Azure bietet verschiedene Services von „Software as a Service“ bis hin zu „Infrastructure as a Service“ an. Je nach Service ist der Administrationsaufwand für die SCS unterschiedlich gross, was sich auf die betrieblichen Kosten auswirkt. Aus diesem Grund, wird „Platform as a service“ als zu bevorzugenden Service angesehen, da dort der Aufwand für das Warten des Betriebssystems entfällt.

Microsoft Azure unterstützt native Applikationen, sofern diese in 64 Bit kompiliert wurden[1]. Da für den Solver nicht alle Libraries in 64 Bit vorhanden sind, kann dieser nur in 32 Bit kompiliert werden. Dies hat zur Folge, dass „Platform as a service“ als möglicher Service-Art entfällt und nur noch „Infrastructure as a service“ infrage kommt, da dort eine virtuelle Maschine ohne Einschränkungen verwendet werden kann.

[1] http://msdn.microsoft.com/en-us/library/hh694038.aspx

7.3.3 Matrix-Generatoren

Um unsere Vision einer hohen Wiederspielbarkeit zu realisieren, benötigt das Spiel Aufgaben, welche bei jedem Durchgang neue Herausforderungen bieten. Maximale Abwechslung kann hierbei mit zufällig generierten Levels, bzw. Matrizen, erreicht werden. Die Umsetzung dieser Idee fordert aber die Antwort auf eine entscheidende Frage:

> Wie wird der Schwierigkeitsgrad einer Aufgabe definiert?

Ein Level auf Stufe 8 sollte bei jedem Durchgang auch ähnlich schwer ausfallen. Aus dem Spielprinzip wird klar ersichtlich, dass die Grösse der Matrix eine entscheidende Rolle spielt: Je grösser die Matrix, desto schwieriger ist prinzipiell die Aufgabe. Aber auch die Anzahl unterschiedlicher Elemente oder leerer Felder haben einen Einfluss auf das Herausforderungspotential. Bei einem kurzen Selbstversuch wurde herausgefunden, dass die Umstellung von fixen nach variablen Reihen einen grossen Sprung in der Komplexität verursacht. Es wird schnell klar, dass viele Variablen den Schwierigkeitsgrad beeinflussen und die Frage nur anhand von experimentellen Daten geklärt werden kann.

Eine spezielle Rolle spielt der Faktor „Zeit“. Auf einfache Weise können die verfügbaren Sekunden soweit beschränkt werden, dass dem Spieler diese davon laufen. Nur eine Beschränkung der vorgegebenen Zeit ist aber sehr kontraproduktiv für ein angenehmes Spielerlebnis, da der Benutzer sich bevormundet fühlt, wenn das Spiel im die Aufgabe vor der Nase wegnimmt. Für das optimale Erfolgserlebnis sollte der Benutzer also nur knapp mehr Zeit zur Verfügung haben als er benötigt. Aus diesem Grund wird in den Experimenten die Zeit ausgeschlossen und im Nachhinein den Ergebnissen angepasst.

Optimal wäre, wenn man die erwähnten Variablen, welche den Schwierigkeitsgrad definieren, unabhängig voneinander quantifizieren könnte. Dies würde aber den Rahmen der Arbeit sprengen, da sehr viele Experimente mit unterschiedlichsten Probanden durchgeführt werden müsste. Als Alternative stellen wir deshalb drei Algorithmen für das Generieren der Matrizen zur Verfügung. Der Versuch ist folgendermassen Aufgebaut:

- Das Zeitlimit für die Aufgaben wird im Back-End auf „Unlimited“ gesetzt.
- Probanden versuchen die Aufgaben so schnell wie möglich zu lösen. Dabei wechseln sie mit den Algorithmen ab.

Anschliessend wertet man die durchschnittliche Zeit der Levels (Schwierigkeitsgrade) aus und stellt diese als Kurve dar. In ersten Durchgängen wurde schnell festgestellt dass die benötigte Zeit der Probanden einer exponentiellen Kurve gleicht.

