

Vertiefungsmodul 1

RFID-Monopoly

Hochschule Luzern

MSE Computer Science

Vorgelegt am 22.01.2023 durch:

Robert Schlittler

Glärnischstrasse 7

8800 Thalwil

Betreut durch:

Prof. Dr. René Meier

Experte:

Rolf Schmidinger

Inhaltsverzeichnis

1	Einleitung	1
2	Stand der Forschung.....	3
2.1	Domain Driven Design.....	3
2.2	Clean Architecture.....	6
2.3	Representational State Transfer	7
2.4	Internet of things	9
2.5	Integration von Technologie in Brettspiele	10
2.6	Evaluation von Technologien zum Tracking von Spielfiguren	11
2.6.1	Ultra-Breitband.....	12
2.6.2	Visuelles Tracking.....	12
2.6.3	Radio Frequency Identification.....	13
3	Design.....	16
3.1	Spielregeln Monopoly	16
3.2	Hardwarekomponenten	17
3.2.1	RFID-Module	17
3.2.2	RFID-Sensoren und Mikrokontroller	19
3.2.3	RFID-Tags.....	21
3.2.4	Zentrale Rechnerkomponente	23
3.2.5	LED-Lampen.....	24
3.2.6	Display	24
3.2.7	Stromversorgung	24
3.3	System Design.....	25
3.3.1	Funktionsübersicht	25
3.3.2	Domänen-Design.....	26
3.3.3	System Architektur.....	33
3.4	Prototyp	35
4	Umsetzung.....	38
4.1	Hardware (Game Board)	38

4.1.1	Spielbrett	38
4.1.2	Sensor Events Collector	39
4.1.3	Sensor Events Processor.....	40
4.1.4	Paystation	41
4.1.5	Evaluation RFID-Tag Erkennung	41
4.2	Backend (Game Logic)	42
4.2.1	Projekt Struktur	42
4.2.2	http-Schnittstelle.....	43
4.2.3	Eventing	43
4.2.4	Beispiel-Implementation	46
4.3	Frontend	47
4.3.1	Websocket Implementation.....	47
4.3.2	Beispiel-Implementation Grundstück-Kauf.....	48
5	Evaluation.....	50
6	Schlussfolgerung	51
7	Eigenständigkeitserklärung	52
8	Literaturverzeichnis.....	53
9	Abbildungsverzeichnis	57
10	Tabellenverzeichnis.....	58
11	Anhang	59
11.1	Anhangsverzeichnis	59

1 Einleitung

Das Internet der Dinge (engl. Internet of Things, IoT) ist ein wichtiges Forschungsgebiet der letzten zwei Jahrzehnte (Ahmad et al., 2022, S. 67). Mithilfe von Mikrocontrollern und Sensoren sind Gegenstände in der Lage Veränderungen in der Umwelt zu messen und falls nötig darauf zu reagieren. Die sinkenden Kosten für Hardware ermöglichen es, immer mehr Gegenstände mithilfe von IoT zu verbessern. Diese Gegenstände kommunizieren untereinander über ein Netzwerk (oftmals das Internet) und ermöglichen so eine Erweiterung der eigentlichen Funktionalität oder erleichtern die Verwendung des Gegenstandes.

Im Rahmen dieser Arbeit wird untersucht, inwiefern sich IoT, beziehungsweise Technologie im Allgemeinen, dafür eignet, um das Spielerlebnis des Brettspiels Monopoly zu verbessern. Es gilt als eines der bekanntesten Brettspiele, das ursprünglich im Jahr 1935 von Charles Darrow entwickelt wurde. Es ist ein Spiel, bei dem die Spieler um den Besitz von Immobilien und die Kontrolle über eine fiktive Stadt konkurrieren. In der Standardversion des Spiels werden Würfel verwendet, um die Bewegungen der Spielfiguren auf dem Spielbrett zu bestimmen, und Papiergegeld, um den Kauf und Verkauf von Immobilien und anderen Gegenständen zu simulieren.

Brettspiele werden von Menschen seit Tausenden von Jahren gespielt und tragen zu Spass, Entspannung und Bildung bei (Magerkurth et al., 2004, S. 1). Klassische Brettspiele konkurrenzieren gerade bei jüngeren Menschen mit Computerspielen, wobei Brettspiele den Vorteil haben, dass sie klar mit einem sozialen Austausch assoziiert werden (Magerkurth et al., 2004, S. 1). Für fast alle Spiele sind mehrere spielende Personen erforderlich, welche sich an einem Tisch befinden und miteinander interagieren, voneinander lernen und kommunizieren. Doch Brettspiele können gerade für Anfänger oder jüngere Personen auch gewisse Hürden beinhalten. So gibt es komplizierte Regeln und viele Details, die während dem Spiel beachtet werden müssen. Dies kann dazu führen, dass das Spielen nicht als Spass, sondern als Anstrengung wahrgenommen wird (Magerkurth et al., 2004, S. 1). Hier bieten Computerspiele den Vorteil, da Spielende in einer spielerischen Art und Weise an die Spielmechanismen herangeführt werden, während sie langsam in die Spielwelt eintauchen. Trotz der fehlenden physischen sozialen Interaktion sind Computerspiele sehr beliebt, da mithilfe von Akustik und phantasievollen Umgebungen eine immersive Parallelwelt erschaffen wird (Magerkurth et al., 2004, S. 1).

Durch die Integration von Technologie in klassische Brettspiele sollen die bestehenden Hürden abgebaut werden, während die Vorteile wie eine physische soziale Interaktion und das Erlebnis des Spielbretts verstärkt werden sollen. Durch direktes Feedback des physischen Spiels soll eine immersive Spielerfahrung kreiert werden, welche auch junge Personen interessieren soll.

Im Rahmen dieser Arbeit werden anhand einer Literaturrecherche die unterschiedlichen technischen Möglichkeiten und Sensoren evaluiert, welche sich für die Überwachung des Spielbretts eignen. Mithilfe dieser Evaluation der Technologie wird im Anschluss die verfügbare Hardware verglichen und ausgewählt. Nebst dem IoT-Aspekt beinhaltet diese Arbeit eine umfangreiche System-Architektur, welche anhand von Domain Driven Design und Clean Architecture erstellt wird. Diese Architektur bildet die Ausgangslage für diese Arbeit und soll sowohl die Einhaltung der Spiellogik als auch die Integration des physischen Spielbretts als Teil der Anwendung beinhalten.

Basierend auf der erarbeiteten Architektur wird ein Prototyp erstellt. Dabei liegt der Fokus klar auf der Integration des Spielbretts mithilfe der evaluierten Technologie, statt auf der Implementierung der gesamten Spiellogik. Es sollen die Möglichkeiten und Grenzen der verwendeten Technologie in Bezug auf das Tracking der Spielfiguren auf dem Spielbrett evaluiert werden. Aufgrund des beschränkten Umfangs dieser Arbeit ist keine umfassende Evaluation des Prototyps mit einer Reihe von Probanden möglich. Die Evaluation erfolgt anhand eines Testspiels und den daraus resultierenden Implikationen in Bezug auf das Spielerlebnis.

Für dieser Arbeit wird nebst der wissenschaftlichen Literatur auch sogenannte graue Literatur verwendet. Gerade im Bereich der Software Engineering Themen und der Verwendung von Technologie im Zusammenhang mit Brettspielen gibt es viele wertvolle und hoch aktuelle Internetbeiträge wie Blogs, Videos oder Artikel in Tech-Magazinen, welche einen hohen Detailgrad sowie eine hohe Relevanz vorweisen.

2 Stand der Forschung

Im folgenden Abschnitt werden die Grundlagen für das vorgestellte Projekt erarbeitet. Dazu werden einerseits Software Engineering Themen wie Domain Driven Design und Clean Architecture erklärt, um die System-Architektur der Anwendung anhand klarer Muster und Vorgehensweisen zu strukturieren. Andererseits werden die Grundlagen für die Hardware des Spielbretts erarbeitet. Dazu werden mehrere Technologien für das Tracking der Spielfiguren auf dem Brett anhand aktueller Literatur evaluiert. Um der Arbeit einen thematischen Kontext zu verleihen, werden die Themen Internet der Dinge und die allgemeine Integration von Technologie in Brettspiele erläutert.

2.1 Domain Driven Design

Domain Driven Design (DDD) ist ein Ansatz in der Softwareentwicklung, bei dem die Domäne - das fachliche Gebiet, auf dem sich die Anwendung konzentriert - im Mittelpunkt steht. Das Ziel von DDD ist es, die Kommunikation und Zusammenarbeit zwischen Entwickelnden, Fachpersonen und Anwendenden zu verbessern, um eine Anwendung zu erstellen, welche die Bedürfnisse der Endbenutzenden erfüllt (Banks, 2022). Die folgenden Begriffe sind im Zusammenhang mit DDD zentral:

Kontext: Der Kontext, in dem ein Wort oder eine Aussage erscheint, bestimmt seine Bedeutung. Aussagen über ein Modell können nur in einem Kontext verstanden werden (Banks, 2022).

Modell: Ein System von Abstraktionen, das ausgewählte Aspekte eines Bereichs beschreibt und zur Lösung von Problemen im Zusammenhang mit diesem Bereich verwendet werden kann (Banks, 2022).

Subdomäne: Eine Domäne kann aus mehreren Subdomänen bestehen, die einzelne Geschäftslogiken gruppieren (Vernon, 2013, S. 44). Statt alles in einer Domäne zu implementieren, propagiert Vernon, dass die gesamte Domäne in eine zentrale Domäne (Core Domäne) und weitere Subdomänen unterteilt wird, welche zusammenfassend die vollständige Geschäftslogik abbildet (2013, S. 44).

Allgegenwärtige Sprache: (Ubiquitous Language): Eine gemeinsame Sprache, die um das Domänenmodell herum strukturiert ist und von allen Teammitgliedern verwendet wird, um alle Aktivitäten des Teams mit der Software zu verbinden (Banks, 2022).

Begrenzter Kontext (Bounded Context): Eine Beschreibung eines Bereichs oder Teilsystems innerhalb dessen ein bestimmtes Modell definiert und anwendbar ist (Banks, 2022). Es bildet einen semantischen Rahmen, in dem jeder Teil der Software eine spezifische Bedeutung hat und der Kontext widerspiegelt die gemeinsame Sprache des Teams (Vernon, 2016, S. 13). Das gleiche Wort kann in unterschiedlichen Kontexten eine andere Bedeutung haben.

Nebst diesen Begriffen, definiert DDD auch sogenannte Building Blocks (Bausteine). Um DDD im Rahmen eines Projektes anzuwenden, müssen diese abstrakten Konzepte unbedingt definiert und verstanden werden. Building Blocks sind wichtig, da sie die Grundlage für die Modellierung von komplexen Strukturen in einer Geschäftsdomäne darstellen. Sie ermöglichen es, die Konzepte in der Geschäftsdomäne auf eine strukturierte und verständliche Weise darzustellen und die Geschäftslogik klar von anderen Aspekten der Anwendung zu trennen. Folgende Bausteine gibt es:

Entities: Dies sind die zentralen Objekte in der Geschäftsdomäne, die eine eindeutige Identität und einen Zustand haben (Banks, 2022). Beispiele für Entities in einem E-Commerce-System könnten Kunden, Produkte oder Bestellungen sein.

Value Objects: Dies sind Objekte, die keine eindeutige Identität haben, sondern lediglich einen bestimmten Wert repräsentieren (Banks, 2022). Beispiele für Value Objects in einem E-Commerce-System könnten Preise, Mengen oder Adressen sein.

Domain Event: Ein Domain Event ist ein Konzept, das dazu verwendet wird, Änderungen im Zustand einer Entity oder einem Value Object in der Geschäftsdomäne zu dokumentieren (Banks, 2022). Domain Events informieren stets über eine Zustandsänderung, welche in der Vergangenheit liegt und bereits erfolgt ist. Ein Domain Event könnte zum Beispiel dazu verwendet werden, um zu dokumentieren, dass eine Kundin oder ein Kunde eine Bestellung aufgegeben hat. Dieser Domain Event könnte dazu benutzt werden, um andere Komponenten in der Anwendung, wie z.B. eine Lieferungsverfolgung, auszulösen.

Aggregate: Ein Aggregate ist eine Sammlung von Entities und Value Objects, die zusammen eine Einheit bilden (Banks, 2022). Aggregates werden in der Regel verwendet, um die Grösse und Komplexität von Entities und Value Objects in der Geschäftsdomäne zu begrenzen (Banks, 2022). Ein Aggregate könnte verwendet werden, um eine Bestellung als Sammlung von Entities und Value Objects zu modellieren. Die Bestellung könnte aus Entities wie Kunde und Produkt bestehen, sowie Value Objects wie Preis und Mengenangaben.

Service: Dies sind abstrakte Konzepte, die Geschäftslogik implementieren, die nicht direkt zu einer Entity oder Value Object gehören (Banks, 2022). Beispiele für Services in einem E-Commerce-System könnten die Berechnung von Versandkosten oder die Validierung von Zahlungen sein.

Repository: Ein Repository ist ein Konzept, das dazu verwendet wird, den Zugriff auf Entities und Value Objects in der Geschäftsdomäne zu verwalten (Banks, 2022). Repositories stellen eine abstrakte Schnittstelle bereit, die es ermöglicht, Entities und Value Objects zu speichern, zu laden und zu ändern, ohne dass die technologische Implementierung dahinter sichtbar wird (Banks, 2022).

Factory: Eine Factory ist ein Konzept, das dazu verwendet wird, Instanzen von Entities und Value Objects zu erstellen (Banks, 2022). Factories stellen eine abstrakte Schnittstelle bereit, die es ermöglicht, Instanzen von Entities und Value Objects zu erzeugen, ohne dass die technologischen Details dahinter sichtbar werden (Banks, 2022).

Um DDD im Rahmen eines Projekts einzusetzen, gibt es mehrere Ansätze. Gemäss Vernon (2016, S. 117) kann mithilfe des sogenannten Event Storming die Domäne innerhalb kurzer Zeit definiert werden. Dabei werden zuerst die relevanten Events der Geschäftsfälle identifiziert. Anschliessend werden diese mit der auslösenden Aktion und der entsprechenden Entität verknüpft. Dies erlaubt dann die Zuordnung zu einem Bounded Context. Durch dieses Vorgehen sollen möglichst schnell erste Erfolge erzielt werden und die Domäne mitsamt der gemeinsamen Sprache wird kollaborativ von allen Beteiligten entwickelt und besser verstanden (Vernon, 2016, S. 117).

Die Anwendung von DDD bringt folgende Vorteile:

- Verbesserte Kommunikation: Die gemeinsame Sprache, welche kollaborativ erarbeitet wird, vereinfacht die Kommunikation der Projektbeteiligten und minimiert den Anteil an technischem Fachjargon während des gesamten Entwicklungszyklus (Banks, 2022).
- Mehr Flexibilität: Das entwickelte Modell ist modular und abgekapselt. Dies erlaubt später eine einfache Änderung des Systems oder einzelner Komponenten (Banks, 2022).
- Domäne vor Interface: Der Fokus liegt auf der Entwicklung einer robusten Domäne, welche die Geschäftslogik korrekt und optimal abbildet, statt auf der Orientierung am grafischen Interface (Banks, 2022).

Mit der Anwendung von DDD ist es möglich, bessere Software zu konzipieren. Die positive Resonanz trägt wesentlich zur steigenden Beliebtheit von DDD bei, welche in einer zunehmenden Anzahl an publizierten Büchern, Konferenzen, Referaten und Kursen resultiert (Tilkov, 2021). Doch DDD ist nicht für jedes Projekt geeignet und es existieren auch gewisse Nachteile:

- Fundiertes Domänen-Design notwendig: Um ein erfolgreiches Domänen-Design zu erstellen benötigt es nicht nur technisch versierte Projektmitarbeitende, sondern vor allem Fachpersonen, welche die Domäne sehr gut verstehen und erklären können (Banks, 2022).
- Iteratives Vorgehen zwingend: Dies kann als Vorteil gesehen werden, ist jedoch für viele Organisationen gerade in der Zusammenarbeit mit den Anwendenden noch ein Hindernis für die Umsetzung von DDD (Banks, 2022). Das iterative Verbessern und Herantasten an die Domäne und die gemeinsame Sprache ist ein zentrales Element der Vorgehensweise.
- Ungeeignet für sehr technische Projekte: DDD ist nicht für Projekte geeignet, welche eine sehr hohe technische Komplexität und nur einen geringen Anteil an Domänenlogik vorweisen (Banks, 2022).

