MONOPOLY

Projekt Software-Entwicklung 3 SS2021

Projektteilnehmer:

Tabea Schuler - ts204@hdm-stuttgart.de
Nina Nolden - nn028@hdm-stuttgart.de
Marcel Wagner - mw225@hdm-stuttgart.de
Jens Schlegel - js414@hdm-stuttgart.de
Fabian Schütz - fs135@hdm-stuttgart.de

Template 2019-01-04

Über arc42

arc42, das Template zur Dokumentation von Software- und Systemarchitekturen.

Erstellt von Dr. Gernot Starke, Dr. Peter Hruschka und Mitwirkenden.

Template Revision: 7.0 DE (asciidoc-based), January 2017

Inhaltsverzeichnis

Einführung und Ziele

Das Online Multiplayer Spiel “Monopoly” wurde im Rahmen der Lehrveranstaltung Software Entwicklung 3 im Sommersemester 2021 realisiert.

Aufgabenstellung

Aufgabenstellung war es hierbei ein Software Projekt während des Semesters in einer Gruppe umzusetzen.

Das Minimum Viable Product wurde zu Beginn des Projekts wie folgt beschrieben:

- Beim aufrufen der Seite kann ein Spieler eine neue Lobby erstellen, indem er sein Name eingibt.
- Um weitere Spieler zu dem Game hinzuzufügen, teilt der Spieler einen Link.
- Beim öffnen des Links, tritt der Spieler in die Lobby bei und muss dann sein Namen eingeben.
- Die Spieler sind nacheinander an der Reihe.
- Durch das klicken auf einen Button würfelt der Spieler eine Zufallszahl von 1 bis 6.
- Die Zufallszahl gibt die Anzahl der Spielfelder die der Spieler nach vorne geht an.
- Die Position des Spielers wird durch das ändern der der Border Farbe markiert.
- Hat das Grundstück noch keinen Besitzer, stehen die Optionen zum Kauf oder nichtKauf des Grundstücks offen.
- Das kaufen des Grundstückes wird durch einen Button durchgeführt, wenn sich der Spieler auf dem Feld befindet.
- Ist das Grundstück bereits von einem anderen Spieler gekauft, muss die Miete an den Grundstücksbesitzer gezahlt werden.
- Jeder Spieler besitzt sein eigenes Konto mit dem der Spieler Grundstücke kaufen kann, wenn genug Geld auf dem Konto ist.
- Diese gekauften Grundstücke werden mit einem Strich in dem Feld markiert.
- Ereignisfelder und gehe in das Gefängnis Feld haben keine Funktion.
- Der Spieler bekommt Taschengeld, wenn er über “Los” geht, und das doppelte, wenn er auf “Los” kommt.
- Sollte das Konto eines Spielers ins Minus kommen, dann scheidet er aus dem Game aus.

Optionale Funktionen nach erreichen des MVP’s wurden wie folgend definiert :

- Ereignisfelder
- Freiparken
- Grundsteuer für das Feld “Freiparken”
- Datenbank zum speichern des Spielfortschritts
- gehe in das Gefängnis Feld
- Kaufen von Häusern
- handeln von Grundstücke
- Hypothek auf Häuser und Grundstücke
- Anzeigen vom mehreren Spielern auf einem Spielfeld (aktuell nur einer möglich)

Qualitätsziele

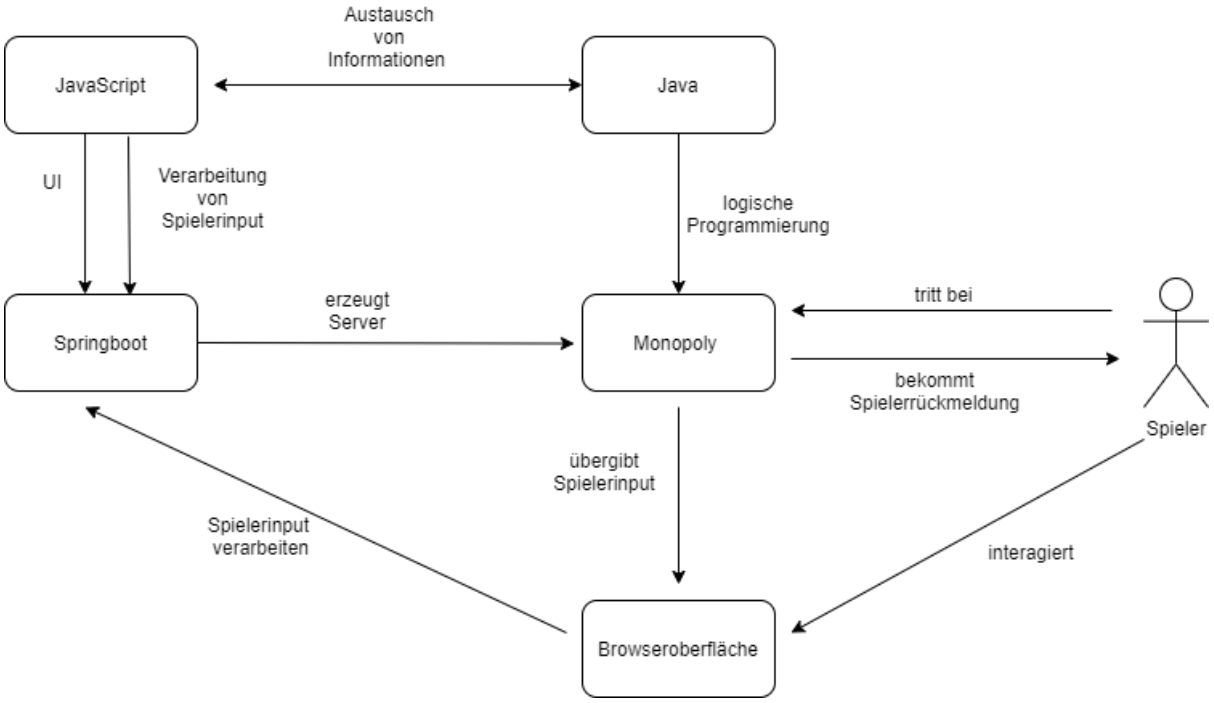
Die folgende Tabelle beschreibt die zentralen Qualitätsziele des Projekts.