Die Formel zur Festlegung der Rundenzeit (t_R) ist deshalb wie folgt definiert:

$$t_R = t_{start} + t_{base}^{level}$$

Definieren wir $t_{start} = 10$ und $t_{base} = 1.5$, steht im 9. Level folgende Zeit zur Lösung der Aufgabenstellung zur Verfügung:

$$t_R = 10 + 1.5^9 \approx \underline{48 \text{ Sekunden}}$$

Die Parameter t_{start} und t_{base} können im Back-End gesetzt werden.

Die wichtigsten Unterschiede der erwähnten Algorithmen werden in folgender Tabelle erläutert:

Generatorname	Beschreibung
BasicTryAndError	Für jedes Feld der Matrix wird eine Zufallszahl zwischen -2 und 4 (in höheren Levels zwischen -2 und 6) generiert. Felder mit den Werten unter 0 werden leer gelassen. Positive Werte und die Zahl 0 stellen die unterschiedlichen Farben dar. Anschliessend wird die Matrix vom Solver optimiert und überprüft, ob die Aufgabenstellung einen höheren Changeover-Wert als die Optimierte besitzt. Falls nicht, wird nochmals von vorne begonnen.
Incremental	Umgekehrte Variante des BasicTryAndError-Algorithmus: Nach einer unter Rahmenbedingungen stehender, zufälligen Befüllung der Matrix, wird diese zuerst optimiert, anschliessend inkrementell verändert und jeweils überprüft, ob sich die Anzahl an Changeovers erhöht hat. Das Ergebnis ist die Aufgabenstellung für den Spieler mit einer hohen Anzahl an Changeovers. Der Algorithmus stellt ausserdem sicher, dass die definierte Anzahl von unterschiedlichen Farben in der Matrix vorkommen (im Gegensatz zum BasicTryAndError). Das Verhältnis von leeren zu gefüllten (farbigen) Feldern weicht dabei im gleichen Level über mehrere Durchgänge maximal 10% voneinander ab. Mit dieser Massnahme wird angestrebt, dass gleiche Levels möglichst einen ähnlichen Schwierigkeitsgrad besitzen.
MaxChanges	Für jede Zeile der Matrix wird eine zufällige Anzahl von Spielsteinen gelegt. Es wird darauf geachtet, dass mindestens ein Spielstein pro Zeile vorhanden ist. Die Anzahl der unterschiedlichen Spielsteine erhöht sich jeden zweiten Level um Eins. Bei der Platzierung der Spielsteine wird darauf geachtet, dass der Spielstein in der oberen Zeile von einer anderen Art ist, so wird eine maximale Anzahl von Farbwechseln erzeugt.

Alle Algorithmen haben gemeinsam, dass sich die Matrix-Dimension, Anzahl Elemente, sowie Anzahl unterschiedlicher Farben mit steigendem Level prinzipiell erhöhen. Ab einem gewissen Level wird jeweils das verschieben der Rows ermöglicht, was den Schwierigkeitsgrad weiter steigert.

Die Implementierung ist unter dem Namespace

Scs.Optimizer.Server.Infrastructure.MatrixGenerators

zu finden und als Strategy pattern in den *SingleplayerSession*- und *MultiplayerSession*-Klassen eingebettet. Die Auswahl des gewünschten Algorithmus und der Zeitformel wird der Firma SCS überlassen. Als Hilfe wird ihnen ein Exceldokument zur Abschätzung der Zeitformel abgegeben. Die Daten mit den Levelzeiten können dabei über den ReportController (siehe D. RESTful-Schnittstelle) als CSV heruntergeladen werden.

	A	B	D	E	G	H
1	**Data**		**Formula**			
2	**Level**	**Seconds Used**	**Level**	**Time Formula**	**Parameters**	
3	1	1	1	7.65	a=	5
4	1	2	2	11.7045	b=	1.53
5	1	2	3	17.907885		
6	1	1	4	27.39906405		
7	1	2	5	41.920568		
8	1	3	6	64.13846903		
9	1	2	7	98.13185762		
10	2	5	8	150.1417422		
11	2	6	9	229.7168655		
12	2	2	10	351.4668042		
13	2	2	11	537.7442105		
14	3	6	12	822.748642		
15	3	3	13	1258.805422		
16	3	4	14	1925.972296		
17	3	3	15	2946.737613		
18	4	9	16	4508.508548		
19	4	14	17	6898.018078		
20	4	8	18	10553.96766		
21	5	10	19	16147.57052		
22	5	36	20	24705.7829		
23	5	21	21	37799.84783		