2.2 Clean Architecture

Clean Architecture ist ein Software-Architekturmuster, das von Robert C. Martin (2017) entwickelt wurde. Es zielt darauf ab, die Kopplung zwischen den verschiedenen Komponenten einer Software-Anwendung zu minimieren und die Wiederverwendbarkeit und Pflegbarkeit von Code zu verbessern. Im Zentrum steht das Clean Architecture Diagramm, welches in Abbildung 1 zu sehen ist. Ein zentraler Grundsatz der Clean Architecture ist es, dass Abhängigkeiten stets nach innen «zeigen», und die inneren Ringe somit unabhängig von den äusseren sind (Martin, 2012). In der Mitte befinden sich die Entitäten der Domäne. Die Entitäten stellen sicher, dass sie sich stets in einem validen Zustand befinden. Sie kapseln die allgemeinste und hochrangigste Geschäftslogik ab, welche damit direkt assoziiert werden kann (Martin, 2012). Verschiedene Entitäten werden dann vom Ring Use Cases dazu verwendet, die Geschäftslogik applikationsweit auszuführen (Martin, 2012). Der nächste Ring «Interface Adapters» stellt sicher, dass die Daten aus der Use-Case-Schicht in einem am besten geeigneten Format an die äussere Schicht weiter geleitet wird (Martin, 2012). Dies bedeutet streng genommen, dass keine der inneren Schichten etwas über die Datenbank wissen sollte (Martin, 2012). Die äusserste Schicht besteht im Allgemeinen aus Frameworks und Tools wie der Datenbank, dem Web-Framework usw. (Martin, 2012). Im Allgemeinen wird in dieser Schicht nicht viel Code geschrieben, ausser dem sogenannten Glue-Code, der mit dem nächsten Kreis nach innen kommuniziert (Martin, 2012).

Indem die Software in Schichten aufgeteilt und die Abhängigkeitsregel eingehalten wird, entsteht ein System, das besser testbar ist (Martin, 2012). Wenn einer der externen Teile des Systems veraltet, wie die Datenbank oder das Web-Framework, kann dieses Element mit einem Minimum an Aufwand ersetzt werden (Martin, 2012).

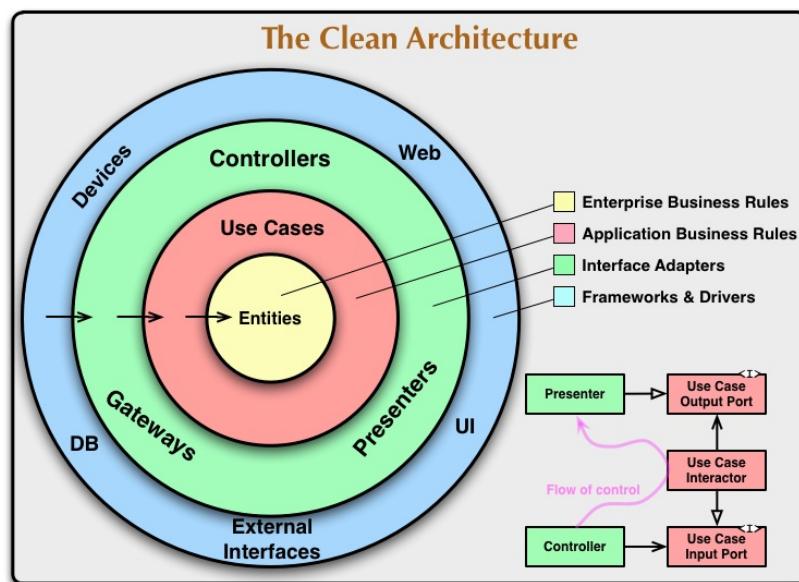


Abbildung 1 The Clean Architecture Diagram (Martin, 2012)

Durch die Verwendung von Clean Architecture können kohärente Systeme entwickelt werden, welche unabhängig von der verwendeten Technologie sind und die Wiederverwendbarkeit von Code begünstigen (Vasconcelos, 2022). Clean Architecture verleiht einem System zudem Struktur und sorgt dafür, dass es auch über lange Zeit noch wart- und erweiterbar bleibt (Vasconcelos, 2022). Für die Anwendung von Clean Architecture in einem Projekt sind jedoch erfahrene Entwickelnde nötig, welche die Design Patterns und die Struktur in der Software rigide umsetzen (Vasconcelos, 2022). Gemäß Vasconcelos (2022) muss evaluiert werden, ob sich der Aufwand von Clean Architecture für kleine und simple Projekte lohnt, oder ob einfachere System-Architekturen eventuell besser geeignet sind. Gerade wenn bereits zu Beginn klar ist, dass ein System nicht erweitert wird oder es sich um ein einmaliges, simples Produkt handelt (Vasconcelos, 2022).

2.3 Representational State Transfer

REpresentational State Transfer (REST) ist ein zentraler Architektur-Stil für Web-Services mit einem Application Programming Interface (API) und gilt als de facto Standard für moderne Webapplikationen (Masse, 2011, S. 5). Im folgenden Abschnitt werden die für diese Arbeit relevanten Aspekte von REST erläutert. REST gliedert die Endpunkte in verknüpfte Ressourcen und stellt diese auf eine konsistente Art und Weise zur Verfügung (Masse, 2011, S. 6). Die darunterliegenden Daten werden in sogenannte Collections und darin enthaltene einzelne Documents unterteilt (Masse, 2011, S. 17). Diese Struktur kann dann in der Repräsentation der Uniform Resource Identifier (URI) verwendet werden. Trotz der Verbreitung von REST besteht kein offizieller Standard. Dennoch gibt es gewisse Grundsätze, welche als allgemein akzeptiert gelten.

Grundsätzlich werden Collections mithilfe von Nomen in der Mehrzahl abgebildet, während ein Document ein Nomen in der Einzahl ist (Donovan & Au-Yeung, 2020; Masse, 2011, S. 17). Eine URI für eine Fussball-Liga könnte zum Beispiel wie folgt abgebildet werden:

<http://api.soccer.restapi.org/leagues/seattle/teams/trebuchet/players>

Bezüglich der Strukturierung der URI gibt es einen weiteren Grundsatz: Funktionen sollten nicht als Verben in der URI abgebildet werden (Donovan & Au-Yeung, 2020; Masse, 2011, S. 18). Gemäß Masse (2011, S. 18) sollten die zur Verfügung stehenden HTTP-Methoden verwendet werden. Ein Endpunkt zum Löschen eines Documents sollte also nicht mithilfe von DELETE /**deleteUser**/1234 sondern mit DELETE /**users**/1234 abgebildet werden (Masse, 2011, S. 18). Generell gilt es als Best Practice, wenn als Standardformat für den Datenaustausch JavaScript Object Notation (JSON) verwendet wird (Donovan & Au-Yeung, 2020; Masse, 2011, S. 47). JSON gilt als leichtgewichtige und allgemeinverständliche Repräsentation der Entitäten und ist ein etabliertes Format, welches von den meisten Systemen unterstützt wird (Donovan & Au-Yeung, 2020; Masse, 2011, S. 47).

Beim Manipulieren von Daten mittels einer REST-Schnittstelle gilt es zu beachten, dass der Web-Client ein Document in dem Zustand übermittelt, in dem er es gerne abbilden möchte (Masse, 2011, S. 44). Möchte ein Web-Client beispielsweise die E-Mail-Adresse einer Person ändern, so würde der entsprechende Aufruf nach REST wie folgt aussehen:

```
PUT http://myapi.org/employees/1234
{"name": "Robert Schlittler", "email": "mymail@example.ch"}
```

Nicht konform wäre der folgende Aufruf:

```
PUT http://myapi.org/employees/1234/updateMail
{"email": "mymail@example.ch"}
```

Dieser Grundsatz lässt sich sehr gut mit dem DDD-Ansatz kombinieren. Da somit eine Art Zustandsmaschine entsteht, in welcher der Web-Client den Wunschzustand vorgibt, und die Applikation die Entität in den gewünschten Zustand bewegt und gleichzeitig die für die Domäne relevanten Events auslöst. In diesem Zusammenhang ist die 2010 eingeführte HTTP-Methode PATCH äusserst relevant. Sie erlaubt eine teilweise Modifikation der Ressource, indem nur jene Felder übermittelt werden, welche geändert werden sollen. Dadurch verändert sich der oben dargestellte Aufruf wie folgt:

```
PATCH http://myapi.org/employees/1234 {"email": "mymail@example.ch"}
```

Der Vorteil davon ist, dass der Client nicht das gesamte Objekt übermitteln muss, was zu einer vereinfachten Logik und reduzierter Bandbreite führt (Miskovic, 2022).

Abschliessend ist anzumerken, dass die vorgestellten Grundsätze keine verpflichtenden Richtlinien darstellen. Konkret bedeutet das, dass die Implementation eines REST-Endpunktes theoretisch nicht der Funktion entsprechen muss, welche durch die URI suggeriert wird. Der Endpunkt

```
PUT http://myapi.org/players/1234
```

könnte auch dazu verwendet werden eine Ressource zu löschen, auch wenn dies in der Literatur auf keinen Fall empfohlen wird (Donovan & Au-Yeung, 2020; Masse, 2011, S. 23–27).

2.4 Internet of things

Das IoT bezieht sich auf die Vernetzung von Alltagsgegenständen und Maschinen mit dem Internet, um Daten zu sammeln und zu verarbeiten (Rose et al., 2015, S. 7). Durch die Verbindung von Gegenständen mit dem Internet werden diese "smart" und können miteinander kommunizieren und gesteuert werden (Rose et al., 2015, S. 7).

Smart Devices sind elektronische Geräte, die für einen bestimmten Zweck entwickelt wurden und über eine Internetverbindung verfügen, um Daten zu sammeln, verarbeiten und übertragen (Silverio-Fernández et al., 2018, S. 2). Smart Devices sind Teil des IoT, da sie miteinander und mit dem Internet verbunden sind und zur Erfassung und Verarbeitung von Daten beitragen. Beispiele für Smart Devices sind Smartphones, Smartwatches, Smart-TVs und Smart-Thermostate (Silverio-Fernández et al., 2018, S. 8).

Der Unterschied zwischen IoT und Smart Devices besteht darin, dass das IoT ein umfassendes Konzept für die Vernetzung von Gegenständen mit dem Internet darstellt, während Smart Devices spezifische Geräte sind, die für einen bestimmten Zweck entwickelt wurden und über eine Internetverbindung verfügen (Silverio-Fernández et al., 2018, S. 9). Smart Devices sind Teil des Internet der Dinge, aber das Internet der Dinge umfasst auch viele andere Gegenstände, die miteinander und mit dem Internet verbunden sind (Silverio-Fernández et al., 2018, S. 9). Ein Beispiel für das IoT wäre ein Smart Home-System, bei dem verschiedene Geräte wie Thermostate, Lichtsteuerungen und Sicherheitskameras miteinander verbunden sind und über das Internet gesteuert werden können. Durch die Vernetzung von Gegenständen mit dem Internet werden neue Möglichkeiten für die Automatisierung und Optimierung von Prozessen geschaffen.

Das IoT hat das Potential, die Art und Weise, wie wir leben und arbeiten, zu verändern, indem es die Effizienz von Prozessen verbessert und neue Dienstleistungen und Anwendungen ermöglicht (Rose et al., 2015, S. 4). Es gibt jedoch auch Bedenken hinsichtlich der Sicherheit und Privatsphäre im Zusammenhang mit dem IoT, da die Verbindung von Gegenständen mit dem Internet das Risiko von Cyberangriffen erhöht (Rose et al., 2015, S. 45). Rose et al. (2015, S. 20–21) heben dabei hervor, dass durch die hohe Verbreitung von IoT-Geräten eine Sicherheitslücke in einem Gerät zu einer weltweiten Bedrohung werden kann, indem beispielsweise Tausende Anfragen oder Spam-Nachrichten von hunderten Geräten auf der ganzen Welt versendet werden. Hinzu kommt, dass IoT-Geräte im privaten Gebrauch oftmals mit sensitiven Daten wie Videoaufzeichnungen, Gesundheitsdaten oder Anwesenheitszeiten im zu Hause arbeiten und ein Angriff auf solche Geräte somit für kriminelle durchaus von Interesse ist (Rose et al., 2015, S. 20–21). Es ist wichtig, dass Sicherheitsmaßnahmen ergriffen werden, um diese Risiken zu minimieren und die Privatsphäre der Benutzenden zu schützen. Denn in Zukunft wird das IoT voraussichtlich eine wichtige Rolle in vielen Branchen wie beispielsweise Gesundheitswesen, Transport, Energie und Fertigung spielen (Rose et al., 2015, S. 8).

Rose et al. (2015, S. 1) kommen zum Schluss, dass es unterschiedliche Szenarien bezüglich der Anzahl an verbundenen IoT-Geräte in der Zukunft gibt. Klar scheint, dass sowohl die Anzahl der verbundenen Geräte als auch die Grösse des IoT-Marktes stark zunehmen wird. Es zeigt sich, dass IoT im privaten Gebrauch zunehmend verbreitet ist und dass immer mehr Menschen IoT-Geräte nutzen, um ihr tägliches Leben zu vereinfachen und zu verbessern (Rose et al., 2015, S. 4). Es ist jedoch wichtig zu beachten, dass die Verbreitung von IoT im privaten Gebrauch je nach Region unterschiedlich sein kann (Rose et al., 2015, S. 18). In einigen Ländern könnte die Verwendung von IoT-Geräten im privaten Gebrauch höher sein als in anderen, abhängig von Faktoren wie der technologischen Entwicklung und dem Zugang zu Internetdiensten (Rose et al., 2015, S. 18).

2.5 Integration von Technologie in Brettspiele

Mit der hohen Verfügbarkeit von Smartphones ist die Integration von Apps in ein Brettspiel naheliegend (Technavio, 2020). Durch eine App können Brettspiele um eine Dimension erweitert werden und zum Beispiel gewisse Szenen als Video wiedergeben. Darüber hinaus kann eine App genutzt werden, um den spielenden Personen die Spielmechanik zu veranschaulichen und so die Lernkurve und Lernerfahrung für neue Spieler wesentlich verbessern (Technavio, 2020). Nebst der Integration von Smartphones ist zu beobachten, dass einige Hersteller auf die Integration von NFC, Bluetooth oder Augmented Reality zurückgreifen (Technavio, 2020).

Ahmad et al. (2022) publizierten eine Literaturrecherche zum Stand der Integration von IoT in sogenannte Serious Games. Darunter fallen Spiele, welche nicht ausschliesslich der Unterhaltung dienen, sondern auch Informationen und Bildung vermitteln sollen. Ahmad et al. (2022, S. 77) kommen zum Schluss, dass sich die Integration von IoT in solche Spielen noch in der frühen Anfangsphase befindet. Aus der Untersuchung gehen unter anderem die folgenden Herausforderungen und Probleme hervor (Ahmad et al., 2022, S. 78): Greifbare Benutzeroberfläche, Skalierbarkeit, Virtualisierung (Augmented- und Virtual Reality) und Architektur der Anwendung (Schichten-Architektur oder Game as a Service).

Nebst den technologischen Herausforderungen ist gemäss Kosa und Spronck (2018, S. 8) auf der Seite der Konsumierenden eine gewisse Zurückhaltung bezüglich der neuen Generation von Brettspielen vorhanden. Wie bereits in der Einleitung erwähnt, ist die soziale Interaktion zwischen den Mitspielenden ein grosser Teil des Erlebnisses beim Spielen. Kosa und Spronck (2018, S. 8) kommen zum Schluss, dass Brettspieler und Brettspielerinnen Angst davor haben, das klassische Gefühl beim Spielen eines Brettspiels zu verlieren. Diese Angst wird zusätzlich durch die negative Einstellung gegenüber Technologie im Allgemeinen verstärkt (Kosa & Spronck, 2018, S. 8). Hinzu kommt, dass ein gewisses Misstrauen gegenüber den Spieleherstellenden vorhanden ist, wenn es darum geht die verbaute Technologie zu reparieren oder instand zu halten (Kosa & Spronck, 2018, S. 8).