Qualitätsziel:	**Motivation und Erläuterung:**
Networking	Für den Ablauf des Spiels ist es Voraussetzung, dass die Mehrspieler Funktion umzusetzen wird. Hierfür ist es notwendig mit Networking den unterschiedlichen Spieler*innen eine Verbindung zum Spiel zu ermöglichen.
User Interface	Die Bedienung des Spiels in der Konsole wäre weniger sinnvoll. Deshalb wird die Bedienung der Spielzüge mit einem intuitiven Frontend umgesetzt.
Clean Code	Der Code wird möglichst verständlich gehalten, um den Einstieg in das Programm einfach zu gestalten. Außerdem sind anschließende Änderungen sowie Tests leichter umzusetzen.

Randbedingungen

- Organisatorische Bedingungen
    - Ein Team aus 5 Studierenden und zwei Mentoren, die das Projekt betreuen.
    - Abgabetermin für die Projektdokumentation, sowie das Repository ist der 18.07.2021.
    - Sinnvolle Projektorganisation durch das Erstellen eines Minimal Viable Projects (MVP) vor dem Beginn der Entwicklung.
    - Wöchentliche oder halb wöchentliche Besprechungen des Entwicklungsstands.
    - Nutzung von GitLab mit der Verwendung von:
        - CI/CD
        - Branching
        - Issue-Tracking
- Technische Bedingungen
    - Wesentliche Funktionalität ist abgedeckt durch Unittest.
    - Die Sprache im Quellcode, sowie den dazugehörigen Kommentaren ist einheitlich auf Englisch gehalten.
    - Anwendung von Clean Coding.
    - Verwendung einer sinnvollen Softwarearchitektur.
    - 2 ausgewählte Techniken des Projekts:
        - Networking
        - UI

Kontextabgrenzung



Fachlicher Kontext

Schnittstelle	**Bedeutung**	**Technischer Kontext**
Benutzer/Spieler	Der Benutzer interagiert mit dem Spielserver über den Webclient. Dort werden in einer Lobby 4 Spieler zusammengebracht. Daraufhin wird eine Session der Monopoly Application gestartet. Die Benutzer interagieren abwechselnd mit dem Spieleserver. Dies wird solange fortgesetzt, bis eine Session endet. Das geschieht dann, wenn ein Spieler den Spielregeln entsprechend verliert. Daraufhin ist die Session des Servers beendet.	Menschlicher Benutzer
Spring boot	Das Server Setup, sowie die Kommunikation zwischen dem Backend und der Webanwendung.	Framework/Web-server

“Monopoly”	Brettspielvorlage des Klassikers “Monopoly” in vereinfachter Version. Dies ist der fertige Zusammenbau aus allen genannten Anwendungen. “Monopoly” läuft über Springboot im Webclient auf einem Server. Es besteht aus Frontend- und Backend Code, der jeweils in Javascript, sowie Java geschrieben ist.	Spielanwendung
Weboberfläche	Unter dem Link https://monopoly-fro.vercel.app zu erreichender Link, der eine Lobby für 4 Spieler bereithält. Dort kann dem Spiel beigetreten werden.	Application
Java Backend	Logische Programmierung des Ablaufs von Monopoly. Die Interaktion der Spieler mit dem Spiel wird hierbei umgesetzt. Das Spielbrett, samt Felder, Würfel und Logik der Bank wird generiert. Es werden auch Dinge, wie das Ende des Spiels oder der Besitz eines Grundstücks hier logisch abgebildet.	Logik
Javascript Frontend	Programmierung jeglicher Darstellungen des Spielbretts, der Spieler, der gekauften Häuser, usw. Sowie die stetige UI Aktualisierung des Spiels, bei jedem Zug eines Spielers, für alle Teilnehmer sichtbar. Zudem werden Nachrichten an Spieler versendet über Aktionen, die im Spiel passieren.	Darstellung
Websocket	Baut die Verbindung zwischen unserem Client und Server aus. Darüber findet dann der Datenaustausch statt.	Kommunikations-protokoll
JSON	Das Datenformat welches verwendet wurde, um die Daten von unseren Backend zum Frontend zu versenden. Gerade mit JavaScript ist es einfach die Daten damit zu verarbeiten, um diese dann dem Spieler anzeigen zu lassen.	Datenformat
Jackson	Eine Bibliothek die wir gebraucht haben, um die Java Objekte zum JSON Datenformat umzuwandeln.	JSON für Java

Lösungsstrategie

Technologieentscheidungen

Da wir alle in Java vertraut sind, war es klar, dass wir die Programmiersprache in unserem Projekt verwenden wollen. Wir haben alle das letzte Semester für unsere Benutzeroberfläche JavaFX verwendet. Das hat aber den Nachteil, dass Nutzer, die unsere Anwendung nutzen wollen, zuerst die Applikation auf dem PC installieren müssen. Da das aber immer mit Aufwand und Vertrauen verbunden ist, und heutzutage immer mehr alles nur noch über den Browser funktioniert, haben wir uns gegen JavaFX entschieden. Dazu fanden wir es auch interessanter und hilfreicher, neue Einblicke in das Framework React zu bekommen.