ABBILDUNG 31: DATENBEREICH DES EXCEL-SHEETS FÜR DIE OPTIMALE ZEIT-FORMEL

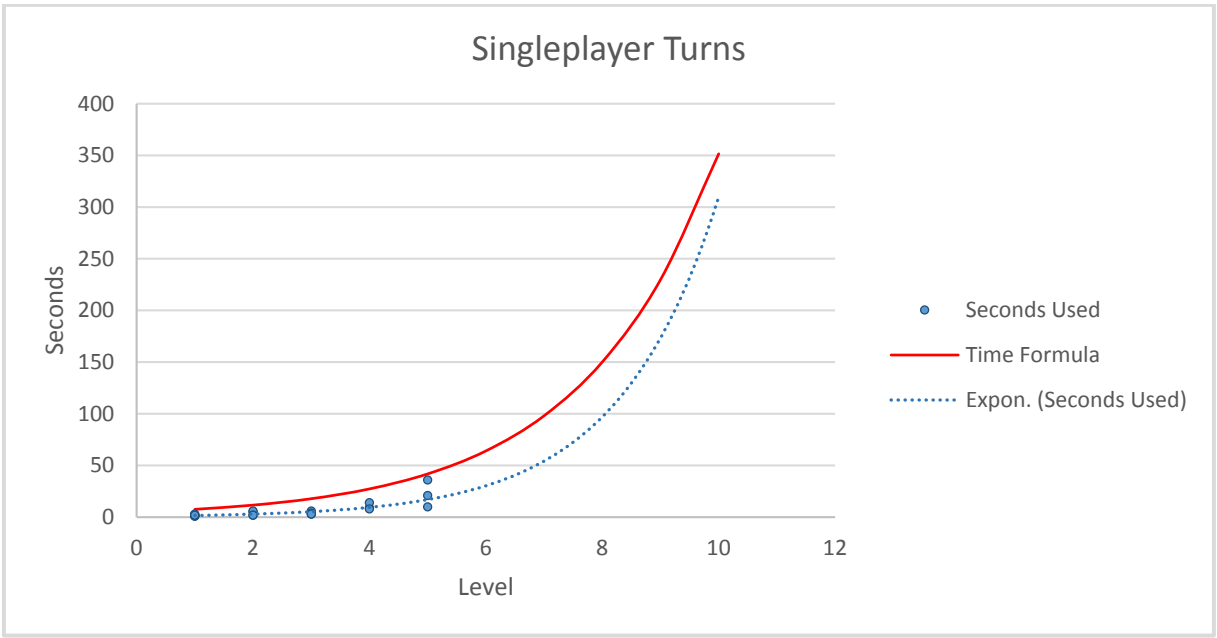


ABBILDUNG 32: EXCEL-SHEET – VISUALISIERUNG DER ZEITFORMEL GEGENÜBER DEN LEVEL-ZEITEN INKL. TREND

7.3.4 Administrations-Back-End

Das Administratrions-UI ist lokal per default unter http://localhost:24605 verfügbar.

Zur Übersicht können Statistiken zu den Single- und Multiplayer-Spielen (siehe Abbildung 34) angesehen werden. Die Authentifizierung der Admin-Seite basiert auf den Optimizer-App-Usern in der RavenDB. Das Property *IsAdmin* authorisiert den Benutzer für den Zugriff auf das Back-End.