Diesen Vorbehalten stehen jedoch auch erwartete Vorteile bei der Verwendung von Technologie in Brettspielen gegenüber. So erwarten Brettspieler und Brettspielerinnen mehr Freude beim Spielen und eine vereinfachte Benutzung des Spiels (Kosa & Spronck, 2018, S. 8). Hinzu kommt, dass solche Spiele als Neuheit gelten und somit spannend und neuartig sind, was die menschliche Neugier antreibt (Kosa & Spronck, 2018, S. 8). Die positiven Auswirkungen der Integration von Technologie in Brettspielen werden in der gefundenen Literatur mehrmals hervorgehoben: Verringelter kognitiver Aufwand und dadurch eine verbesserte soziale Interaktion, immersives Spielerlebnis, neuartige Spielmodi oder örtlich getrenntes Spielen mit physischem Spielbrett (Bohn, 2004, S. 2; Floerkemeier & Mattern, 2006, S. 1; Hinske & Langheinrich, 2007, S. 17; Lima & Sarinho, 2019, S. 587; Magerkurth et al., 2004, S. 1). Dabei gilt es jedoch anzumerken, dass diese Arbeiten jeweils ein technologisch unterstütztes Brettspiel erarbeiten und die Validierung dieser Aussagen nicht breit abgestützt ist. Einzig in der Arbeit von Han et al. (2010, S. 1153) wird die Implementierung validiert, indem das RFID-unterstützte Lernspiel durch 29 Kinder unter Aufsicht der Lehrperson getestet wurde. Sie kommen zum Schluss, dass das modifizierte Spiel in den Kategorien Unterhaltungswert, pädagogischer Wert und Benutzungsfreundlichkeit deutlich besser abschneidet als die normale Variante des Spiels (Han et al., 2010, S. 1153). Um eine endgültige Aussage über die positiven und negativen Effekte der Integration von Technologie in Brettspiele zu treffen, benötigt es jedoch noch weitere Untersuchungen.

2.6 Evaluation von Technologien zum Tracking von Spielfiguren

In dem nachfolgenden Abschnitt wird eine geeignete Technologie zur Verfolgung der Position der Spielfiguren auf dem Spielbrett evaluiert. Das Tracking der Spielfiguren soll in der Anwendung Spielereignisse auslösen, um eine Interaktion mit der spielenden Person zu erreichen. Dabei steht neben der Genauigkeit der Positionierung auch die Möglichkeit zur unauffälligen Integration der Technologie in das physische Spielbrett im Vordergrund. Das Spielerlebnis soll von der Technologie schliesslich verbessert werden. Im Optimalfall ist die Hardware für die anwendenden Personen unsichtbar und verursacht keinen zusätzlichen Aufwand zur Installation. Hinzu kommt, dass die benötigten Komponenten im Verbrauchermarkt zu verhältnismässig tiefen Preisen erhältlich sein sollen.

Als Basis der Evaluation dient die Arbeit von Hinske und Langheinrich (2007), welche in ihrer Publikation bereits eine Technologie für die genaue Positionsverfolgung von Spielfiguren auf einem grossen Spiel-Schlachtfeld evaluierten. Da die Ergebnisse von Hinske und Langheinrich (2007) bereits älter als 10 Jahre sind, werden die Ergebnisse mit weiteren, aktuelleren Quellen ergänzt.

2.6.1 Ultra-Breitband

Zum Zeitpunkt der Untersuchung von Hinske und Langenheinrich (2007, S. 7) stellte die Grösse sowie der Preis von Ultra-Breitband-Sensoren ein Ausschlusskriterium dar. Hallek (2019) berichtet jedoch von deutlich tieferen Preisen und kleineren Dimensionen. Dennoch ist eine Verwendung im Rahmen eines Brettspiels auszuschliessen, da die Genauigkeit von bis zu 30 cm (Hallek, 2019) bzw. 5 bis 10 cm (Connell, 2015) keine Zuordnung einer Spielfigur zu einem spezifischen Spielfeld erlaubt.

2.6.2 Visuelles Tracking

Gemäss Hinske und Langenheinrich (2007, S. 8) existieren bereits Brettspiele basierend auf visueller Erkennungstechnologie. Dabei wird eine Kamera zur Verfolgung der Position verwendet. Sie verweisen dabei auf das Tracking anhand von Form und Farbe der Spielfiguren. So wäre es beispielsweise möglich, die blaue Spielfigur mit einer spielenden Person zu verknüpfen und via Koordinaten auf dem Spielbrett einem entsprechenden Spielfeld zuzuordnen. Denkbar ist jedoch auch die Verwendung von QR-Code basiertem Tracking der Figuren (*Tracking movement of game pieces on a board*, 2014). QR-Codes sind in der Lage die nötigen Spielerinformationen direkt wiederzugeben und sind für eine Maschine leicht zu erkennen (Sivakami, 2018, S. 4). Der Vorteil einer kamerabasierten Lösung ist, dass nur sehr wenige Modifikationen am Spielbrett benötigt werden. Die Kamera muss installiert und kalibriert werden und die Spielfiguren mit entsprechenden QR-Codes an der Oberseite gekennzeichnet werden. Der Nachteil einer visuellen Lösung ist jedoch, dass sichergestellt werden muss, dass die Kamera stets gleich kalibriert ist, um die erkannten Koordinaten einer Spielfigur dem richtigen Spielfeld zuzuordnen (Hinske & Langheinrich, 2007, S. 8). Hinzu kommt, dass der Aufnahmebereich der Kamera das gesamte Spielfeld abdecken muss und die Kamera stets eine Sichtlinie auf die Figuren haben muss. Das führt dazu, dass die nötige Vorrichtung sich nicht unauffällig in das Spielbrett integrieren lässt (Hinske & Langheinrich, 2007, S. 8). Für ein Brettspiel könnte beispielsweise die in Abbildung 2 gezeigte Vorrichtung verwendet werden.

Im Rahmen der Literaturrecherche konnte eine Arbeit gefunden werden, welche ein Gesellschaftsspiel mithilfe von visuellem Tracking entwickelt. Lee et al. (2005) stellen ein Kartenspiel vor, bei welchem eine Kamera mithilfe eines Spiegels unter einem Glastisch die aufgelegten Karten erkennt und es dem System ermöglichen darauf zu reagieren. Diese Arbeit liegt jedoch schon einige Jahre zurück und der Aufbau der Lösung ist umständlich.

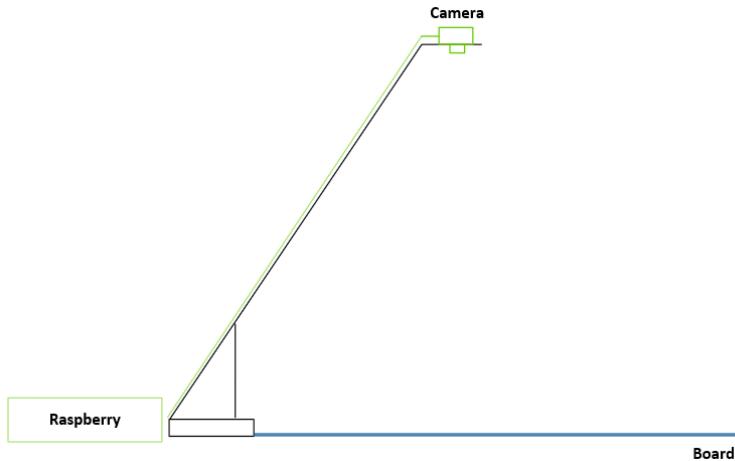


Abbildung 2 Kamera-Vorrichtung für das Tracking von Spielfiguren

2.6.3 Radio Frequency Identification

Zur Verwendung von Radio Frequency IDentification (RFID) wird ein Lesegerät sowie ein RFID-Transponder (engl. Tag) benötigt (Want, 2006, S. 27). Je nach Anwendungsbereich gibt es verschiedene Arten von Transpondern. Während aktive Transponder eine eigene Energiequelle benötigen und dadurch eine höhere Reichweite erzielen, benötigen passive Transponder keine Energiequelle und haben dementsprechend eine geringere Reichweite (Want, 2006, S. 25). Grundsätzlich gibt es für passive Systeme zwei Arten, um die Energie vom Lesegerät an den Tag zu senden: Magnetische Induktion und elektromagnetische Wellenerfassung (Want, 2006, S. 26). Nachfolgend wird auf die magnetische Induktion eingegangen, da diese für RFID im Nahfeldbereich verwendet wird.

Das Prinzip der magnetischen Induktion ist eine Methode, die in passiven RFID Systemen verwendet wird, um die Kommunikation zwischen einem Sensor und einem Tag zu ermöglichen (Want, 2006, S. 27). Die untenstehende Abbildung 3 zeigt die Kommunikation zwischen einem Lesegerät und RFID-Tag: Der Reader erzeugt ein wechselndes magnetisches Feld in seinem lokalen Bereich sobald ein Tag mit einer kleineren Spule in diesem Feld platziert wird, wird eine wechselnde Spannung in der Spule des Tags induziert (Want, 2006, S. 27). Diese Spannung kann verwendet werden, um den Chip des Tags zu betreiben (Want, 2006, S. 27). Daten werden von dem Tag zum Reader durch Lastmodulation übertragen, bei der der Transponder sein eigenes magnetisches Feld in Reaktion auf Änderungen des von der Tag-Spule gezogenen Stroms verändert (Want, 2006, S. 27). Diese Modulation kann verwendet werden, um Informationen zu codieren, wie zum Beispiel die ID des Tags, die danach vom Reader gelesen werden kann (Want, 2006, S. 27). Die Nahfeld-Kopplung hat einige physikalische Einschränkungen. Zum Beispiel die begrenzte Reichweite und eine Verringerung der für die Induktion verfügbaren Energie mit zunehmender Entfernung vom Reader (Want, 2006, S. 27). Diese Einschränkungen haben zu der Entwicklung von Fernfeld-Kommunikation für passive RFID-Systeme geführt, welche eine grössere Reichweite haben (Want, 2006, S. 27). Während Nahfeld-RFID

typischerweise eine Frequenz von 13.56 MHz verwendet, werden Fernfeld-RFID-Systeme in der Regel mit höheren Frequenzen zwischen 860 und 960 MHz betrieben (Want, 2006, S. 27).

RFID-Systeme sind nicht nur in der Lage Daten von einem Tag zu lesen. Bei gewissen Systemen ist es möglich auch Daten auf ein RFID-Tag zu schreiben (Want, 2006, S. 31). Dies ermöglicht eine Vielzahl von unterschiedlichen Einsatzgebieten wie zum Beispiel Logistik, Warenhandel, Gesundheitswesen oder Zutritts-Systeme (Want, 2006, S. 30–31). Für die Anwendung im Kontext eines Brettspiels eignet sich RFID besonders, da sich die Hardware unauffällig in das Spielbrett und die Figuren integrieren lässt (Hinske & Langheinrich, 2007, S. 9). Jedes Spielfeld wird mit einem RFID-Sensor versehen und jede Figur wird mit einem RFID-Transponder am Boden der Figur gekennzeichnet. Weitere Vorteile von RFID sind: Die benötigte Hardware ist preiswert und es können mehrere Figuren gleichzeitig auf einem Spielfeld erkannt werden (Hinske & Langheinrich, 2007, S. 17). Als Nachteile identifizieren Hinske und Langheinrich (2007, S. 17) die Schwierigkeit eine grosse Anzahl an Sensoren auf engem Raum zu verbauen, ohne dass es zu einer gegenseitigen Frequenzstörung kommt. In einer neueren Publikation von Lima und Sarinho (2019, S. 587) gelang es jedoch, eine RFID basierte Version des Spiels Tic-Tac-Toe zu erstellen, wo trotz der geringen Distanz zwischen den Feldern von keiner Signalstörung berichtet wird.

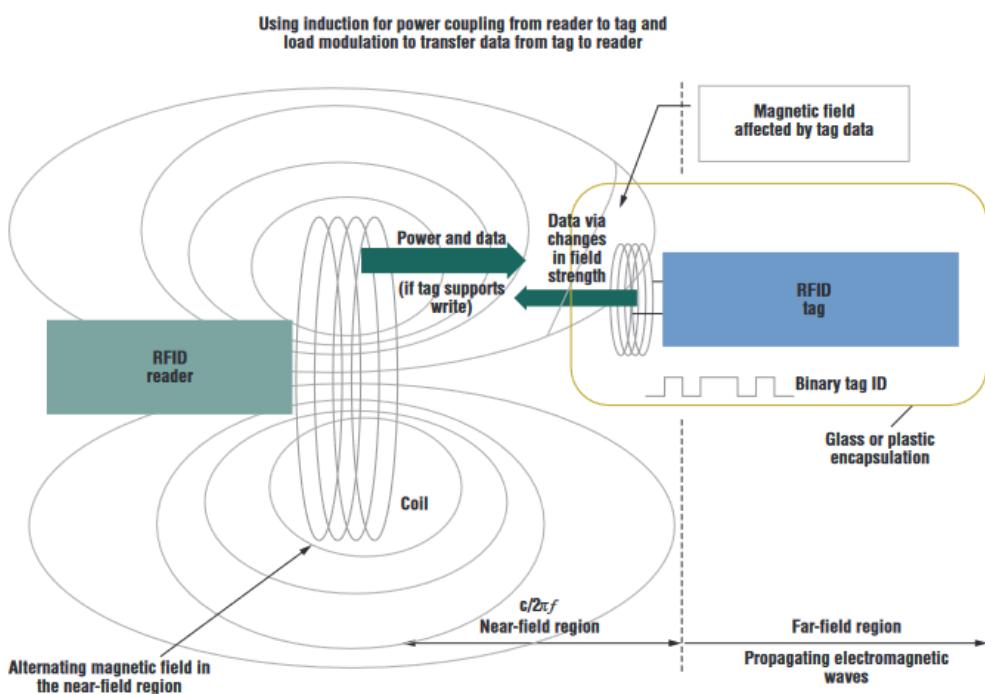


Abbildung 3 Nahfeld-RFID Kommunikation (Want, 2006, S. 26)

Dass die RFID-Technologie auch in weiteren Arbeiten im Zusammenhang mit Brettspielen eingesetzt wird, deutet zusätzlich auf die Eignung der Technologie für dieses Anwendungsgebiet hin. So konnten Han et al. (2010) mithilfe einer Sensor-Matrix von acht RFID-Sensoren mit je acht Antennen ein Spielbrett erstellen, welches es erlaubt vorgegebene Figuren mithilfe eines RFID-Tags nachzuzeichnen. Auch Tan und Rau (2015) entwickelten ein eigenes Brettspiel, welches mithilfe von RFID-Tags an den Spielfiguren und Sensoren auf dem Spielbrett ein immersives Spielerlebnis ermöglicht.

Es können jedoch nicht nur Brettspiele mithilfe von RFID umgesetzt werden: Flörkemeier und Mattern (2006) setzten ein Kartenspiel mithilfe von RFID um, wobei jeder Stapel eines Spielers mit einem Sensor und jede Spielkarte mit einem RFID-Transponder versehen wurde. Bereits vor knapp 20 Jahren stellte Bohn (2004) ein durch RFID unterstütztes Puzzle vor, in welchem die Teile durch den Kontakt mit Scanner weitere Informationen preisgeben. Zur selben Zeit wurde auch Tagaboo entwickelt. Ein Spiel, bei welchem RFID-Tags in Taschen versteckt werden und anschliessend von der spielenden Person mithilfe eines RFID-Handschuhs gefunden werden müssen, bevor die Zeit abläuft (Konkel et al., 2004).

Somit erfüllt RFID sämtliche Kriterien für die Verwendung bei der Umsetzung der vorliegenden Arbeit. Die Komponenten sind günstig erhältlich und es wurde bereits mehrfach bewiesen, dass sich die Technologie unauffällig in ein Brettspiel integrieren lässt. Hinzu kommt, dass sich im Internet und in der Wissenschaft zahlreiche Beiträge zum Thema RFID finden, was die Implementation einer solchen Lösung zusätzlich begünstigt.

3 Design

Nachdem im vorherigen Abschnitt die Grundlagen für das System-Design erarbeitet wurden, sowie RFID als geeignete Technologie zur Umsetzung dieser Arbeit evaluiert wurde, wird im folgenden Abschnitt die angestrebte Zielarchitektur definiert. Diese umfasst eine Beschreibung der Spielregeln von Monopoly, eine Übersicht über die Systemarchitektur und die Beschreibung der einzelnen Systemkomponenten, eine Planung für die einzelnen Hardwarekomponenten sowie Pläne für den zu erstellenden Prototyp. Dieses Kapitel stellt somit die Ausgangslage für die Umsetzung dar.