Somit war es klar, dass unser Backend die Sprache Java verwenden sollte und mit einer Web Anwendung kommunizieren sollte. Dafür kommt das Framework Spring Boot in Frage. Das soll für eine höhere Produktivität sorgen und die Entwicklungszeit reduzieren.
Dazu beinhaltet es ein eigenes Framework für Dependency Injection.
Und mit Spring Web wird der Apache Tomcat als Container verwendet. Welcher gut ist, wenn man einen Server für eine Web Applikation betreiben möchte.

Es ist sehr einfach gemacht, das Projekt mit den Dependencies, die man benötigt, zu erstellen.

Hierbei wurde das Build-Management-Tool Maven gewählt.

Wir brauchten dann auch noch ein Kommunikationsprotokoll, welches uns ermöglicht, die Daten zwischen unserem Server und Client auszutauschen.
Hier haben wir uns für Websocket entschieden. Der entscheidende Punkt war der, dass bei Websocket der Server Nachrichten an den Client schicken kann. Diese werden dann beim Erhalt auf der Client Seite aktualisiert. Damit alle Spieler in der Lobby gleichzeitig die Daten vom Server bekommen und immer auf dem neuesten Stand sind. Das spielt gerade bei Online Spielen oder Messenger eine große Rolle. Das Ganze spielt in Echtzeit ab. Das liegt an der geringen Latenz von Websocket, die dies ermöglicht.

Da die Verbindung bei Websocket bestehen bleibt, und nicht bei jedem Request sich neu verbinden muss, ist das Protokoll sehr effizient.

Dazu läuft die Verbindung Bi-Directional ab. Das heißt, ausgehende und eingehende Nachrichten laufen über denselben Kanal ab.

Hierbei hat uns Spring Boot auch die Arbeit erleichtert, da wir beim Erstellen des Projektes angeben konnten, dass wir Websocket verwenden wollen. Dafür wurden dann die Bibliotheken direkt zu unserer Pom.xml hinzugefügt.

Bei dem Format für den Datenaustausch haben wir uns für JSON entschieden,

da es einfach ist mit JSON umzugehen und andere Formate schnell nach JSON umgewandelt werden können. Die Bibliothek Jackson ermöglicht uns, die Daten von unseren Java Objekten in JSON umzuwandeln, damit wir dann von unseren Backend direkt JSON zu unseren Frontend schicken können. Auf der Client Seite haben wir ‘JSON.parse()’ beim Empfangen von Nachrichten gebraucht, damit JSON zu JavaScript Objekten umgewandelt werden und ‘JSON.stringify()’, um es beim Versenden wieder zu JSON umzuwandeln.

Letztendlich haben wir uns im Frontend für das Framework Next.js entschieden, da es gegenüber React noch ein paar wichtige Vorteile mitbringt. Bei Next.js wird nämlich jede Seite vorgerendert. Somit wird für jede Seite HTML Code generiert und das muss dann nicht mehr auf der Client Seite passieren. Für jede dieser generierten HTML Seite wird nur der minimal benötigte JavaScript Code mitgeliefert. Somit muss dieser JavaScript Code dann nur noch ausgeführt werden, wenn die Seite lädt, um diese interaktiv zu machen.

Das alles sorgt für eine insgesamt deutlich bessere Performance der Webanwendung, was für ein besseres Nutzungserlebnis des Anwenders zugutekommt.

Um unsere Webanwendung zu stylen, haben wir das CSS Framework Tailwind CSS verwendet.
Hat den Vorteil, dass es nur die Styles hinzufügt, die man auch benötigt. Damit bleibt die CSS Konfigurationsdatei deutlich kleiner. Dazu braucht man auch keine IDs, sondern kann direkt die Styling Parameter in className zu dem React Komponent schreiben.
Das hat bei uns zu höherer Effizienz im Team gesorgt und ist für uns auch leserlicher, da man nicht mehr zwischen den CSS Stylesheet und den Komponenten herwechseln muss, sondern nur noch in der einen Datei bleiben kann, in der sich das Komponent befindet.

Ein weiterer Vorteil von Tailwind ist, dass es Mobile First entwickelt wurde. Das bedeutet, dass man zuerst für die Displaygröße des Smartphones entwickelt und dann Schritt für Schritt nach oben geht.

Für uns war es klar, dass es keinen Sinn macht, dass Monopoly Spiel für die Größe eines Smartphones zu entwickeln, da wir das Spielbrett nicht unter die Größe eines Smartphone Bildschirm bekommen hätten.

Dennoch war es ein wichtiger Punkt, der es uns erleichtert hat, die Webanwendung Responsive zu machen. So können die Spieler nicht auf einem Smartphone spielen, aber wenigstens auf einem Tablet oder auch einem kleineren Notebook.

Wir haben uns dann noch für den Just-in-Time Mode in Tailwind entschieden.
Mit diesen war es möglich, dass wir Variablen in den Styling Parametern verwenden konnten.
Somit können wir von unseren Backend den hexadezimalen Farbcode zu unserem Frontend senden. Welcher dann für die Farbe des Spielers verwendet wird.