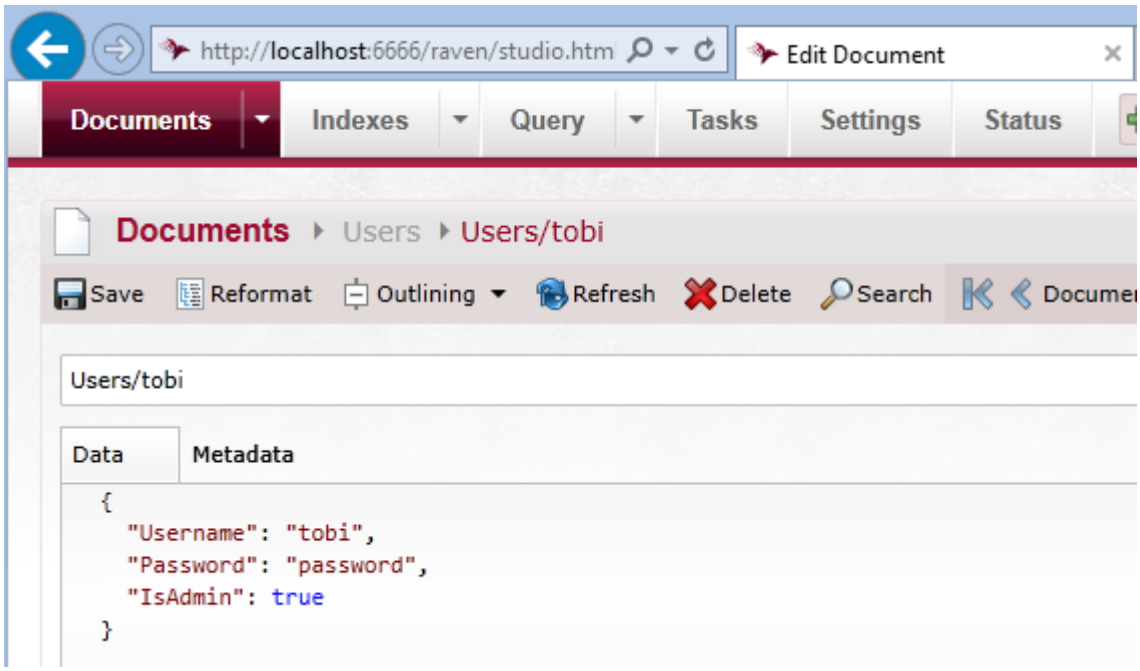


ABBILDUNG 33: USER-EINTRAG IN RAVENDB-SILVERLIGHT-CONSOLE

In der Umgebung der Bachelorarbeit wurde der Port der RavenDB-Console explizit auf 6666 gesetzt, da der Default Port (8080) bereits durch den IIS belegt ist.