3.1 Spielregeln Monopoly

Monopoly ist ein Brettspiel für zwei bis acht Spielende mit dem Ziel als letzte Person solvent zu bleiben, während man versucht die anderen Spielenden in Zahlungsunfähigkeit zu treiben (*Monopoly Game Rules*, o. J.). Das Spielfeld besteht aus 40 Feldern unterteilt in folgende Kategorien:

Käufliche Felder		Nicht käufliche Felder	
Anzahl	Art	Anzahl	Art
22	Grundstück	1	Steuern
4	Bahnhof	1	Start
2	Versorgungswerk	1	Frei parken
		1	Gehe ins Gefängnis
		3	Ereignisfelder
		3	Gemeinschaftsfelder

Tabelle 1 Übersicht Felder Monopoly

Monopoly gilt als weltweit bestverkauftes privat patentiertes Brettspiel der Geschichte (*Monopoly Game Rules*, o. J.). Jede spielende Person beginnt mit einem Startkapital und kann die von ihr gewählte Spielfigur mithilfe eines Wurfes von zwei Würfeln auf dem Spielbrett fortbewegen (*Monopoly Game Rules*, o. J.). Beim Überqueren des Startfeldes erhält die spielende Person Geld von der Bank (*Monopoly Game Rules*, o. J.). Landet eine Spielerin oder ein Spieler auf einem Grundstück, das noch nicht im Besitz einer anderen Person ist, kann der Spieler das Grundstück kaufen (*Monopoly Game Rules*, o. J.). Wurde das Feld bereits durch eine andere spielende Person gekauft, muss eine entsprechende Gebühr bezahlt werden (*Monopoly Game Rules*, o. J.). Grundstücke sind in farbliche Gruppen unterteilt, wobei es von Vorteil ist wenn eine spielende Person alle Felder einer Farbe besitzt (*Monopoly Game Rules*, o. J.). Dadurch erhöht sich die zu bezahlende Gebühr aller Felder der Gruppe und die spielende Person kann diese durch Investitionen in Häuser und Hotels weiter erhöhen (*Monopoly Game Rules*, o. J.). Falls die Bank keine Häuser und Hotels mehr zum Verkauf hat, müssen die Spielenden warten, bis jemand bereits gekaufte Häuser oder Hotels wieder an die Bank verkauft, um an Geld zu gelangen (*Monopoly Game Rules*, o. J.).

Auf dem Spielbrett befinden sich auch nicht käufliche Felder. Landet eine Spielfigur auf dem «Gehe ins Gefängnis» Feld, muss diese Person für maximal drei Spielzüge ins Gefängnis (*Monopoly Game Rules*, o. J.). Durch eine entsprechende «Du kommst aus dem Gefängnis frei» Karte oder das Würfeln eines Pasch kann die Person frühzeitig das Gefängnis verlassen (*Monopoly Game Rules*, o. J.). Hinzu kommen weitere nicht käufliche Felder wie die Ereignis- und Gemeinschaftsfelder. Bei diesen muss die spielende Person eine Karte aus dem entsprechenden Stapel ziehen und die darauf befindlichen Anweisungen befolgen (*Monopoly Game Rules*, o. J.). Landet eine Spielfigur auf dem «Frei Parken» Feld, muss keine Gebühr an andere Spielende bezahlt werden jedoch auch kein Grundstück gekauft werden (*Monopoly Game Rules*, o. J.). Sollte eine spielende Person in Geldnot geraten, können Grundstücke ohne Häuser und Hotels an andere Spielende verkauft werden (*Monopoly Game Rules*, o. J.). Häuser und Hotels können für den halben Preis an die Bank verkauft werden (*Monopoly Game Rules*, o. J.). Darüber hinaus können bereits gekaufte Grundstücke von den Spielenden mit einer Hypothek der Bank belehnt werden, um kurzfristig Geld zu erhalten (*Monopoly Game Rules*, o. J.).

3.2 Hardwarekomponenten

Ein Hauptfokus des System-Designs ist die Integration von RFID-Technologie für das Tracking der Spielfiguren. Im folgenden Abschnitt wird die dafür verwendete Hardware vorgestellt respektive evaluiert.

3.2.1 RFID-Module

Aus dem vorherigen Abschnitt geht RFID als geeignete Technologie zum Nachverfolgen der Spielfiguren-Positionen hervor. Nun wird ein Vergleich der auf dem Markt erhältlichen RFID-Lesegeräte erstellt, um das am besten geeignete Modul für die vorliegende Arbeit zu evaluieren. Die einzelnen Module unterscheiden sich in Grösse, Form, Preis und Funktionalität. Damit die Module im Rahmen dieses Projektes verwendet werden können, sollten folgende Kriterien erfüllt sein:

1. Kompatibel mit Arduino / Raspberry Pi: Die Inbetriebnahme mit einem der beiden Mikrokontrollern soll möglichst einfach sein. Am besten stellt das Lesegerät eine SPI-Schnittstelle zur Verfügung.
2. Kleine Dimensionen: Ein Spielfeld misst ca. 4 cm x 6.5 cm. Das Lesegerät soll diese Dimensionen im besten Fall nicht überschreiten.
3. Tiefer Preis: Insgesamt sollen bis zu 16 Lesegeräte verwendet werden. Der Stückpreis sollte somit wenige Schweizer Franken nicht überschreiten, um die Projektkosten möglichst tief zu halten.
4. Erhältlichkeit: Die Lesegeräte sollen trotz der momentanen Lieferengpässe im Elektronik- und vor allem in Chip-Bereich innert maximal sechs Wochen verfügbar sein.

5. Vorhandene Frameworks und Tutorials: Die Kommunikation zwischen dem Lesegerät und RFID-Tag beinhaltet viele technisch komplexe Vorgänge. Im Rahmen dieser Arbeit kann diese Kommunikation nicht von Grund auf implementiert werden. Deshalb ist es zwingend, dass zum entsprechenden Lesegerät eine Software-Bibliothek besteht, welche die Kommunikation abstrahiert, und eine Verwendung des Lesegeräts vereinfacht.

Die gängigsten RFID Lesegeräte im privaten Verbrauchermarkt sind die vier, in der Abbildung 4 ersichtlichen Modelle RDM6300, MFRC522, PN532 und PN5180 (Playful Technology, 2018).

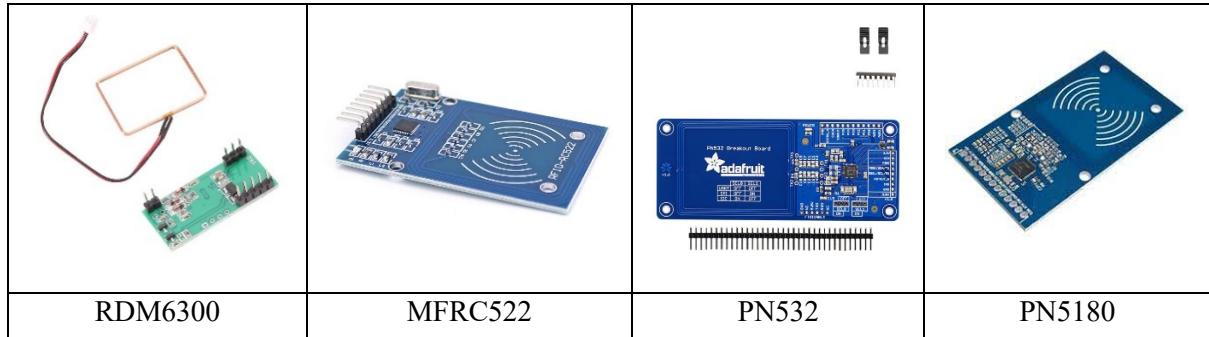


Abbildung 4 RDM6300 mit Antenne (Rdm6300 ID Kartenleser Modul Rfid RF Modul Uart Serieller Ausgang, o. J.), MFRC522 (RFID RC522 Reader Writer, o. J.), PN532 (PN532 NFC/RFID controller, Adafruit, o. J.), (arduino-rfid-PN5180, 2018/2022)

RDM6300: Der RDM6300, ein kostengünstiges und kleines Modul, arbeitet im 125-KHz-Frequenzbereich und verwendet das EM4100-Protokoll, eines der am häufigsten verwendeten Formate für RFID-Tags (Playful Technology, 2018). Mit einer Grösse von 3 cm x 2 cm ist es klein genug, um in diesem Projekt eingesetzt zu werden. Das Modul verfügt über eine externe Antenne und die Verbindung zum Mikrokontroller geschieht via Hardware Serial Verbindung (Playful Technology, 2018). Die Anzahl solcher Anschlüsse in einem Arduino sind begrenzt und die gleichzeitige Verwendung mehrerer RDM6300 kann gemäss Playful Technology (2018) jedoch zu Problemen führen. Hinzu kommt, dass das Modul lediglich über dein Ein- und Austritt von Tags im Lesebereich informiert (Playful Technology, 2018). Eine Abfrage, ob ein Tag zu einem beliebigen Zeitpunkt gelesen wird, ist nicht möglich (Playful Technology, 2018).

MFRC522: Das MFRC522 RFID-Modul ist eine beliebte Wahl für RFID-Projekte – Nicht zuletzt weil es mit 0.60 CHF pro Stück sehr preiswert ist (Playful Technology, 2018). Die verbreitete Nutzung führt dazu, dass zur Verwendung des Modules eine etablierte Bibliothek besteht und es zahlreiche Ressourcen im Internet gibt (Playful Technology, 2018). Das Lesegerät misst 6 cm x 4 cm und kann Daten von und zu RFID-Karten und -Tags lesen und schreiben und mit Mikrocontrollern über das SPI-Protokoll kommunizieren (Playful Technology, 2018). MFRC522 Lesegeräte arbeiten im 13.56 MHz-Frequenzbereich, haben eine Reichweite von 4-5 cm und verwenden den ISO 14443A-Standard, der auch von vielen RFID-Karten und -Tags verwendet wird (Playful Technology, 2018). Für dieses Modul spricht zusätzlich, dass es von Lima und Sarinho (2019, S. 588) in ihrer «Open Hardware RFID

Architecture for Electronic Board Games» verwendet wird und für diesen Verwendungszweck als geeignet befunden wurde.

PN532: Das PN532 Modul bietet zwar eine höhere Reichweite als die kleineren Module, ist jedoch mit 60 CHF pro Stück für die Verwendung in diesem Projekt zu teuer (Playful Technology, 2018). Hinzu kommt, dass die Dimensionen (10 cm x 5 cm) zu gross sind.

PN5180: Dieses Modul wird von Playful Technology (2018) empfohlen, da es ein neues Modul ist und dadurch eine verbesserte Reichweite wie das MFRC522 Modul bietet, wobei es nur minimal grösser ist. Jedoch ist dieses Lesegerät nur schwer erhältlich und mit 20 CHF pro Stück zu teuer, um in diesem Projekt eingesetzt zu werden.

Aus der Evaluation geht hervor, dass das MFRC522-Lesegerät am geeignetsten für das vorliegende Projekt ist. Der niedrige Preis und die relativ kleinen Dimensionen sowie die Verfügbarkeit einer etablierten Library zusammen mit zahlreichen Internetressourcen für die Inbetriebnahme des Moduls sind die ausschlaggebenden Punkte für diese Entscheidung. Während die Module PN532 und PN5180 zu teuer sind, um im Rahmen eines Brettspiels verbaut zu werden, ist der Betrieb von mehreren RDM6300 Lesern mit einem Arduino fehleranfällig, weshalb die Module im Rahmen dieses Projekts nicht eingesetzt werden.

3.2.2 RFID-Sensoren und Mikrokontroller

Die Grundlagen für das Hardware-Layout dieses Projektes stammt aus der Arbeit von Lima und Sarinho (2019). In ihrer Arbeit erarbeiten sie eine «Open Hardware RFID Architecture for Electronic Board Games» mithilfe eines Mikrocontrollers und mehreren MFRC522 Modulen (Lima & Sarinho, 2019, S. 588). Als Kommunikationsprotokoll verwenden sie MQTT und als Hauptkomponente wird ein Raspberry Pi 3 verwendet (Lima & Sarinho, 2019, S. 588). Dabei wird festgehalten, dass nicht mehr als zwei RFID Module gleichzeitig an einem Mikrocontroller vom Typ LoLin NodeMCU 1.0 betrieben werden kann (Lima & Sarinho, 2019, S. 588). In diesem Projekt sollen jedoch gleichzeitig bis zu 16 Felder überwacht werden, dies würde bedeuten, dass rund acht Mikrocontroller nötig sind. Die vollständige Version von Monopoly hat sogar 40 Felder und benötigte somit 20 solche Mikrocontroller. Aus diesem Grund wird die von Lima und Sarinho (2019) vorgestellte Architektur für das vorliegende Projekt angepasst. Aus dem Video von Playful Technology (2017) geht hervor, wie gleichzeitig vier RFID RC522 Module mithilfe eines Arduino Uno betrieben werden können. Damit halbiert sich die Anzahl der benötigten Mikrocontroller. Eine weitere Verbesserung ist mit der Verwendung von einem noch leistungsfähigeren Mikrocontroller wie zum Beispiel dem Arduino Mega zu erwarten. Im Rahmen der Recherche konnte dazu eine entsprechende Arbeit gefunden werden, die mithilfe eines Arduino Mega zehn RFID RC522 Module gleichzeitig betreibt (MIRACLE TECH, 2020). Damit wären für 40 Spielfelder nur noch vier Mikrocontroller nötig. Da die Arduino Mega jedoch angeschafft werden müssten und bereits mehrere Arduino Uno im Rahmen des Projekts verfügbar sind, wird die Architektur

anhand eines Arduino Uno erarbeitet. Die Stromversorgung des Arduino Uno wird via USB sichergestellt.

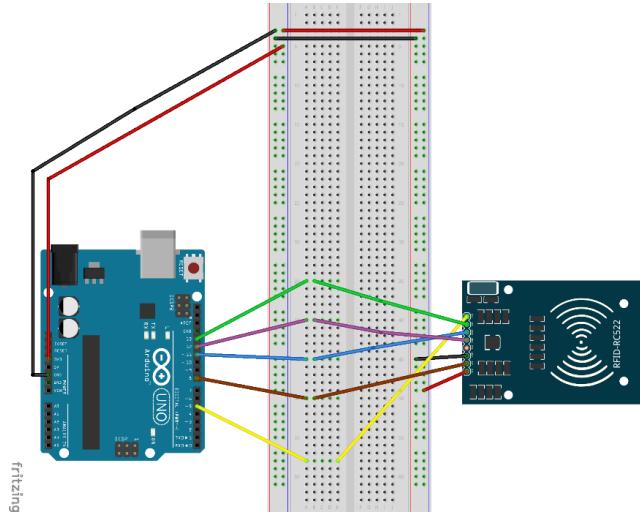


Abbildung 5 Konfiguration eines MFRC522 mit Arduino Uno

Abbildung 5 zeigt, wie ein einzelner MFRC522 mit einem Arduino Uno verbunden wird. Dabei wird das Serial Peripheral Interface (SPI) vom Arduino mithilfe eines sogenanntes Steckbretts angeschlossen. Die genauen Verbindungen kann der untenstehenden Tabelle 2 entnommen werden.

Pin	Farbe	Funktion
5	Yellow	Serial Data (SDA)
8	Brown	Reset
11	Blue	Master Out Slave In (MOSI)
12	Purple	Master In Slave Out (MISO)
13	Green	Serial Clock (SCK)
3.3V	Red	Stromversorgung (+)
GND	Black	Stromversorgung (-)

Tabelle 2 Pin Layout MFRC522 mit Arduino Uno (In-Depth, 2018)

SDA: Signaleingang

Reset: Sollte das Modul keine Antwort mehr liefern oder ein Fehler tritt auf, kann es hiermit zurückgesetzt werden.

MOSI: Transportiert Daten vom Arduino zum Modul

MISO: Transportiert Daten vom Modul zum Arduino.

SCK: Takteleitung für die SPI-Kommunikation

Wie Abbildung 6 zeigt, kann die vorgestellte Konfiguration wie bereits erwähnt auf vier MFRC522 Module ausgeweitet werden. Hierbei dient die SDA-Verbindung (gelb) als sogenannter «Slave Select Pin» womit gesteuert wird, mit welchem Modul gerade kommuniziert wird. Damit werden die Pins zwei bis fünf des Arduino Uno belegt.

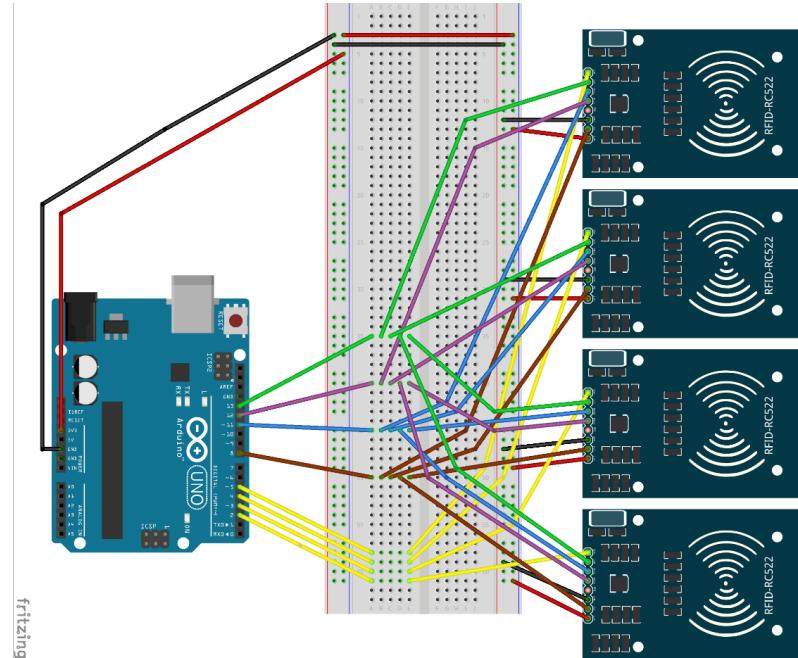


Abbildung 6 Konfiguration von vier MFRC522 mit Arduino Uno

3.2.3 RFID-Tags

Die verwendeten RFID-Tags sollen passiv und somit ohne Stromversorgung funktionieren. Falls möglich sollen sie klein sein, damit sie einfach in die Spielfigur integriert werden können. Im Rahmen einer Internetrecherche wurden die in der Tabelle 3 abgebildeten Tags evaluiert.