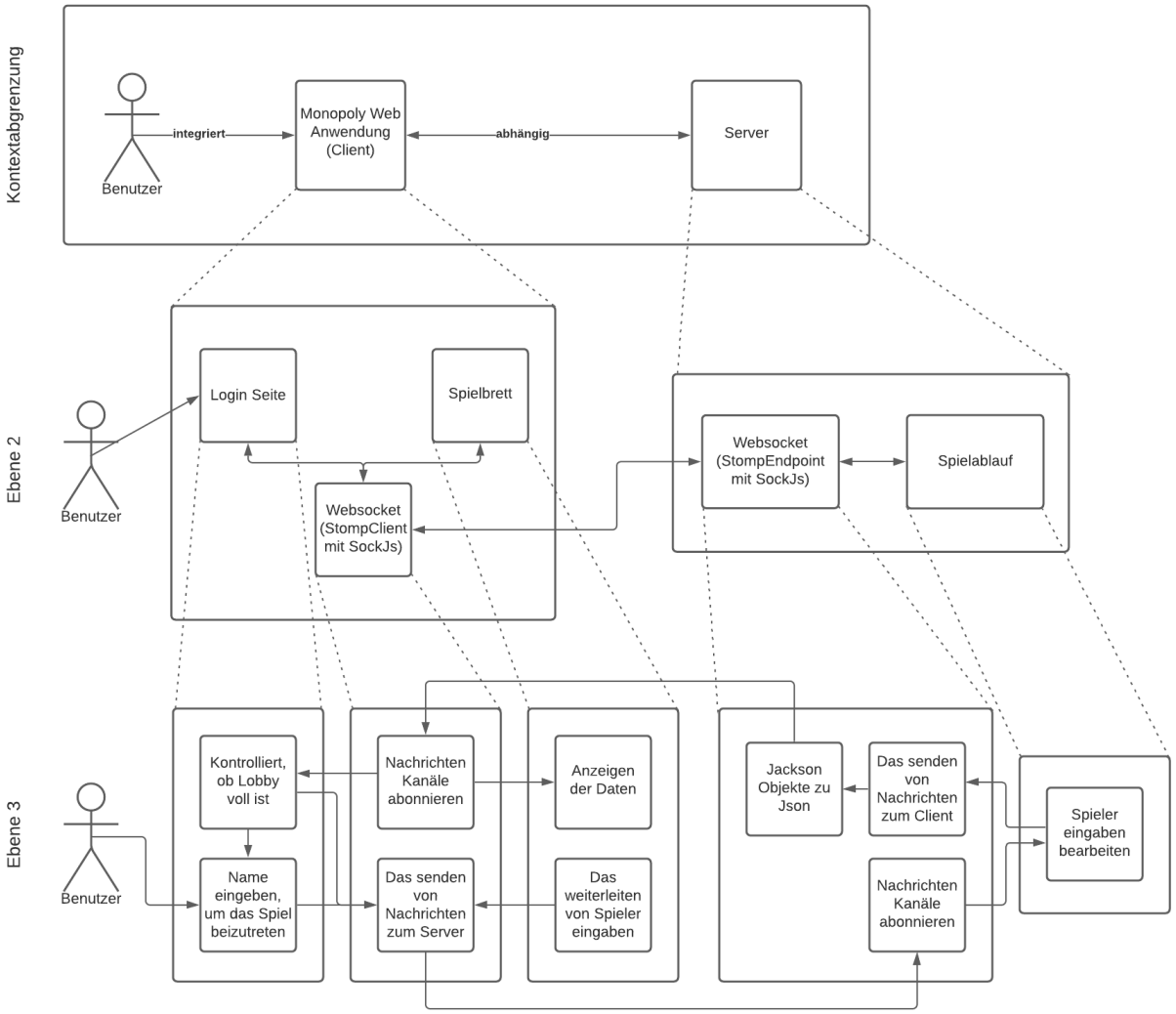
Entscheidungen zur Erreichung der wichtigsten Qualitätsziele

Um unser Projekt gut umsetzen zu können, haben wir bei unserem MVP genau darauf geachtet, welches die wichtigsten Eigenschaften des Spiel sind und deswegen als erstes erledigt werden müssen. Wir haben es in essentielle und optionale Funktionen aufgeteilt. Die essentiellen Funktionen sind entscheidend dafür, damit es einen Spielablauf gibt. Die Optionalen sind Erweiterungen, welche das Spiel abwechslungsreicher machen.
Die Projektstruktur war also von Anfang an so aufgebaut, dass man unkompliziert weitere Features hinzufügen kann.
Da die Aufgabenbereiche sich auch oft überschnitten haben, war es wichtig, die Klassen und Methoden zu dokumentieren. Außerdem haben wir ziemlich früh mit Loggern gearbeitet, um Fehler schnell zu erkennen und zu finden.