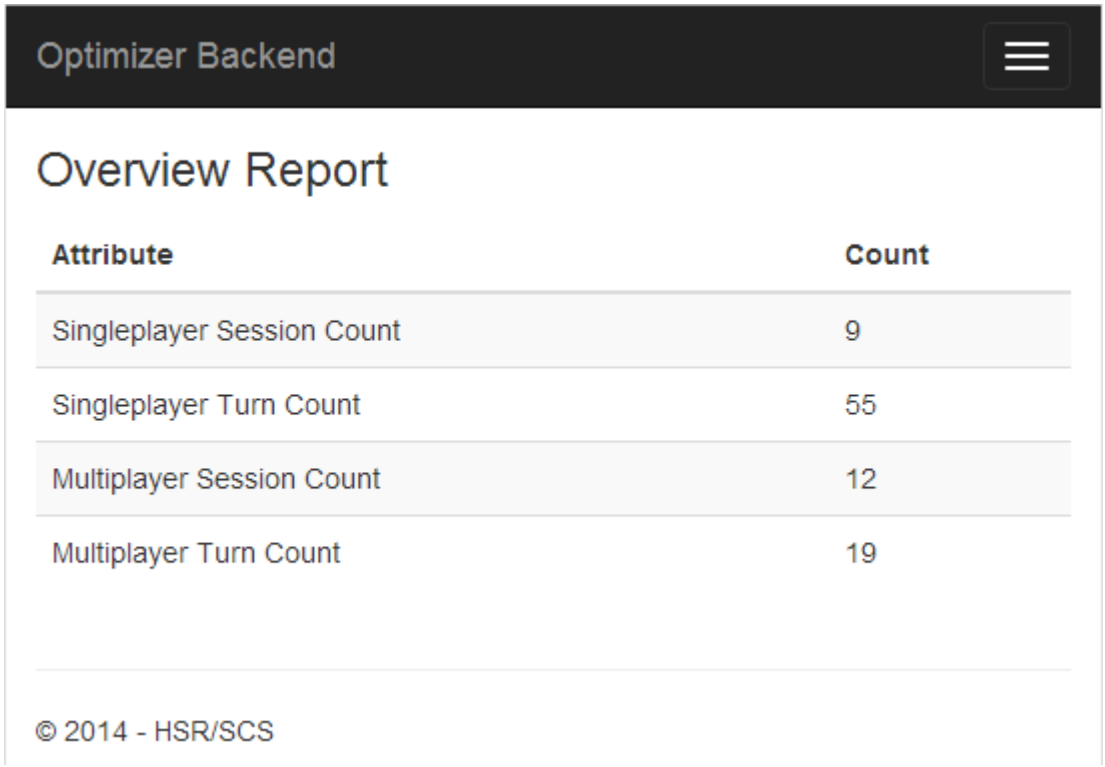


ABBILDUNG 34: RESPONSIVE MVC 5 WEBSITE MIT GLOBALEN STATISTIKEN

Zur Bestimmung eines optimalen Matrix-Generators dient die Grafik in Abbildung 35, welche dem Administrator ebenfalls zur Bestimmung der maximale Rundenzeit hilft.