		
Mifare Classic	NTag213	NTag215
Von links nach rechts:		
<i>Abbildung 7 RFID Badge (2N RFID-Badge Mifare Classic 1k RFID Badge, 13.56 MHz, o. J.)</i>		
<i>Abbildung 8 Ntag213 (50pcs Ntag213 Nfc Tags 13.56mhz Iso14443a, o. J.)</i>		
<i>Abbildung 9 NTag215 (NTag215 Mini Rfid NFC Tag, o. J.)</i>		

Tabelle 3 RFID-Tags

- **RFID-Badge Mifare Classic:** Dieser Tag wird mit den bereits evaluierten RFID-Sensoren MFRC522 zusammen verkauft. Eigentlich als Badge gedacht, könnte dieser Tag im Rahmen des Brettspiels dennoch am Boden einer Spielfigur angebracht werden. Mit 4 cm x 3 cm ist er zwar eher gross, wobei er durch die Plastik-Hülle jedoch auch robuster ist. Im Internet finden sich keine genauen Spezifikationen zum Speicherplatz oder der Lesedistanz. Die Integration in eine Spielfigur benötigt sicherlich Anpassungen an der Figur.
- **NTag213:** Mit einem Durchmesser von 2.5 cm und einer sehr geringen Dicke, ist dieser Tag besonders geeignet, da er sich einfach an die Unterseite einer Spielfigur anbringen lässt, ohne dass eine Anpassung nötig wäre. (*50pcs Ntag213 Nfc Tags 13.56mhz Iso14443a*, o. J.). Die Lesedistanz beträgt 1 cm – 5 cm und der Speicherplatz 144 Bytes (*50pcs Ntag213 Nfc Tags 13.56mhz Iso14443a*, o. J.).
- **NTag215:** Dieser Tag ist mit 1.2 cm Durchmesser sehr klein und bietet dennoch 504 Bytes an Speicherplatz (*NTag215 Mini Rfid NFC Tag*, o. J.). Er ist ein wenig dicker wie die NTag213, weshalb es für die Integration in eine Spielfigur sicherlich eine Anpassung benötigt.

Alle vorgestellten Tags werden im Rahmen des Prototyps getestet, damit festgestellt werden kann, ob die Reichweite auch durch das Spielbrett hindurch ausreicht. Die Evaluation erfolgt anhand der Aufzeichnung von 50 Spielzügen pro RFID-Tag. Anhand dessen kann die Erkennungsrate berechnet werden, wodurch der zu verwendende Tag bestimmt wird.

Als Bezahlkarte wird eine, ebenfalls mit dem MFRC522 mitgelieferte, RFID-Karte verwendet. Die weisse Karte im Kreditkartenformat ist kompatibel mit der 13.56 MHz-Frequenz und verfügt über einen Speicher von 1 Kilobyte, wovon 768 Bytes frei sind. Der Lesebereich liegt zwischen 2 cm und 10 cm, was für die Verwendung in diesem Projekt ausreicht.

3.2.4 Zentrale Rechnerkomponente

Die Signale der verschiedenen Mikrocontroller werden von einer zusätzlichen Rechnerkomponente verarbeitet. Diese ist ebenfalls zuständig für die Darstellung der grafischen Benutzeroberfläche sowie den Betrieb der Spiellogik. Zusätzlich wird ein einzelner RFID MFRC522 an diese Komponente angeschlossen, welcher im Spiel als Bezahlstation dient. Im Rahmen dieser Arbeit wird hierfür ein Raspberry Pi 3B+ verwendet. Die Stromversorgung des Raspberry wird mittels eines Netzteils sichergestellt. Die Verbindung der zwischen dem RFID-Modul und dem Raspberry Pi ist in Abbildung 10 ersichtlich.

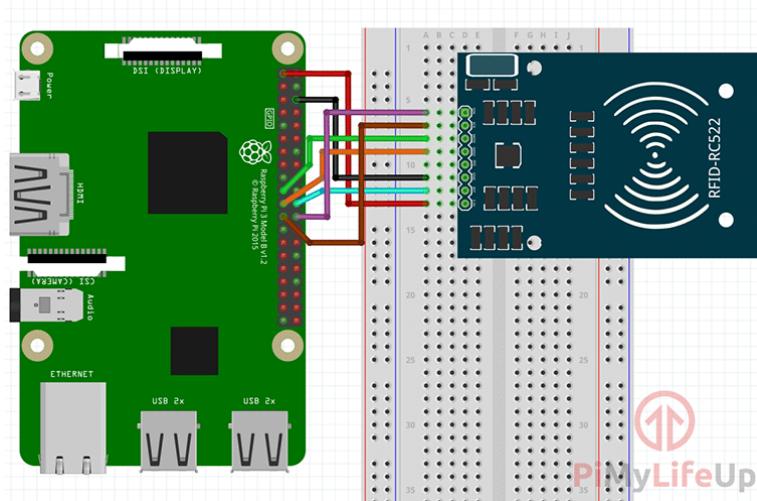


Abbildung 10 Konfiguration von einem MFRC522 mit Raspberry Pi 3B+

Aus der untenstehenden Tabelle 3 können wiederum die einzelnen Verbindungen der Pins entnommen werden. Eine Beschreibung der Funktionen kann dabei dem vorangehenden Kapitel entnommen werden.

Pin	Farbe	Funktion
24	● (Purple)	Serial Data (SDA)
22	● (Cyan)	Reset
19	● (Green)	Master Out Slave In (MOSI)
21	● (Orange)	Master In Slave Out (MISO)
23	● (Brown)	Serial Clock (SCK)
1	● (Red)	Stromversorgung (+)
6	● (Black)	Stromversorgung (-)

Tabelle 4 Pin Layout RFID RC522 mit Raspberry Pi 3B+

3.2.5 LED-Lampen

Damit die Benutzenden ein Feedback erhalten, ob die Spielfigur auf dem entsprechenden Feld vom System erkannt wurde, wird pro Sensor (und somit pro Feld) eine LED eingesetzt. Diese soll aufleuchten, sobald das RFID-Tag vom Sensor erkannt wurde. Spielende können somit direkt erkennen, sollte die Figur nicht auf Anhieb erkannt werden und sind somit in der Lage, die Spielfigur eventuell erneut zu platzieren. Jeweils vier LEDs werden von einem Arduino Uno bedient und separat pro Sensor angesteuert. Dafür wird die in Abbildung 11 dargestellte Konfiguration verwendet.

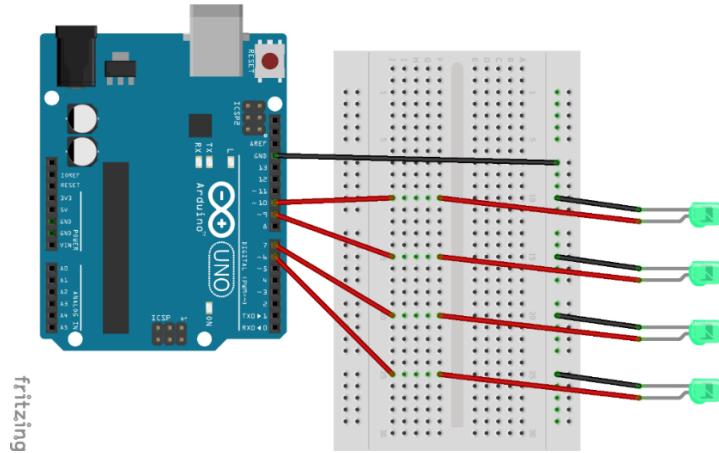


Abbildung 11 Konfiguration der LEDs mit dem Arduino Uno

Die Aufteilung auf vier unterschiedliche Pins ermöglicht es, die LEDs unabhängig voneinander aufzuleuchten zu lassen. Dafür werden die freien Pins 6, 7, 9 und 10 des Arduino Unos verwendet. Zusätzlich ist ein Erdungs-Pin erforderlich. Als LED werden 5 mm LEDs in grün verwendet.

3.2.6 Display

Folgende Anforderungen muss das Display erfüllen: Es soll möglichst schlanke Dimensionen haben, eine Touch-Funktion bieten und wenn möglich über einen HDMI-Anschluss verfügen. Für dieses Projekt wird der sieben Zoll grosse LC-Display Joy-it RB-LCD7-2 von Joy-IT verwendet. Wie in Abbildung 12 zu sehen ist, werden für den Betrieb nur zwei Kabel benötigt: Ein USB-Kabel für die Touch-Funktion und ein HDMI-Kabel für den Signaleingang.

3.2.7 Stromversorgung

Um die Mikrokontroller sowie das Display mit genügend Strom zu versorgen, wird ein aktiver USB-Hub verwendet. Dies ist nötig, da das Raspberry Pi nicht alle Mikrokontroller und das Display mit genügend Strom versorgen kann, wenn das System unter Last ist. Der in Abbildung 13 gezeigte USB-Hub verfügt pro Port über eine Leistung von bis zu 900 mA, wobei insgesamt maximal 2'000 mA Ausgangsleistung verfügbar ist.



Abbildung 12 (links) LC Display von Joy-IT welcher im Rahmen dieses Projekts verwendet wird (Joy-it RB-LCD7-2 (Display), o. J.)

Abbildung 13 (rechts) Aktiver USB-HUB (Xystec Aktiver USB-3.0-Hub (USB A), o. J.)

3.3 System Design

Der letzte Abschnitt zeigt die verwendete Hardware für die Umsetzung des Projekts. Nachfolgend wird eine umfassende System-Architektur erarbeitet. Ausgehend von den Anforderungen an die Anwendung wird mithilfe von DDD die Domäne beschrieben und anhand von Klassen- und Komponentendiagrammen visualisiert. Zusätzlich zeigt die System-Architektur auf, welche Komponenten miteinander kommunizieren. Abschliessend werden die Pläne für den Prototypen gezeigt, welche im Anschluss für die Umsetzung massgebend sind. Wie bereits erwähnt liegt der Fokus dieser Arbeit auf der Integration von RFID-Technologie in ein System, statt auf der kompletten Implementierung der Spiellogik. Deshalb wird die Anwendung lediglich das Starten und Beenden des Spiels, das Kaufen von Grundstücken sowie den damit einhergehenden Transaktionen beim anschliessenden Landen auf dem Feld und das Ziehen von Ereigniskarten abbilden.

3.3.1 Funktionsübersicht

Die zentrale Funktionalität der Applikation ist es, auf die Bewegung der Spielfigur innert nützlicher Zeit zu reagieren und das Resultat des neuen Zustands auf dem Spielbrett via Display wiederzugeben. Das untenstehende Use-Case-Diagramm in Abbildung 14 zeigt, dass im Wesentlichen fünf Use-Cases bestehen. Die zentralen Use Cases für das Spielgeschehen sind dabei der Grundstückkauf (Buy property), der Kauf von Häusern und Hotels (Buy upgrades) sowie das Bezahlen von Transaktionen (Pay with card). In der Abbildung 14 ist zu sehen, dass dafür jeweils die Spielfigur bewegt werden muss, was wiederum voraussetzt, dass ein Würfel geworfen wurde. Durch das Bewegen der Spielfigur können jedoch noch andere Use-Cases ausgelöst werden. Je nachdem wo die Spielfigur landet, muss eine Karte gezogen werden oder eine Gebühr an andere Spielende bezahlt werden. Die anwendende Person interessiert sich darüber hinaus für die aktuellen Kontostände und möchte wissen, welche Spielerin oder welcher Spieler als nächstes an der Reihe ist. Zusätzlich soll die Anwendung durch die Spielenden gestartet und beendet werden können. Die Anwendung verfügt somit über eine überschaubaren Funktionsumfang. Umso wichtiger ist es, dass die Schnittstelle zwischen den Benutzenden und der

Applikation möglichst simpel und frei von technischen Hürden umgesetzt wird. Die Verwendung der vorgestellten Lösung soll keinesfalls technisches Vorwissen benötigen. Gängige Prozesse wie das Starten des Spiels oder der Kauf eines Grundstücks sollen möglichst einfach und übersichtlich gestaltet werden.

Die gezeigten Use-Cases sollen durch die Anwendung und die vorgestellte Architektur unterstützt werden. Das System soll nach dem Starten automatisch erkennen welche Spielfigur auf welches Spielfeld gezogen wird. Zieht eine spielende Person die Figur auf ein Grundstück, welches einer anderen Spielerin oder Spieler gehört, wird die Person zur Zahlung aufgefordert. Ist das Grundstück ohne Besitzerin oder Besitzer, werden die Informationen über das Grundstück angezeigt und es besteht die Möglichkeit für den Kauf des Feldes. Sollte die Figur auf einem Ereignisfeld landen, zieht das System eine digitale Karte und zeigt den Kartentext an. Es ist in der Lage, die darauffolgende Aktionen wie die Bezahlung einer Gebühr, das Erhalten von Geld oder das Aufbewahren einer «Du kommst aus dem Gefängnis frei» Karte abzubilden.

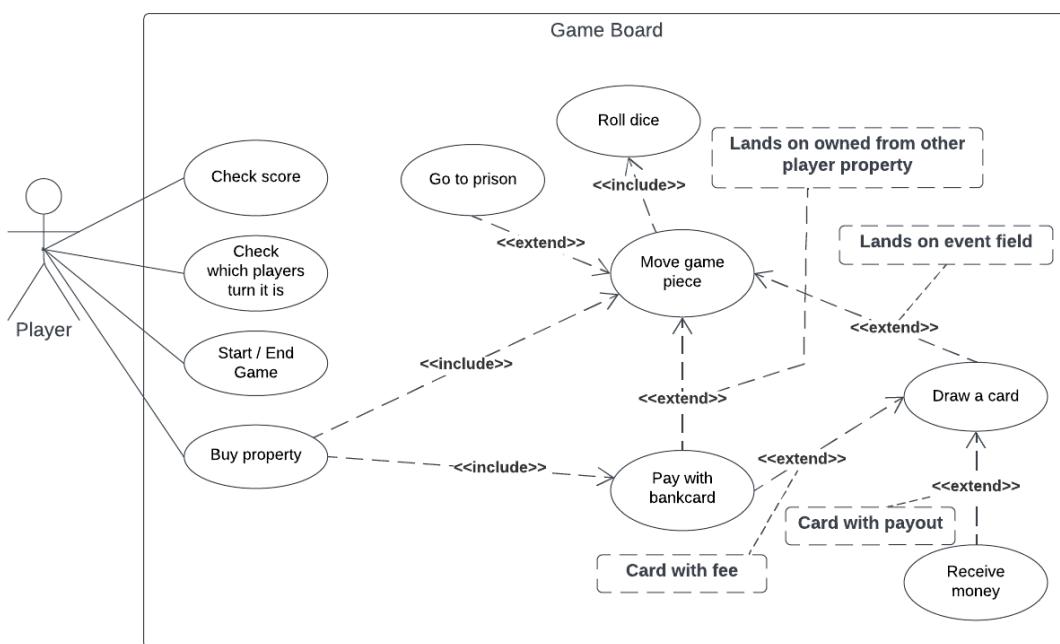


Abbildung 14 UML-Use-Case Diagramm

3.3.2 Domänen-Design

Als erster Schritt wird die gemeinsame Sprache der Monopoly-Domäne erarbeitet. Und die darin enthaltenen Entitäten, Value Objects und Aggregates fett hervorgehoben. Generell wird im Kontext von Monopoly von einer **Spielerin / einem Spieler (Player)** bzw. einer Spielfigur gesprochen. Mehrere Teilnehmende spielen zusammen das **Spiel (Game)**, indem Sie Figuren auf **Spielfeldern** des **Spielbretts (Board)** bewegen. Dadurch registrieren die **RFID-Sensoren (Sensor)** die Tags auf den einzelnen Feldern. Dabei gibt es unterschiedliche Arten von Feldern: **Grundstücke (PropertyField)**,

Ereignisfelder (EventField), und sonstige Felder (StandardField). Will eine teilnehmende Person ein Grundstück kaufen, muss eine entsprechende Zahlung getätigt werden (**PendingPropertyTransfer**). Bei Ereignisfeldern muss die spielende Person entweder eine Steuer bezahlen oder eine **Spielkarte (Card)** ziehen kann. Die Spielkarte kann ein **Spielereignis (GameEvent)** darstellen, welches wiederum eine **Gutschrift** oder **Zahlung** bedeuten kann. Allenfalls zieht die spielende Person auch eine «Du kommst aus dem Gefängnis frei Karte». Grundstücke können mithilfe von **Häusern und Hotels (PropertyUpgrade)** aufgewertet werden, wodurch sich der zu **bezahlende Preis (FinancialDetails)** verändert. Eine spielende Person hat ein **Bankkonto (Account)**, auf welches die Einnahmen und Ausgaben gebucht werden. Mithilfe einer zugehörigen RFID-Karte können spielende Personen **Transaktionen (Transaction)** an der Bezahlstation bezahlen.