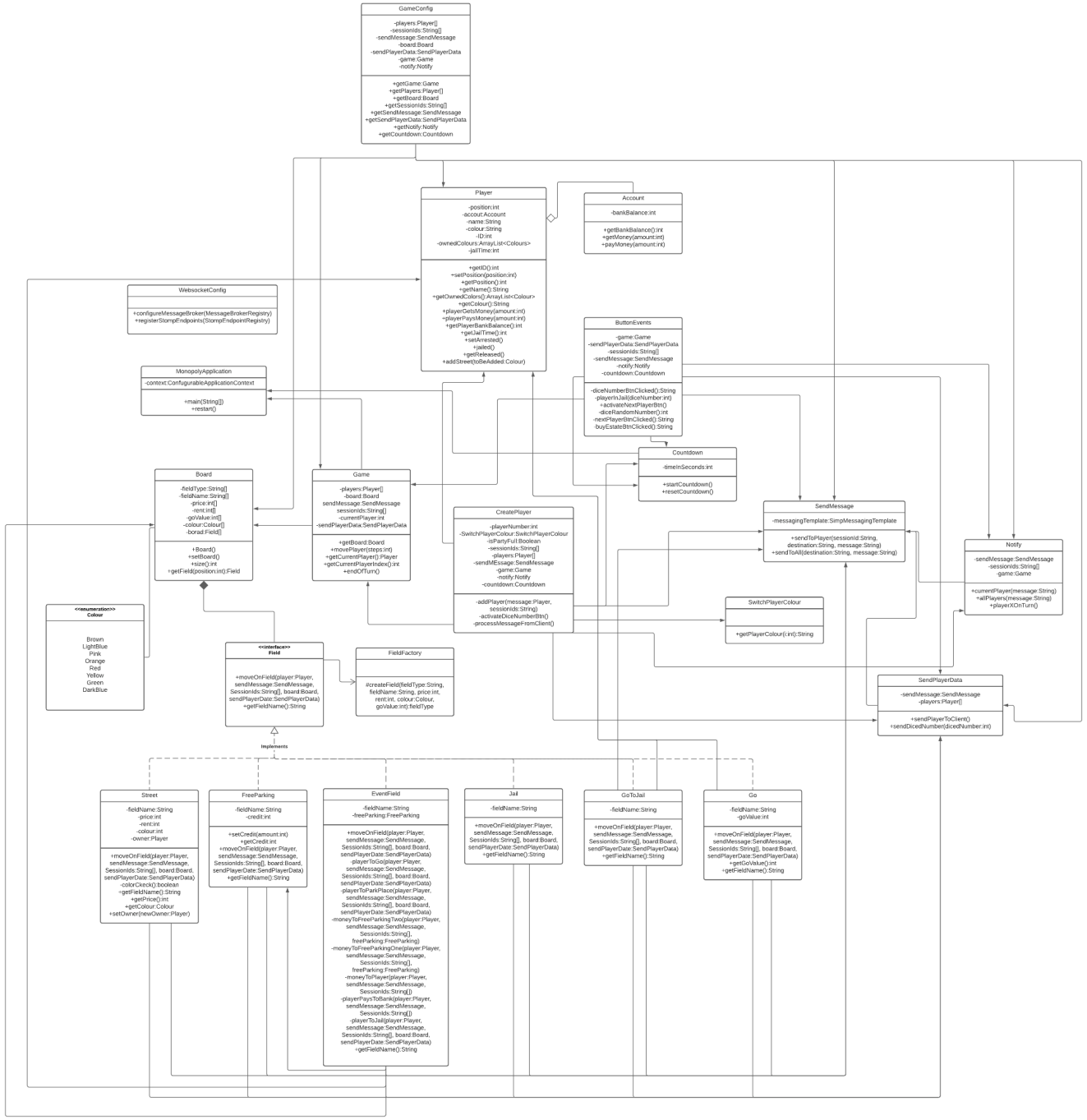
Organisatorische Entscheidungen

Um uns zu organisieren, haben wir am Anfang grob die Aufgaben verteilt. Ab da haben wir uns meistens zwei Mal die Woche getroffen, damit jeder ein Update zu seinem/ihren Teil geben kann oder eine neue Aufgabe übernehmen kann. Dadurch wussten wir immer genau, wer an was gerade arbeitet und man wusste, wer bei bestimmten Fragen der/die Ansprechpartner*in ist.
Für die Aufgaben hat dann jedes Teammitglied eine Issue auf Gitlab erstellt und auf dem eigenen Branch gearbeitet. Den Merge-Request musste immer ein anderes Teammitglied bestätigen, welches sich mit diesem Thema auch auskannte. Dadurch konnten Bugs und Fehler im Programm vermieden werden.