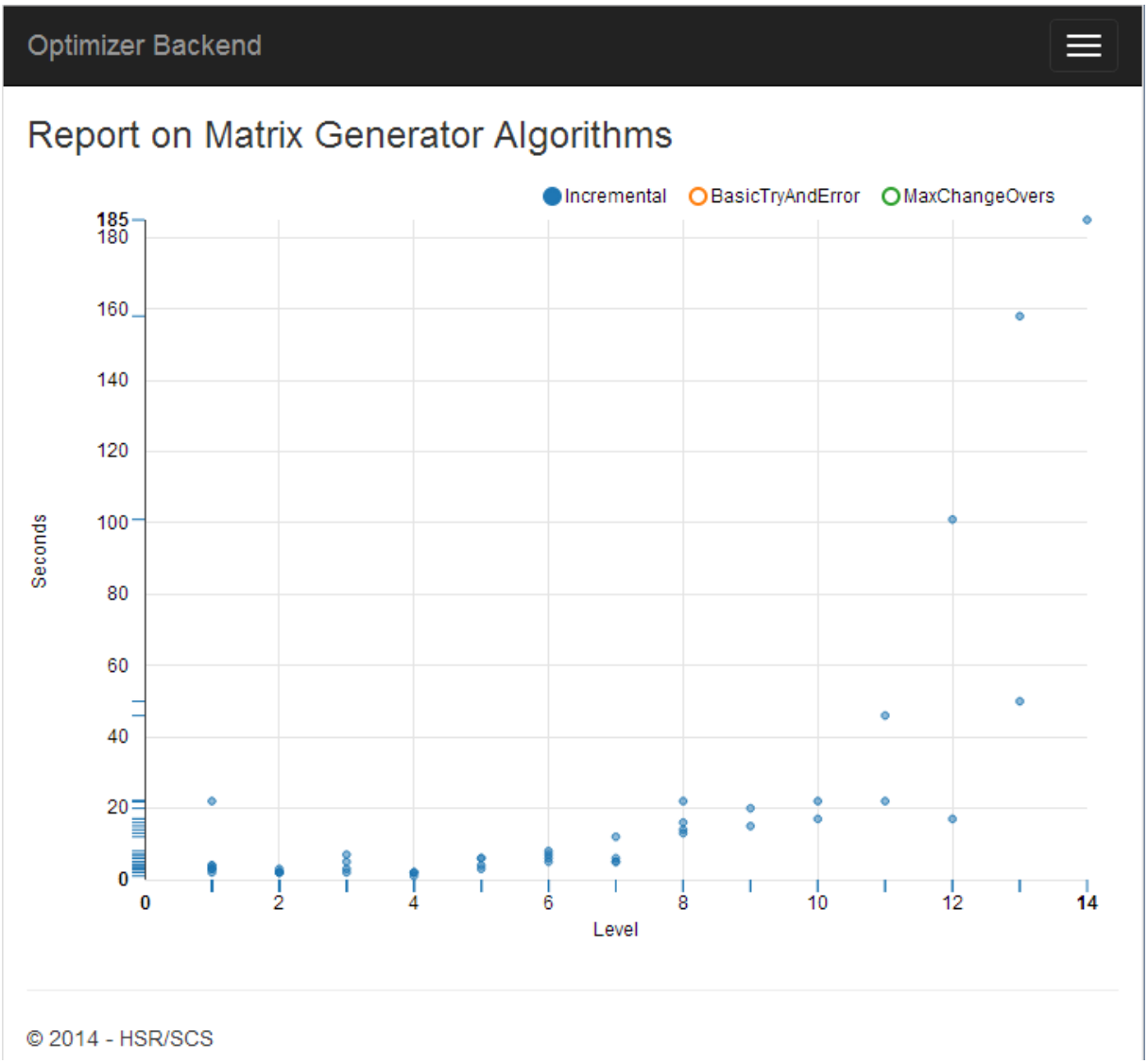


ABBILDUNG 35: AUSWERTUNG DER MATRIX-GENERATOREN ANHAND DER RUNDEN-SPIELZEIT

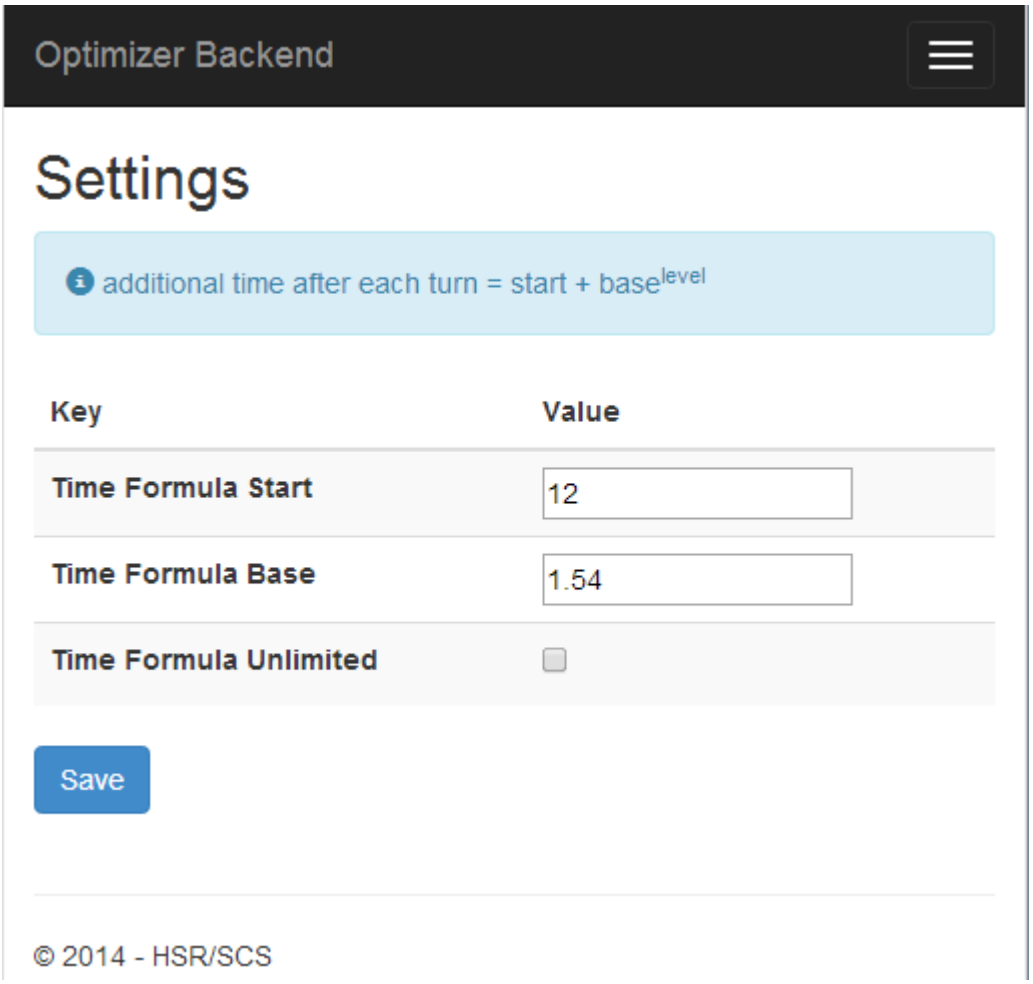


ABBILDUNG 36: PARAMETRIERUNG DER RUNDEN-ZEIT-FORMEL

8 Ergebnis

8.1 Xamarin und MvvmCross

Xamarin bietet die Möglichkeit, dass die Logik der Applikation nur einmal geschrieben werden muss und anschliessend für Android, IOS und Windows verwendet werden kann. Plattformspezifische Logik muss im Core als Interface definiert sein. Die plattformspezifische Implementierung wird im jeweiligen Plattformprojekt implementiert und über Injection verfügbar gemacht.

8.1.1 Vorteil

Xamarin ermöglicht das Schreiben von Applikationen für Android und IOS in Visual C#.

Alles bis auf die View muss nicht neu geschrieben werden.

8.1.2 Nachteil

Das Userinterface muss für jede Plattform implementiert werden, was für ein grafisch aufwendiges Spiel einen grossen Aufwand darstellen kann.

Nicht alle Funktionen und Klassen sind verfügbar, was dazu führt, dass nach Alternativen zu den fehlenden Klassen gesucht werden muss.

8.1.3 Fazit

Die Plattformunabhängige Entwicklung mit Xamarin und MvvmCross ist für C# Entwickler ganz klar zu Empfehlen. Die Kosten, von 1000 Franken pro Entwickler pro Plattform, sind in Relation zu den Entwicklungskosten für die Logik der Applikation sehr gering. Angenommen der Entwickler kostet 200 Franken pro Stunde, dann lohnt sich Xamarin bereits nach 5 Stunden Entwicklungszeit der Clientlogik. Zusätzlich kann der Entwickler in seiner gewohnten Programmiersprache entwickeln was einen positiven Einfluss auf die Entwicklungsgeschwindigkeit hat.