Wie bereits erwähnt werden im Rahmen dieser Arbeit nicht alle Spielfunktionen umgesetzt. Nachfolgend wird die Domäne anhand des von Vernon (2016, S. 112) vorgestellten Event Storming erarbeitet und als erstes die relevanten Events identifiziert:

- game_started: Das Spiel wurde gestartet und es ist klar, wie viele spielende Personen teilnehmen.
- game_ended: Das Spiel wurde durch eine teilnehmende Person beendet.
- tag_read: Ein RFID-Tag wurde von einem Sensor auf einem Feld erkannt.
- card_read: Eine RFID-Karte wurde an der Bezahlstation erkannt.
- player_on_owned_field: Eine Spielfigur ist auf einem Feld mit Besitzer gelandet.
- player_on_unowned_field: Eine Spielfigur ist auf einem Feld ohne Besitzer gelandet.
- transaction_created: Eine Transaktion wurde erstellt und muss bezahlt werden.
- transaction_resolved: Eine Transaktion wurde bezahlt.
- lap_finished: Eine Spielfigur hat das Spielfeld einmal umrundet.
- card_drew: Eine Ereigniskarte wurde gezogen.
- game_event_with_payout_accepted: Ein Game Event mit einer Auszahlung an die spielende Person wurde akzeptiert.
- game_event_with_fee_accepted: Ein Game Event mit einer zu bezahlenden Gebühr wurde akzeptiert.

Diese Events können nun den entsprechenden Aggregates zugeordnet werden.

- **Game:** game_started, lap_finished, game_ended, game_event_with_fee_accepted, card_drew, player_on_unowned_field, game_event_with_payout_accepted
- **Transaction:** transaction_created, transaction_resolved
- **Sensor:** tag_read, card_read

Anhand dieser Aggregates, der beschriebenen Use Cases und der Domänen-Events kann nun die Domäne modelliert werden. Alle UML-Klassendiagramme finden sich in den Anhängen 2-3.

Sensor: Diese Subdomäne repräsentiert das physische Spielbrett (GameBoard). Alle Sensoren sind einem Gerät zugewiesen und können ein TagReadEvent auslösen, sobald sie ein RFID-Tag erkannt haben. Die darin enthaltene Kombination von deviceId und sensorId erlaubt eine eindeutige Zuordnung zu einem Spielfeld auf dem Brett. Die tagId repräsentiert eine Spielfigur und somit eine spielende Person. Nebst den Sensoren pro Spielfeld gibt es einen weiteren Sensor für die Bezahlstation, welcher innerhalb dieses Kontexts überwacht wird und beim Kontakt mit einer Zahlkarte ein CardReadEvent auslöst.

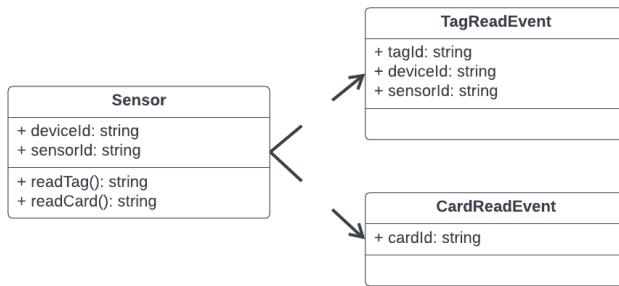


Abbildung 15 Entitäten Sensor Subdomäne

Player: Die Teilnehmenden werden mithilfe der Player Entität abgebildet. Jeder Player kennt seine Position auf dem Spielbrett und verfügt über einen Account (Bankkonto). Die Player-Entität wird in der gesamten Applikation mehrfach referenziert. Beispielsweise als Eigentümer von Grundstücken, oder im Kontext von Spielkarten.

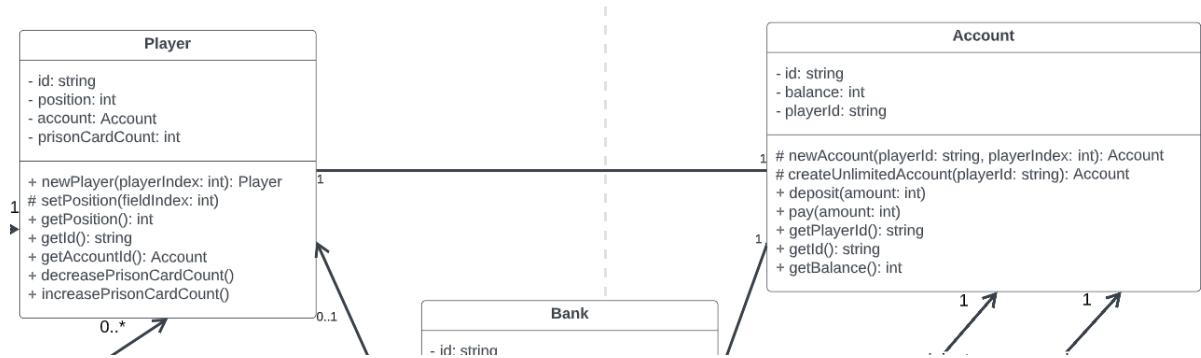


Abbildung 16 Player und Account Entität

Game: Dies bildet die Core-Domäne ab und ist das führende Aggregate, welches für den Lebenszyklus des Spiels zuständig ist. Hier wird das Spiel mit einer entsprechenden Anzahl spielenden Personen gestartet (game_started) und irgendwann beendet (game_ended). Zentrale Funktionen wie MovePlayer, BuyProperty und DrawCard bilden den Grossteil der Spielmechanik ab und lösen dabei relevante Domänen-Events aus. Die ausgelösten Events können dem Anhang 4 entnommen werden. Es ist anzumerken, dass sich ein Game sowohl die gerade pendente Spielkarte (falls eine solche gezogen wird) und einen pendenten Grundstückskauf speichert. Damit kann die asynchrone Natur des Spiels

abgebildet werden. Beispielsweise wenn eine Person eine Karte zieht, den digitalen Kartentext liest und nach dem Akzeptieren der Karte auf die Bezahlung der erstellten Transaktion gewartet wird.

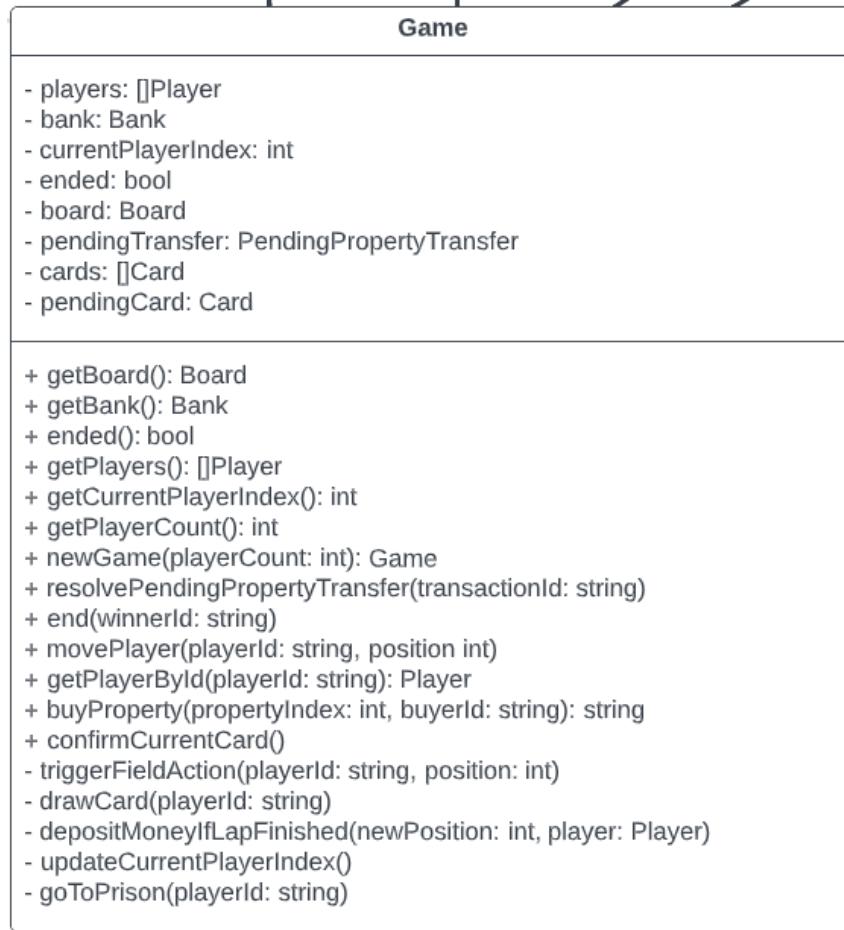


Abbildung 17 Aggregate Game

Board: Die Board Entität verwaltet alle auf dem Spielbrett befindlichen Felder und regelt den Zugriff auf diese. Jedes Game Objekt hat eine Beziehung zu einem Board. Dadurch ist es in Zukunft möglich auch Spiele mit mehr Feldern oder anderen Feldkombinationen einfach abzubilden.

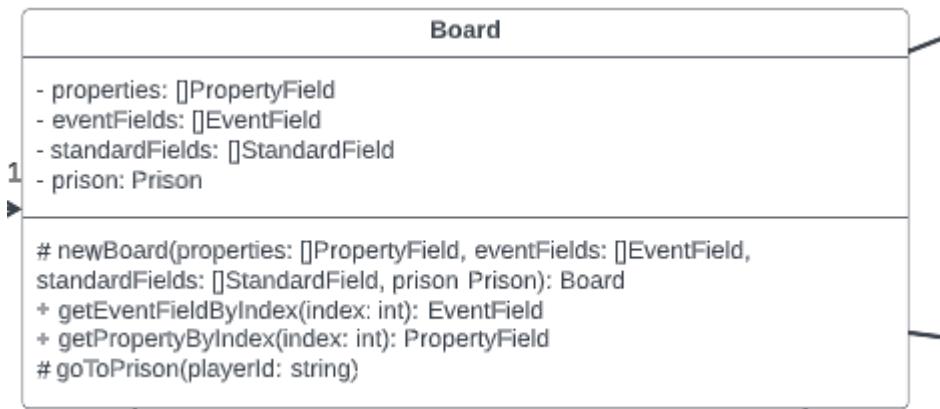


Abbildung 18 Board Value Object

Property: Alle Grundstücke, welche auf dem Spielbrett von einer Person gekauft werden können, werden hiermit abgebildet. Hier wird die Zuordnung zwischen Grundstück und spielenden Personen sichergestellt. Die PropertyField-Entität wird vom Board verwaltet. Das Game Aggregate wird entsprechende Events beim Kaufinteresse der spielenden Person erzeugen (property_transfer_created). Die pendente Transaktion wird mithilfe der PendingPropertyTransaction-Entität abgebildet. Damit wird sichergestellt, dass das Grundstück erst nach Bezahlung an die spielende Person übertragen wird. Grundstücke verfügen über Upgrades in der Form von Häusern und Hotels sowie die Information über die entsprechenden Preise bei den unterschiedlichen Ausbaustufen. Die ownerId des PropertyFields sowie die buyerId der PendingPropertyTransaction stellen die Verbindung zum Player her.

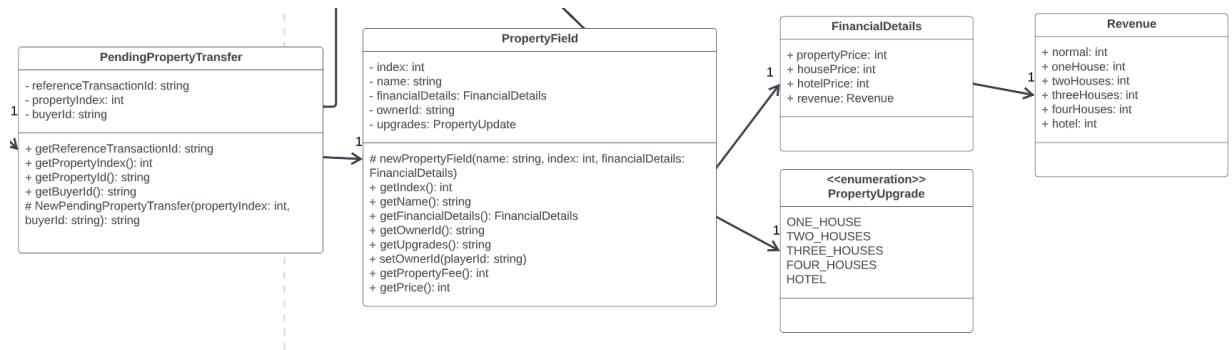


Abbildung 19 Entitäten und Value Objects für das Modellieren der Grundstücke

GameEvents: Nebst den Grundstücken befinden sich auch Ereignisfelder auf dem Spielbrett. Diese werden mit der Entität EventField modelliert. Zusätzlich werden die Ereigniskarten mithilfe der Entität Card abgebildet. Damit kann vom Game Aggregate sichergestellt werden, wann eine Karte gezogen wird, und wann die entsprechende Aktion ausgelöst wird (card_drew, game_event_with_payout_accepted, game_event_with_fee_accepted). Um die Verwechslungsgefahr mit den Events aus dem DDD zu verringern, wird dieser Kontext nicht nur Events, sondern GameEvents genannt. Die Beziehung zum Player in der Card Entität erlaubt es GameEvents mit komplexer Logik zu erstellen.

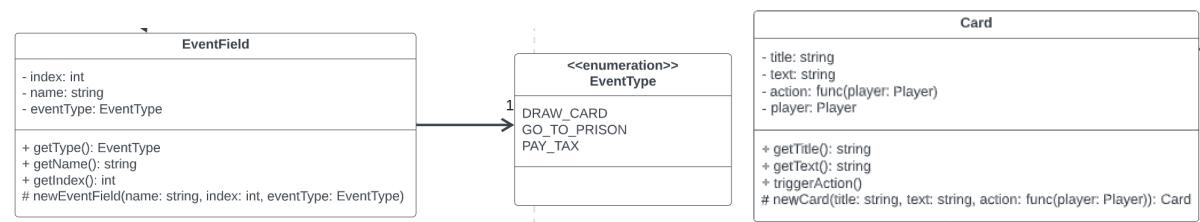


Abbildung 20 Entitäten und Value Objects rund um GameEvents

Finance: Ein weiteres zentrales Aggregate ist die Transaktion. Das Erzeugen einer Transaktion löst den Event `transaction_created` aus und soll die spielende Person zur Bezahlung auffordern. Die Bezahlung der Transaktion löst den Event `transaction_resolved` aus, was wiederum für die Eigentumsübertragung eines Grundstücks verwendet werden kann. Der Account wird vom Transaction-Aggregate referenziert, damit die entsprechenden Gutschriften und Belastungen erfolgen können.