Bausteinsicht

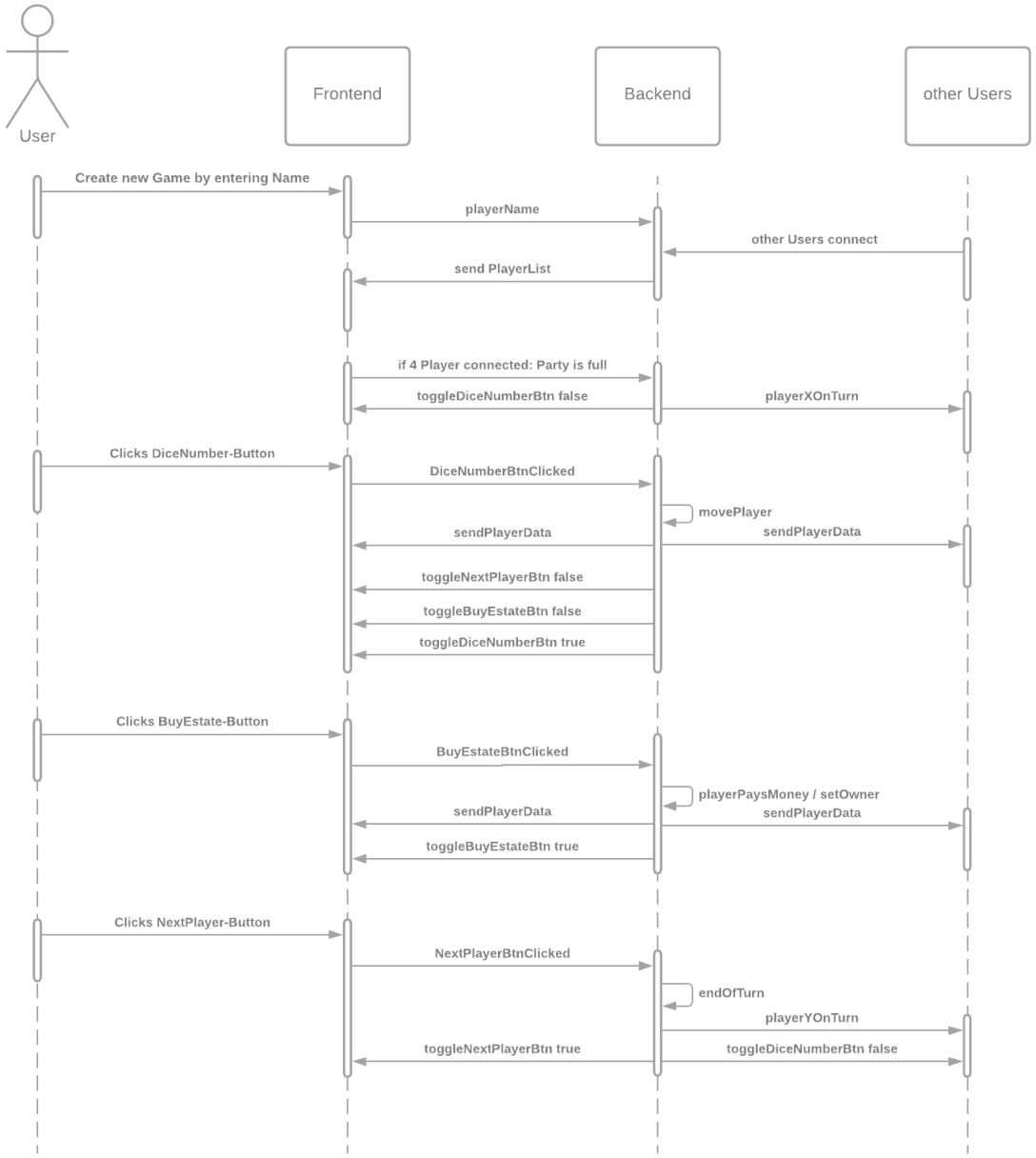


Klassendiagramm



Laufzeitsicht

Systemablauf bei Kauf eines Grundstücks



Querschnittliche Konzepte

User Experience (UX)

Bei einer digitalen Version des Brettspiels Monopoly ist eine Graphische Benutzeroberfläche (GUI) unerlässlich, da diese maßgeblich an der User Experience (UX) beteiligt ist.

Nach dem start des Frontends mit den Befehlen *‘npm install’* und *‘npm run dev’* im Terminal (Unix-Shell, Windows-Eingabeaufforderung, …), wurde das Konzept einer ausschließlichen Navigation über die Benutzeroberfläche verfolgt. Während des Spielverlaufs dient das Terminal lediglich als Ausgabemedium für das parallel betriebene Logging.

Die GUI übernimmt die Informationsvermittlung vom User zur Spiellogik. Im ersten Screen wird der User aufgefordert sein Ingame Name einzugeben, falls schon 4 von 4 Spielern einer Runde beigetreten sind, bekommen alle weiteren Spieler eine Meldung das die Runde bereits voll ist.

Konnte der Runde erfolgreich beigetreten werden, erscheint die eigentliche Spieloberfläche. Jetzt können die zur Auswahl stehenden Optionen gewählt werden. Für die UX werden die Interaktionen der Spieler nicht nur Visuell auf dem Spielbrett angezeigt sondern ebenso in Textform als Nachricht geschickt um eine bessere Spielübersicht für den Spieler zu gewährleisten. Beendet ein Spieler sein aktuellen Zug, werden alle aktiven Buttons ausgegraut und der nächste Spieler ist am Zug.

Architektur- und Entwurfsmuster

Bei der Architektur des Projekts wurde das Konzept der Objektorientierung durchgehend eingehalten. Sogenannte Factory Klassen kümmern sich um die Erstellung der einzelnen Objekte. Jede Feldart (Grundstücke, Ereignisfelder, Los, Gefängnis) wird durch die Klasse *‘Field Factory’* als Objekt erzeugt und gespeichert. Ebenso werden alle beteiligten Spieler als einzelne Objekte erzeugt.

Protokollierung

Für Verbesserungen und Erweiterungen von Monopoly durch Dritte sind die vorhandenen Analysemöglichkeiten von Interesse. Einige Funktionalitäten lassen sich gut über die Unit-Tests überprüfen, trotzdem wird das Programm während des Spielverlaufs Protokolliert.

Durch den Entschluss der reinen GUI Navigation (siehe Querschnittliche Konzepte -User Experience), bleibt die Konsole unangetastet und somit übersichtlicher für die Nachverfolgung der Loggingschritte. Innerhalb des Spiels gibt es feingranularere Logging-Ausgaben, diese werden mit Hilfe der Fremdbibliothek log4j in der Konsole und in einem separaten Logfile ausgegeben. Alle essentiell Wichtigen Klassen werden Protokolliert.