Zusätzlich zu der schnelleren Entwicklungszeit der Applikation wird auch die Sicherheit erhöht, da nicht mehr 3 verschiedene Anwendungen gepflegt werden müssen sondern nur noch eine.

9 Zusammenfassung und Ausblick

9.1 Server

9.1.1 Zusammenfassung

Der Server wurde mit der „Onion Architektur“ erstellt und ist komplett ausprogrammiert, die einzige Änderung welche noch ansteht ist die Wahl welcher Algorithmus für die Erstellung der Multiplayer Turns verwendet werden soll.

Ausblick

Für ein besseres Spielerlebnis seitens der Spieler könnte Push Nachrichten implementiert werden, damit der Spieler sofort informiert wird sobald der Gegner sein Spiel beendet hat.

9.2 Client

9.2.1 Zusammenfassung

Das Spiel wurde gemäss den Wireframes komplett für den Windows Store implementiert. Zusätzlich wurde das Spiel als Prototyp für die Android Plattform umgesetzt.

9.2.2 Ausblick

Das Userinterface für die Android Applikation muss noch fertiggestellt werden. Die bestehenden Views müssen noch den Coding Guidelines angepasst werden.

Für iOS Geräte (iPhone, iPad) müssen noch alle Views und die Plattformspezifischen Services erstellt werden.

9.3 Allgemein

9.3.1 Ausblick

Das Spiel bietet momentan die Möglichkeit den Algorithmus für das Erstellen von Levels auszuwählen, diese Funktionalität muss noch entfernt werden bevor das Spiel in den Store gestellt wird. Diese Funktionalität ist nur für die Untersuchung der Level Generierung gedacht.

A. Abbildungsverzeichnis

B. Literaturverzeichnis

1. **Csikszentmihalyi, Mihaly.** *Flow: The Psychology of Optimal Experience.* 1. s.l. : Harper Perennial Modern Classics, 2008.