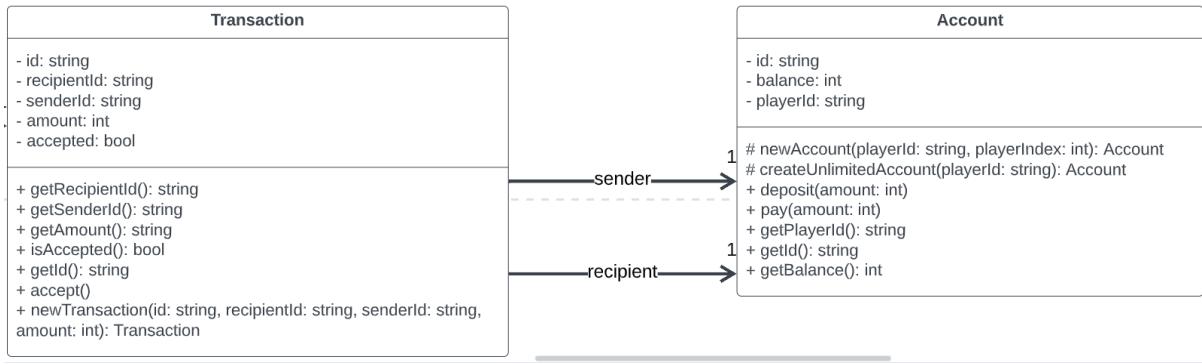


Abbildung 21 Entitäten Finance Subdomäne

Die vorgestellten Aggregates, Entities und Value Objects sind in Kombination dazu in der Lage die benötigte Spiellogik abzubilden. Wie die einzelnen Domänenevents mit der Domäne interagieren, kann der Abbildung 22 entnommen werden. Daraus ist die Interaktion zwischen Game und Finance ersichtlich, welche die bereits angesprochene asynchrone Natur der Geldtransaktionen widerspiegelt. Einerseits lösen manche Events das Erstellen einer Transaktion aus, während andererseits darauf gewartet wird, ob eine Transaktion akzeptiert wird, um anschliessend das Grundstück zu Übertragen. Darüber hinaus ist ersichtlich, dass die physischen Komponenten des Spielbretts zwei Schnittstellen der Applikation ansprechen müssen. Zum einen sollen die Bewegungen der Figuren gemeldet werden und zum anderen ist die Zahlkarte, welche am Terminal im Kontext einer Transaktion vorgelegt wird, von Interesse.

Durch die strikte Unterteilung in unterschiedliche Komponenten, welche untereinander anhand der Events und IDs kommunizieren, entsteht eine lose gekoppelte Applikation. Die einzelnen Komponenten haben klare und einfach erklärbare Zuständigkeiten. Dadurch können zukünftige Erweiterungen einfach vorgenommen werden. Hier zeigen sich die Vorteile von Domain Driven Design: Die modellierte Domäne kann anhand der einzelnen Kontexte einfach veranschaulicht werden und es ist sofort klar, welche Komponente für welche Funktionalität zuständig ist.

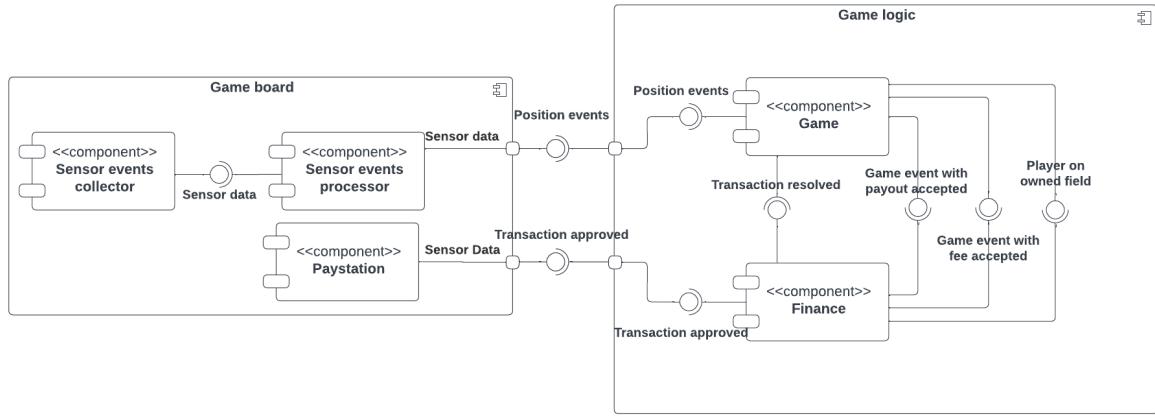


Abbildung 22 UML-Komponenten-Diagramm interne Events

Der Zustand der Applikation wird grösstenteils anhand der Position-Events vorangetrieben. Jedoch erfordern gewisse Aktionen, wie zum Beispiel das Starten des Spiels oder der Kauf eines Grundstücks, eine direkte Interaktion mit der Applikation. Damit diese Funktionalität abgebildet werden kann, benötigt das System eine weitere Schnittstelle. Dazu wird das in Abbildung 22 gezeigte Komponenten-Diagramm abgeändert und in Abbildung 23 im Kontext der Benutzenden-Interaktion dargestellt. Die bereits in Abbildung 22 dargestellten Events werden zur übersichtlicheren Darstellung nicht erneut dargestellt.

Aus der Abbildung geht hervor, dass das Graphical User Interface (GUI) den spielenden Personen die Möglichkeit gibt, das Spiel zu starten und zu beenden. Zusätzlich dient das GUI dazu, wichtige Kontextinformationen wie die Ereigniskarten, Grundstücksinformationen oder Transaktionsdetails darzustellen. Abbildung 23 zeigt auch, dass das GUI sowohl eingehende als auch ausgehende Schnittstellen hat. Es kann also einerseits Daten des Spiels aktiv verändern und andererseits auf eingehende Events reagieren. Damit wird abgebildet, dass eine Person die Spielfigur auf ein Feld zieht und das entsprechende Event (bspw. Player on unowned property) an das GUI übermittelt wird. Anschliessend ist das GUI in der Lage die entsprechenden Informationen des Grundstücks anzuzeigen und es der spielenden Person zum Kauf anzubieten. Die strikte Isolierung der Spiellogik in die Komponente «Game logic» gibt der Anwendung eine klare Struktur. Es entsteht eine nachvollziehbare Kette an Ereignissen, in welcher ein einfaches Positionierungsevent aus den Sensordaten in ein spielrelevantes Event umgewandelt wird. Die Abkapselung der Spiellogik führt auch dazu, dass das GUI tatsächlich nur noch eine reine Präsentationsschicht gemäss der vorgestellten Clean Architecture ist. Es beinhaltet keine Geschäftslogik, sondern bildet nur die Informationen aus den entsprechenden Events ab und manipuliert simple Zustände wie das Starten des Spiels oder das Akzeptieren eines Grundstückkaufs.

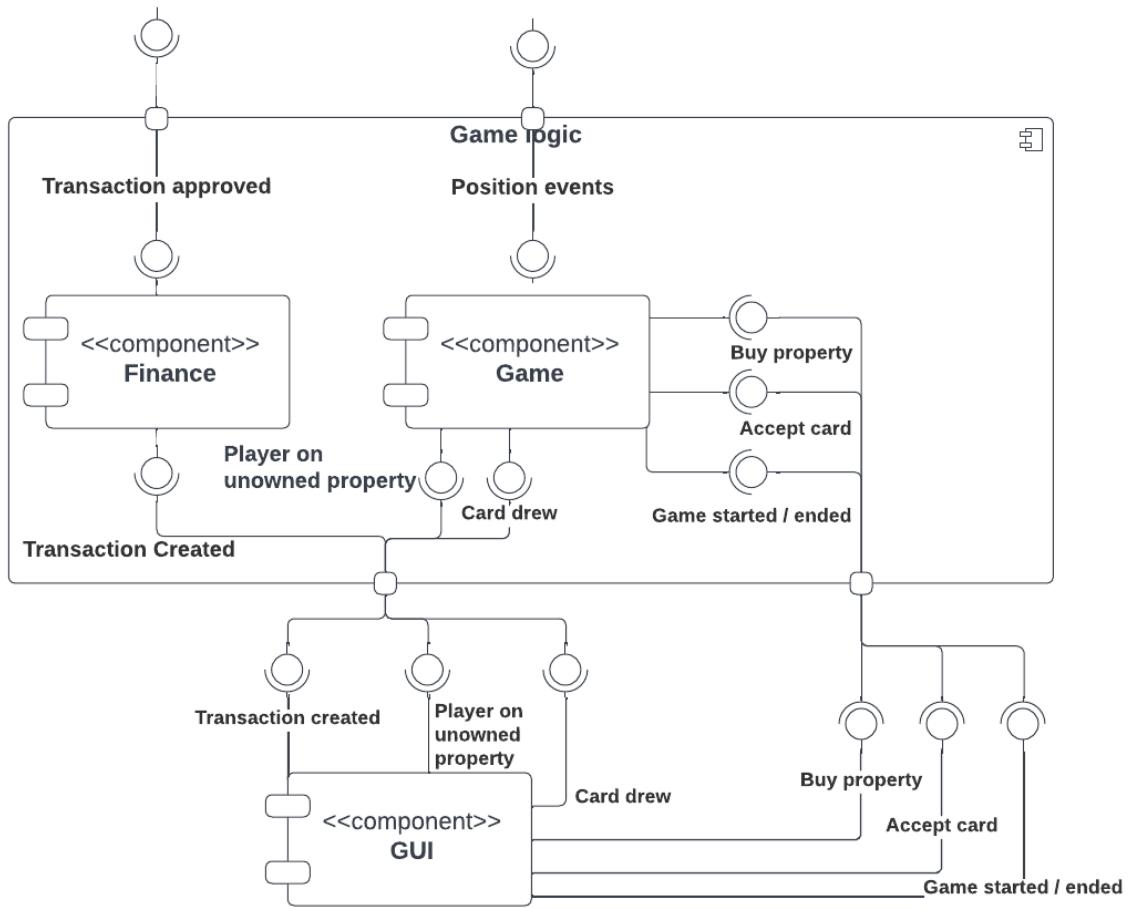


Abbildung 23 UML-Komponenten-Diagramm externe Events

3.3.3 System Architektur

Nachdem in den vorangegangen Abschnitten der Funktionsumfang sowie die Domäne vorgestellt wurde, wird nun eine System Architektur erarbeitet. Diese zeigt auf, wie die Komponenten funktionieren und wie die einzelnen Bestandteile miteinander kommunizieren. Die untenstehende Abbildung 24 zeigt den Aufbau des Systems. Jedes Arduino Uno prozessiert die eingehenden Signale der RFID-Sensoren (Sensor Event Collector). Wird ein RFID-Tag erkannt, wird eine Nachricht mit der ID des Tags und dem Index des Lesegerätes an das Raspberry Pi mittels serieller USB-Schnittstelle übermittelt. Die Verwendung der seriellen Schnittstelle zwischen den Arduinos und dem Raspberry ist schnell und zuverlässig. Dort wird diese Nachricht von einem Modul empfangen, welches dafür zuständig ist, alle eingehenden RFID-Events zu verarbeiten. Das Modul Sensor Event Processor generiert aus den eingehenden Nachrichten eine http-Anfrage, welche an das Backend übermittelt wird. Das Backend berechnet anhand der Positionen der Spielfiguren den neuen Zustand des Spiels und informiert das Frontend via WebSocket über allfällige Events, welche der spielenden Person angezeigt werden müssen. Das Frontend hingegen übermittelt die Eingaben der Benutzenden per http an das Backend und fragt falls nötig mit dem gleichen Protokoll weitere Informationen ab. Die Daten des

laufenden Spiels werden vom Backend in einer Datenbank abgespeichert, wodurch es möglich wird, Spiele über einen längeren Zeitraum hinweg zu spielen. Durch die Verwendung von http als einzige Schnittstelle für die Veränderung von Daten im Backend wird die Spiellogik abgekapselt und es ist nachvollziehbar, wie die aktuelle Spielsituation entstanden ist.

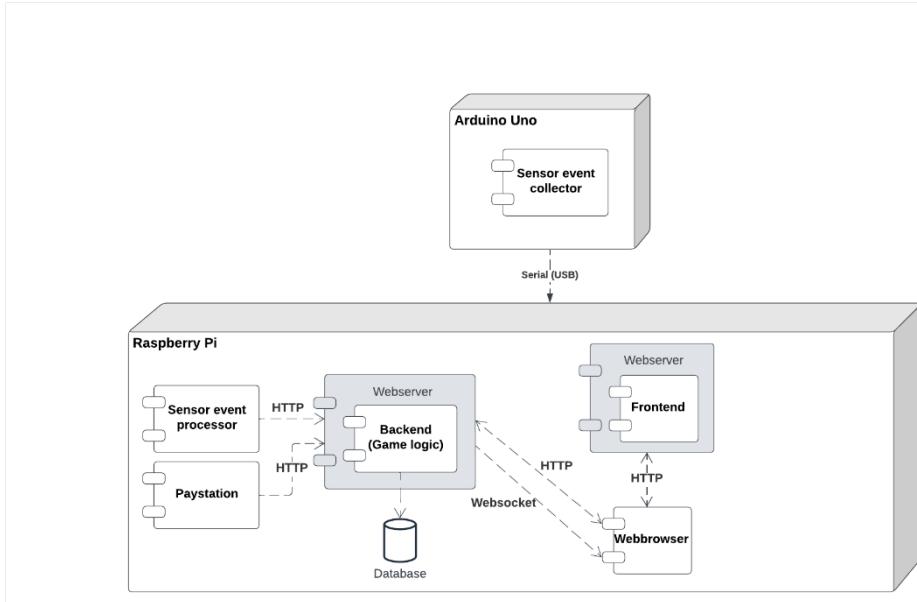


Abbildung 24 UML-Deployment-Diagramm

Die vorgestellte Architektur ermöglicht den lokalen Betrieb des Systems ohne Internetzugang, da die Verbindung zwischen den Komponenten mittels USB geschieht und sich alle weiteren Komponenten, welche miteinander kommunizieren müssen, auf demselben Gerät befinden. Zusätzlich ermöglicht diese Architektur, falls nötig, eine Auslagerung der Backend- und Frontend-Komponenten ins Internet. Dadurch sind weitere Anwendungsfälle, wie ein verteiltes Spiel auf mehreren örtlich getrennten Spielbrettern, denkbar. Die Auslagerung des Backends (und eventuell des Frontends) ins Internet und die damit einhergehende Verbindung des Spielbretts mit dem Internet zusammen mit der Vernetzung zu anderen solchen Spielbrettern würden dieses System durchaus als Smart-Device klassifizieren.

3.4 Prototyp

Im Rahmen dieser Arbeit wird das in Abschnitt 3 vorgestellte Design anhand eines Prototyps umgesetzt. Wie bereits erwähnt liegt der Fokus des Prototyps dabei auf der Integration der Hardware in die gesamte Applikation. Statt der gesamten 40 Spielfelder umfasst der Prototyp ein Brett mit 16 Feldern. Abbildung 25 zeigt das Spielbrett des Prototyps mit den entsprechenden Feldern, dem Display in der Mitte und einer Markierung für das Bezahlen mit der RFID-Karte. Das Spielbrett beinhaltet vier unterschiedliche Gruppen an Grundstücken und drei Ereignisfelder sowie ein Zusatzsteuer-Feld. Wie auch die vollständige Version, beinhaltet der Prototyp die Felder Start, Gefängnis, Frei Parken und Gehe ins Gefängnis. Jedes Feld ist mit einem LED-Indikator versehen, der signalisiert, sobald ein RFID-Tag auf dem Feld erkannt wird.

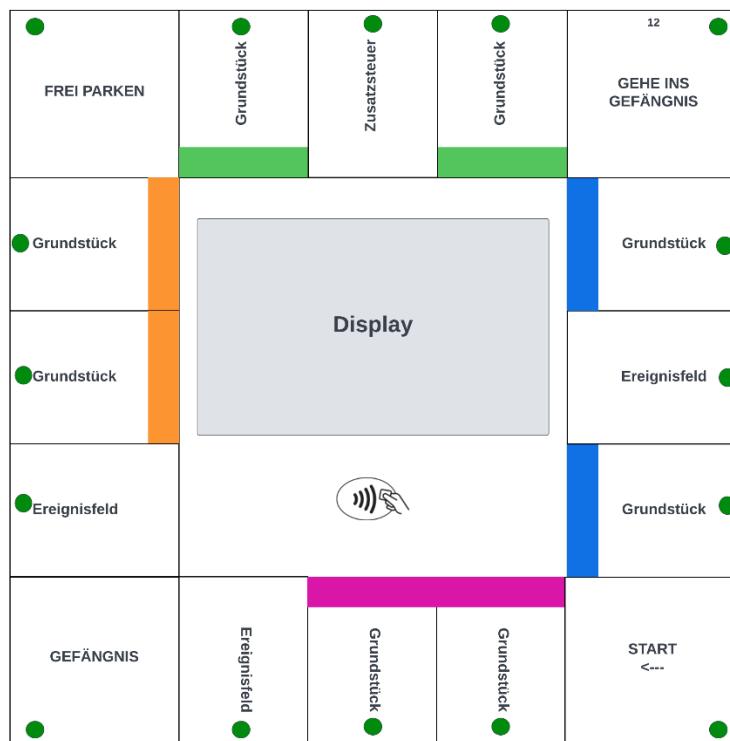


Abbildung 25 Monopoly-Spielbrett Prototyp

Das System unterstützt die folgenden Use Cases: Starten und Beenden des Spiels, Kaufen eines Grundstücks, Ziehen einer Ereigniskarte, Erhalten von Geld beim Überqueren des Startfeldes und bezahlen der Zusatzsteuer. Diese Prozesse werden durch die Anwendung via Bildschirm unterstützt. Zudem zeigt die Applikation, welche Person gerade am Zug ist. Alle Geldtransaktionen dieser Use Cases werden mithilfe der Bezahlkarte abgewickelt und die Position der Spielfigur wird vom Spielbrett verfolgt. Der Prototyp wird ohne Datenbank-Anbindung implementiert.