Entwurfsentscheidungen

Frameworks und Bibliotheken

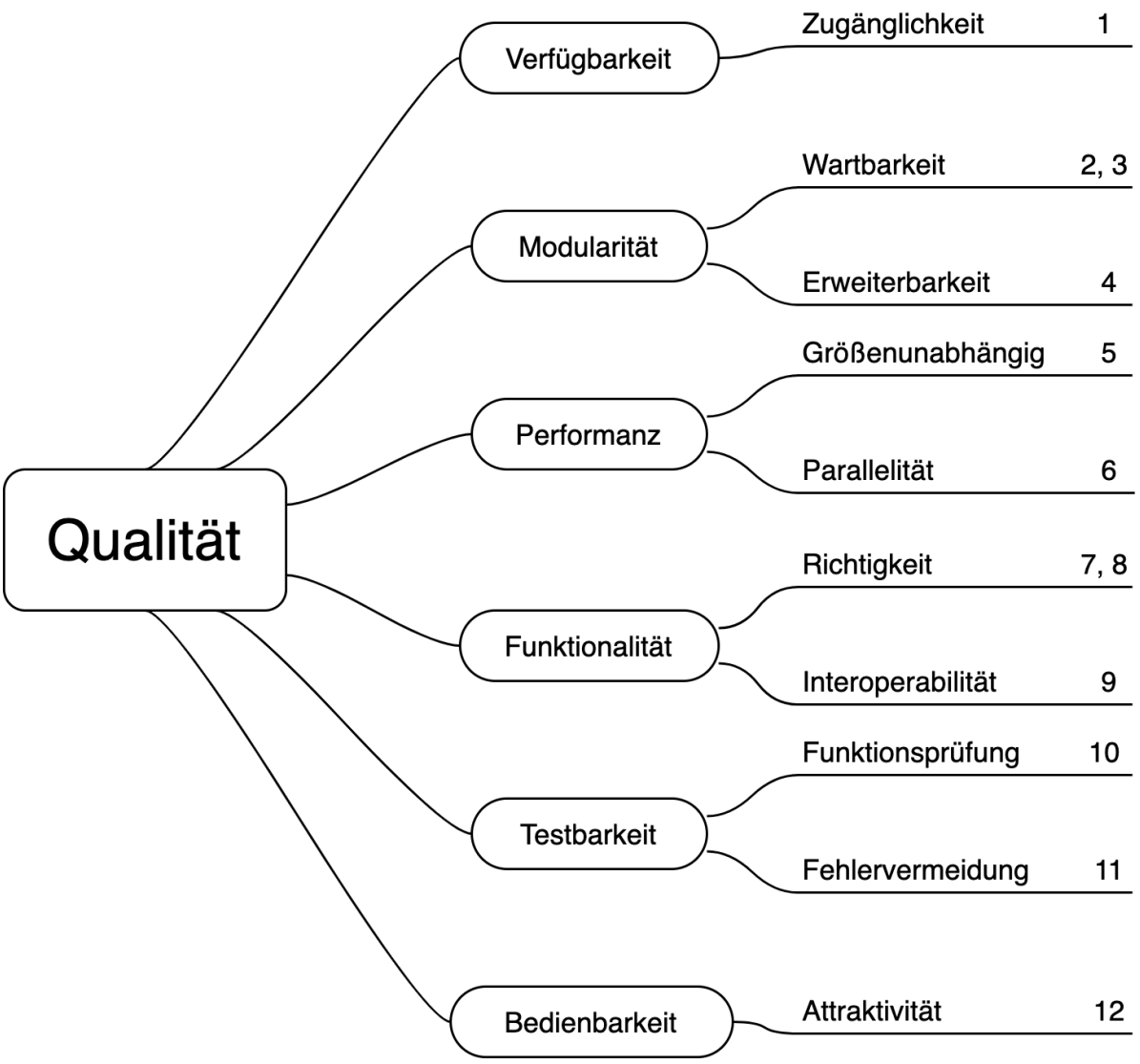
Name des Frameworks	Getroffene Entscheidung
Spring Boot	Erhöht die Produktivität und erleichtert die Aufgaben. Es macht das Verwalten von Bibliotheken einfacher. Kommt mit seinem eigenen Framework für Dependency Injection. Einfaches Entwickeln der Anwendung. Wird oft als Webserver im Zusammenhang mit den Programmiersprachen Java oder Kotlin verwendet.
Next.js	Da es dafür sorgt, dass die Seiten schneller geladen werden. Indem vorher eine statische HTML Seite generiert wird, so dass nur noch der JavaScript Code beim Client geladen werden muss, um diese Seite interaktiv zu machen. Dazu kommt es mit nützlichen Features wie “Fast Refresh”. Sorgt dafür, dass Änderungen im Code bei der Entwicklung direkt angezeigt werden.
Tailwind CSS	Da es komfortabler und lesbarer ist, die Styling Argumente direkt zu dem React Komponenten zu schreiben. Dazu bietet es mehr Optionen, was das Styling angeht. Zusätzlich besitzt es verschiedene Breakpoints, die es erleichtern, die Anwendung responsive zu gestalten. Und die Größe der CSS Datei wird durch Purge deutlich verkleinert.
Spring Boot starter Websocket	Da wir Websocket für die Verbindung zu unserem Client brauchen. Wir haben die Dependency direkt bei der Erstellung des Spring Boot Projektes hinzugefügt.
Spring Boot starter Web	Enthält die Bibliothek Jackson, die wir brauchen um die Java Objekte zum Versenden nach JSON umzuwandeln.
Spring Boot starter Test	Für Tests, die aufwendiger sind und für die Junit nicht mehr ausreicht. Ermöglicht die ganze Anwendung in einem Container zu starten. Um dann mit einem Integrationstest, den Endpoint testen, ob die Verbindung hergestellt werden kann.
Junit	Ist effizient und perfekt zum Testen von einfachen Operationen.
Log4j2	Wichtig, damit der Programmablauf geloggt werden kann. Damit man verfolgen kann, was passiert und bei Fehlern erkennen, an welcher Stelle es passiert.

Architektur

Name der Architektur	Getroffene Entscheidung
Factory Pattern	Größere Erweiterbarkeit (wenn man ein neues Feld generiert, kann man es in die Factory implementieren), größere Sicherheit vor falsch verwendetem Code. Effektivität in der Entwicklung erhöhen.