2. **Spradley, James P.** *The Ethnographic Interview.*

3. **Nielsen, Jakob.** 10 Usability Heuristics for User Interface Design. [Online] http://www.nngroup.com/articles/ten-usability-heuristics/.

4. **Palermo, Jeffrey.** *The Onion Architecture.* [Online] http://jeffreypalermo.com/blog/the-onion-architecture-part-1/.

5. **NDepend.** Partitioning code base through .NET . [Online] http://www.ndepend.com/Res/NDependWhiteBook_Assembly.pdf.

6. **Ecma.** The JSON Data Interchange Format. [Online] http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-404.pdf.

C. Glossar

Begriff	Beschreibung
Changeover	Als Changeover wird der Farbwechsel innerhalb einer Spalte beschrieben, dabei ist es egal ob zwischen den beiden Farben eine leere Zelle vorhanden ist oder nicht.
Farbwechsel	Siehe Changeover.
Flow	Der Flow beschreibt den Zustand, in welcher ein Spieler alles sich herum vergisst und komplett in das Spiel eintaucht.
Infrastructure as a service	Bei Infrastructure as a service kann Infrastruktur wie zum Beispiel Speicher oder Server gemietet werden.
Optimizer	Siehe Solver.
Software as a service	Bei der Software as a service kann Software wie zum Beispiel das Office gemietet werden.
Solver	Der Solver ist ein von der SCS geschriebenes Programm welches die Anzahl der Changeovers minimiert.
Übergangswechsel	Siehe Changeover.

D. RESTful-Schnittstelle

Die während dieser Arbeit erstellt RESTful-Schnittstelle wurde für die Generierung der Web API 2.1 Help Pages mit Attributen sowie dem Summary erweitert. Unten ist die Startseite der Schnittstelle ersichtlich. Die gesamte Dokumentation ist über die URL http(s)://<server>/help erreichbar.

ASP.NET Web API Help Page

MultiplayerTurn

API	Description
POST api/multiplayer-turns/{id}/solution	Stores the solution to the given turnid.
GET api/multiplayer-turns/{id}	Returns the multiplayerturn with the given id

SingleplayerTurn

API	Description
POST api/singleplayer-turns/{id}/solution	Stores the solution to the given singleplayersessionid
GET api/singleplayer-turns/{id}	Returns the singleplayerturn with the given turnid
GET api/singleplayer-turns/{id}/solution	Returns the solution to the given singleplayersession

Users

API	Description
GET api/users/{id}/multiplayer-session/{opponent}	Returns the multiplayersession for the given userid and opponent
GET api/users/{id}/multiplayer-sessions?status={status}	Returns the MultiplayerSessionRepositoryDto with all session with the given user and state
GET api/users/{id}/multiplayer-encounters	Returns the multiplayer encounters for the given user
POST api/users/{id}/multiplayer-sessions	Stores the given userdto
GET api/users/{id}/random-opponent	Returns a random opponent. The opponent can not have a open game session with the given id.
GET api/users/{id}/highscore?size={size}	Returns the highscore for the given user
GET api/users?pattern={pattern}	Returns all users with the given pattern. It checks wether the user starts with this pattern
GET api/users/{id}	Returns the user with the given username

POST api/users	Stores the given userdto
PUT api/users	Changes the password
POST api/users/{id}/singleplayer-session?algorithm={algorithm}	Starts a new singleplayer session with the given algorithm
GET api/users/{id}/singleplayer-session	Gets the active singleplayer session of the given user.

MultiplayerSession

API	Description
POST api/multiplayer-sessions/{id}/turns	Creates a new turn with the given level
GET api/multiplayer-sessions/{id}/turns?turnStatus={turnStatus}	Returns all turns with the given status
GET api/multiplayer-sessions/{id}/latest-turn	Returns the latest turn for the given session id.

Report

API	Description
GET api/reports/times?algorithm={algorithm}	Returns the time needed by the user to solve the singleplayerturns in the csv format.

SingleplayerSession

API	Description
POST api/singleplayer-sessions/{id}/turns	Creates a new turn for the given singleplayer session
GET api/singleplayer-sessions/{id}/latest-turn	Returns the latest turn for the given singleplayersessionid