Qualitätsanforderungen

Qualitätsbaum



Qualitätsszenarien

Nr.	Szenario
1	Ein Benutzer mit leichten Programmierkenntnisse möchte das Programm starten und eine Runde Monopoly spielen. Sobald das Programm runtergeladen wurde, müssen nur Front- und Backend gestartet werden um einer Runde beitreten zu können. Ab jetzt können auch gänzlich unerfahrene Benutzer einfach dem Spiel Beitreten.
2	Ein Architekt, der arc42 anwenden möchte, sucht zu einem beliebigen Kapitel des Templates einen konkreten Beispielinhalt und findet ihn unverzüglich in der Dokumentation.
3	Ein erfahrener Java-Entwickler sucht die Implementierung eines im Entwurf beschriebenen Moduls. Er findet sie ohne Umwege durch die im Quelltext angegebenen Kommentare.
4	Ein Entwickler will die Spieloption “Hypotheken” hinzufügen. Durch Objektorientierung wird für jedes Grundstück auf dem Spielfeld ein eigenes Objekt angelegt. Alle dafür nötigen Daten werden vor der Erstellung der Objekte in einem Array gespeichert, dort können auch die neuen Werte für die Hypotheken eingesetzt werden.
5	Ein Entwickler will unverhältnismäßig viele neue Grundstücke hinzufügen. Dadurch dass die Grundstücks Objekte am anfang der Runde erstellt werden und dann abrufbar in einer Liste gespeichert sind, wird sich die Performance zur Spielzeit nicht bemerkbar verändern.
6	Die Engine sendet durchgehend parallel an alle Spieler die nötigen Informationen des aktuellen Spielstandes. Es können alle Spieler gleichzeitig angesprochen werden.
7 8	Der Spieler kennt die Spielregeln nicht. Das Programm hat sich größtenteils an die Regeln des Originals “Monopoly Junior” gehalten. Sie können einfach nachgelesen werden.
8 9	In einer Spielsituation gibt die Engine einen oder mehrere regelkonforme Züge zur Auswahl. Die restlichen nicht regelkonformen Buttons werden erkannt und ausgegraut. Der Spieler antwortet mit einem der zur Auswahl stehenden Züge.
10 11	Ein Entwickler fügt neuen Quellcode hinzu, um eine bestehende Funktion zu verbessern oder neue Funktionen einzufügen. Dank der bestehenden Unit-Tests, kann neuer Quellcode direkt getestet werden und der Entwickler sieht ob durch seine Änderung Fehler im Stammcode ergeben. Sollte sich das Verhalten unerwartet geändert haben, würde der betreffende Unit-Test fehlschlagen und es wäre in der Regel genau verfolgbar, welche Änderung zum Fehlschlagen des Unit-Tests geführt hat.
11	Ein Entwickler arbeitet zum ersten mal am Projekt mit. Durch einfache Standards im Code, ist dem Entwickler innerhalb kurzer Zeit klar nach welchen Vorgaben er den Code zu implementieren hat. Somit können schon im vorraus Missverständnisse und die dadurch resultierenden Fehler vermieden werden.
12	Ein Benutzer spielt zum ersten mal eine Runde. Durch klar beschriebene Buttons und ein ansprechendes UI, kann der Spieler einfach durch den Spielablauf navigieren. Er wird durch einfache Benachrichtigungen auf dem laufenden gehalten was gerade passiert.

Risiken und technische Schulden

Risiko	Eventualfall Planung	Risikominderung
Aufwand der Implementierung Alle Teammitglieder hatten zu Beginn des Projekts keine Erfahrung mit dem Spring Boot Framework. Zudem wurde dem Projektteam mitgeteilt “Monopoly” sei in vollem Umfang schwer umzusetzen. Daher war es unklar, ob das Projekt mit allen Features bis zum Abgabetermin fertiggestellt werden kann.	Funktionen, die nicht notwendig sind um ein Monopoly Spiel zu spielen wurden vorerst aus der Planung weggelassen. Ein funktionierendes Spiel mit weniger Funktionen sei hierbei wichtiger. Die Einarbeitung mit Spring Boot sollte als erstes stattfinden, damit genügend Zeit für eine eventuelle Planänderung für das Framework bleibt.	Das MVP enthielt nur die dringend notwendigen Funktionen für das Spiel. Alles andere wurde als optional festgelegt. Dadurch wurde der allgemeine Aufwand reduziert. Besprechungen 1-2 mal in der Woche sollten sicherstellen, dass der Zeitplan eingehalten wird und keine Aufgaben zu lange liegen bleiben.
Verwendung von GitLab CI/CD Niemand von uns hatte bis jetzt CI/CD in einem Projekt verwendet. Außerdem muss bei uns sowohl Frontend als auch Backend miteinbezogen werden.		Auf Build- und Test-Stage konzentrieren
Verknüpfung des Codes mit der UI Die Schnittstelle zwischen Code und UI gestaltet sich häufig schwierig. Im Fall Monopoly tauchten immer wieder Probleme auf.	Die UI sollte sofort paralell zu Beginn der Arbeit mit dem Projekt beginnen. So hatte man genug Zeit nach Lösungen zu suchen, sollte die Verknüpfung der UI mit dem Code ein Problem darstellen.	Es wurden in den wöchentlichen Treffen immer wieder die Probleme besprochen und somit der Konflikt zwischen Code und UI gelöst. Dadurch schreitete die Abeit am Projekt langsamer voran, als manchmal geplant. Aber es konnte dadurch immer an Code und UI gleichzeitig gearbeitet werden und nichts musste am Ende noch komplett von Null geschrieben werden.

JUnit-Tests Um unsere Applikation zu verwenden, müssen sich anfangs erst vier Spieler mit Namen einloggen. Das stellt ein Problem dar für die Tests.	Auf Tests konzentrieren, die das aktive spielen nicht benötigen.	Durch unsere Tests wissen wir, dass das Board richtig aufgebaut wird und die Bankbewegungen richtig funktionieren. Der Rest muss durch aktives ausprobieren und mit Hilfe des Loggings überprüft werden.