Der Prototyp wird aus einem vier Millimeter dicken Holz mit den Massen 36 cm x 36 cm gefertigt. Die Höhe beträgt 10 cm. Damit entsteht eine Box, welche im inneren genügenden Platz für die Mikrocontroller sowie die RFID-Sensoren bieten soll. Die Anordnung der Mikrocontroller,

Steckbretter und des USB-Hubs ist der Abbildung 26 zu entnehmen. Diese zeigt die Organisation der Komponenten im inneren des Prototyps schematisch auf. Dabei gilt es anzumerken, dass die Abbildung nicht massstabsgetreu ist. Es wird jeweils ein Arduino Uno pro Seite positioniert. Diese werden mit dem USB-Hub verbunden und verfügen über sämtliche SPI-Verbindungen für vier RFID-Sensoren sowie vier LEDs gemäss Abbildung 6 und Abbildung 11.. Die Steckbretter sind an der Wand der Box befestigt. Das Raspberry wird via USB-Hub mit allen Arduino Unos verbunden. Der USB-Hub sowie das Raspberry werden direkt durch ein separates Netzteil mit Strom verbunden. Zusätzlich zu den in Abbildung 26 dargestellten Verbindungen wird das Display mit dem USB-Hub sowie mittels HDMI mit dem Raspberry verbunden.

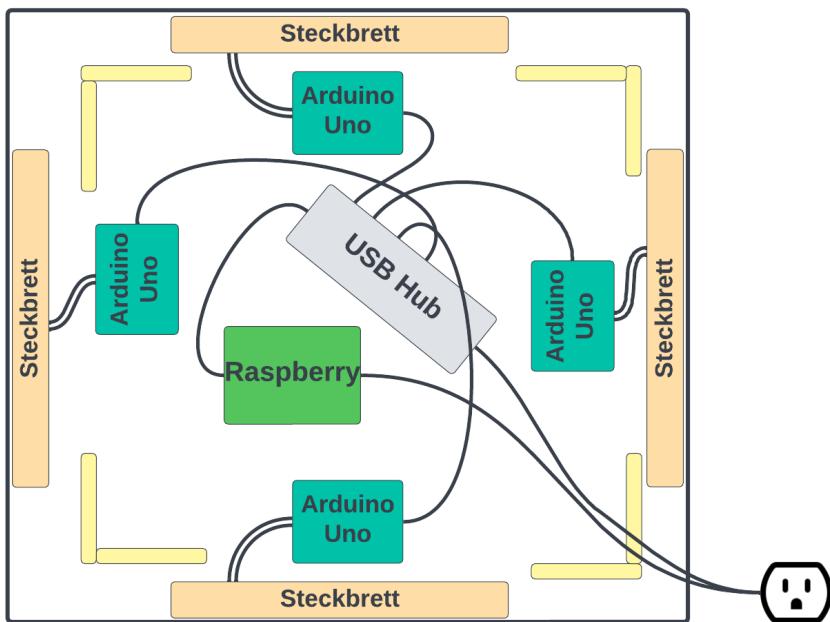


Abbildung 26 Innenansicht Prototyp (nicht massstabsgetreu)

Damit die RFID-Sensoren im Prototyp verbaut werden können, benötigt es eine Konstruktion innerhalb der Box, welche eine Art zweite Ebene erstellt, auf denen die Sensoren positioniert werden können. Dabei ist es von zentraler Bedeutung, dass die Sensoren möglichst nahe an der Unterseite des Deckels positioniert werden können. Dadurch soll sichergestellt werden, dass sich die Sensoren nicht gegenseitig stören und nur Tags lesen, welche auf dem von ihnen überwachten Feld positioniert werden. Hinzu kommt, dass sich die Reichweite der Sensoren durch das 4 mm dicke Holz sicherlich verschlechtern wird und eine nahe Positionierung dem entgegenwirkt. An den in Abbildung 26 dargestellten gelben Holzstützen, welche eine Höhe von rund 8 cm haben, wird eine dünne Holzplatte von rund 6 cm breite befestigt, worauf dann die RFID-Sensoren befestigt werden. Abbildung 27 zeigt die dadurch entstehende «zweite Etage» im inneren der Box. Jeder RFID-Sensor repräsentiert ein Feld und verfügt dabei, wie in Abbildung 6 gezeigt über sieben ausgehende Kabel, welche mit jeweilige Steckbrett verbunden wird. Durch die zahlreichen SPI-Verbindungen für die RFID-Sensoren und LED-

Lampen werden pro Arduino rund 50 Kabel verwendet. Der Platz im inneren der Box ist begrenzt und die Mikrokontroller zusammen mit dem USB-Hub und den entsprechenden USB-Kabel nehmen bereits viel Platz ein. Umso wichtiger ist es, dass sämtliche Kabel gekennzeichnet sind und eine gewisse Ordnung sichergestellt wird. Weitere Komponenten wie der Touchbildschirm und der für die Zahlstation zuständige MFRC522-Sensor, der direkt an das Raspberry angeschlossen werden, sind nicht abgebildet. Diese Teile werden direkt im Deckel integriert, wobei der RFID-Sensor am Bildschirm angeklebt werden soll, damit der Bildschirm durch die Aussparung sichtbar ist und der RFID-Sensor an der Unterseite des Deckels positioniert ist.

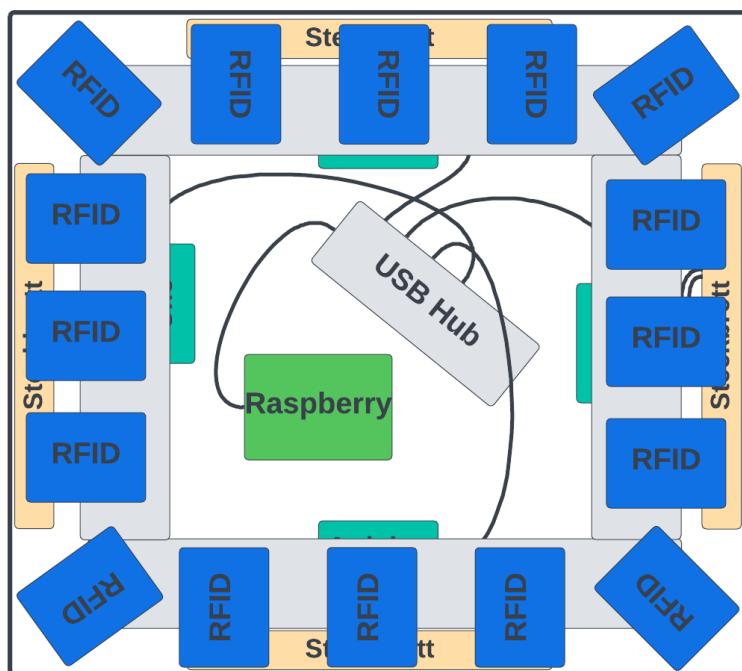


Abbildung 27 Innenansicht Prototyp (Konstruktion RFID, nicht massstabsgetreu)

4 Umsetzung

Nachdem im vorherigen Kapitel die Zielarchitektur ausführlich erläutert wurde und die Pläne für den zu erstellenden Prototyp vorgestellt wurden, zeigt dieses Kapitel nun die zentralen Elemente der Umsetzung des Prototyps. Dabei wird im Speziellen darauf eingegangen, wie die erarbeitete Architektur mit DDD und Clean Architecture implementiert wird und wie die Integration der Hardware in das gesamte System erreicht wird. Zudem wird auf die Evaluation der im Abschnitt 3.2.1 RFID-Module gezeigten RFID-Tags eingegangen, welche im Rahmen der Umsetzung stattfindet.

4.1 Hardware (Game Board)

Im folgenden Abschnitt wird die Integration der MFRC522 Module mit dem Spielbrett erläutert. Zusätzlich findet die Evaluation der RFID-Tags statt.

4.1.1 Spielbrett

Der im Rahmen dieser Arbeit erstellte Prototyp ist in der Abbildung 28 und Abbildung 29 ersichtlich. Die bereits erwähnte hohe Anzahl an Kabeln führt dazu, dass der Platz im inneren des Prototyps trotz der grosszügigen Dimensionen sehr begrenzt ist. Hinzu kommt, dass es bei der Verwendung von Steckbrettern dazu kommen kann, dass ein Kabel den Kontakt verliert, was durch die Verbindung des Deckels mit der Box, wie in Abbildung 29 gezeigt, zu einem hohen Reparaturaufwand führt.

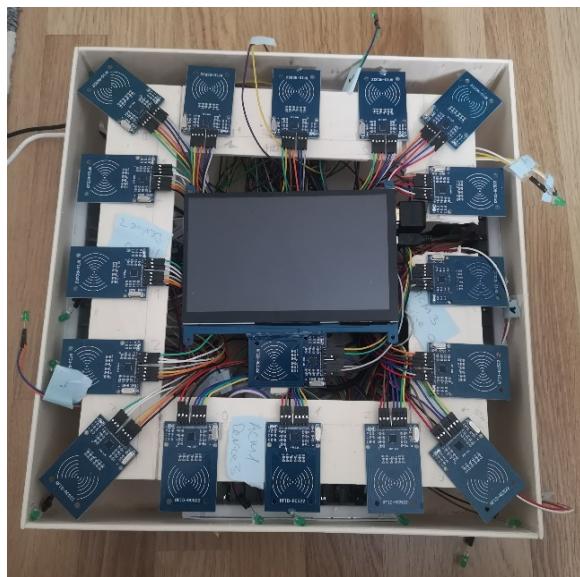


Abbildung 28 Prototyp Innenansicht

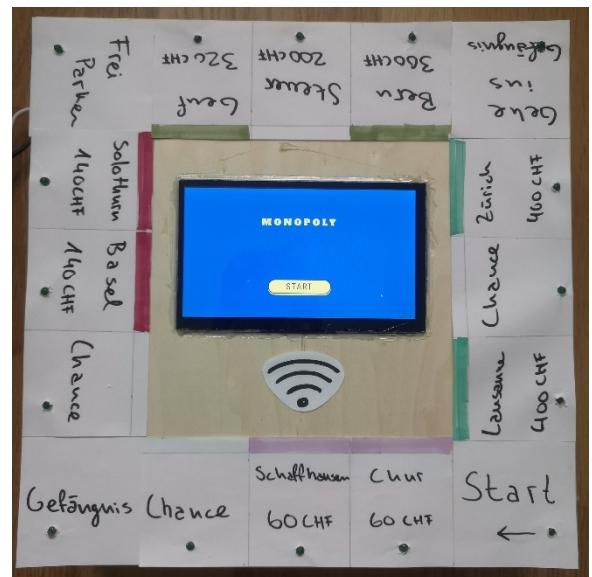


Abbildung 29 Prototyp Aussenansicht

4.1.2 Sensor Events Collector

Jedes Arduino Uno überwacht vier MFRC522-Sensoren und steuert vier LEDs. Für die Kommunikation mit den MFRC522 RFID-Sensoren wird eine entsprechende Bibliothek von Balboa (2012/2022) verwendet, welche öffentlich auf GitHub zugänglich ist. Die Komponente Sensor Events Collector (siehe Abbildung 22) wird auf jedem Arduino ausgeführt und ist für die Überwachung der angeschlossenen MFRC522 Module zuständig. Nach einer Initialisierung der Sensoren, kann mittels einem Funktionsaufruf geprüft werden, ob ein RFID-Tag in Reichweite ist. Abbildung 30 zeigt den relevanten Code-Ausschnitt, welcher die Kommunikation zwischen vier MFRC522 Modulen und einem Arduino Uno implementiert.

```
49 void loop() {
50
51     for (uint8_t reader = 0; reader < NR_OF_READERS; reader++) {
52
53         delay(100);
54         if (mfrc522[reader].PICC_IsNewCardPresent() && mfrc522[reader].PICC_ReadCardSerial()) {
55
56             delay(1500);
57             mfrc522[reader].PICC_ReadCardSerial();
58             mfrc522[reader].PICC_IsNewCardPresent();
59
60             if (mfrc522[reader].PICC_ReadCardSerial()) {
61
62                 digitalWrite(lcdPins[reader], HIGH);
63                 printHex(mfrc522[reader].uid.uidByte, mfrc522[reader].uid.size, reader);
64
65                 mfrc522[reader].PICC_HaltA();
66                 mfrc522[reader].PCD_StopCrypto1();
67
68                 flashLED(reader);
69             }
70         }
71     }
72 }
```

Abbildung 30 Code-Ausschnitt Kommunikation mit MFRC522

Dabei wird in einer For-Schlaufe pro angeschlossenen Sensor geprüft, ob ein neuer RFID-Tag detektiert wurde (Zeile 54). Damit eine Spielfigur nicht sofort vom RFID-Sensor erkannt wird, sorgen die Zeilen 56 bis 58 dafür, dass nach 1.5 Sekunden der Sensor erneut ausgelesen wird. Ohne diese Verzögerung würde das in Brettspielen beliebte gestaffelte Vorrücken der Spielfigur inkorrekte Positionierungs-Events auslösen. Ein kurzes Absetzen der Figur wird so nicht als definitive Positionierung erkannt. Mit der eigens implementierten Funktion printHex auf Zeile 63 wird eine Nachricht via seriellen Ausgang gesendet, welche die ID des RFID-Tags, den Index des Sensors und die ID des Arduinos als JSON-Objekt übermittelt. Dies entspricht dem in Abbildung 15 gezeigten TagReadEvent. Anschliessend wird mit den Zeilen 59 und 60 die Kommunikation mit dem RFID-Tag beendet. Die Funktion flashLED ist ebenfalls eine eigene Implementierung und sorgt für ein Blinkeffekt des entsprechenden LED.

4.1.3 Sensor Events Processor

Im folgenden Abschnitt wird die Implementierung der in Abbildung 22 dargestellten Komponente Sensor Events Processor beschrieben. Die Aufgabe der Komponente ist es, die rein technischen Informationen in brauchbare spielbezogene Nachrichten umzuwandeln und diese via http an die GameLogic Komponente zu übermitteln. Diese Komponente verarbeitet die via serielle Schnittstelle eingehenden tag_read Events der Arduino Unos. Der Fokus liegt auf der in Abbildung 22 dargestellten Umwandlung der eingehenden Nachrichten in für die Spiellogik brauchbare Positionierungs-Events.

Mithilfe der Zeilen 25 bis 29 wird die Verbindung zu den seriellen USB-Schnittstellen der Arduinos initialisiert und konfiguriert. Für jedes Gerät werden nun in den Zeilen 32 bis 47 die eingehenden Nachrichten überprüft. Die Informationen im eingehenden JSON-Objekt werden transformiert. Anhand der deviceId und der fieldId kann der fieldIndex im Kontext des gesamten Spielbretts berechnet werden. Die GameLogic kennt keine TagId. Die Komponente arbeitet mit der playerId, weshalb ein weiteres Mapping zwischen den beiden IDs stattfindet. Anschliessend wird in der Zeile 45 eine http-Anfrage an die GameLogic Komponente ausgeführt. Hierbei gilt es anzumerken, dass jeder Sensor-Event eine http-Anfrage auslöst, auch wenn beispielsweise eine Spielfigur auf einem Feld zwei Mal erkannt wird. Die Logik, ob sich dadurch die aktuelle Spielsituation verändert, liegt im Zuständigkeitsbereich der GameLogic Komponente.

```
25     for i in range(deviceCount):
26         device = serial.Serial("/dev/ttyACM"+str(i),9600,timeout=0.5, )
27         device.baudrate=9600
28         textWrapper = io.TextIOWrapper(io.BufferedRWPair(device, device))
29         devices.append(textWrapper)
30
31
32     while True: # Run forever
33
34         for i in range(deviceCount):
35             line = devices[i].readline()
36             if line.startswith("{"):
37                 try:
38                     message = json.loads(line)
39                     print("read message: ")
40                     print(message)
41
42                     field_index = map_message_to_field_index(message["deviceId"], message["fieldId"])
43                     player_id = map_tag_id_to_player_id(message["playerId"])
44
45                     requests.patch("http://localhost:3000/players/" + player_id, data={"position": field_index})
46                 except Exception:
47                     print("Request failed")
```

Abbildung 31 Python-Code zur Umwandlung der eingehenden seriellen Nachrichten in http-Anfragen

4.1.4 Paystation

Diese Komponente ist zuständig für die card_read Events, welche Aktivitäten an der Bezahlstation repräsentieren. Diese stammen jedoch nicht von einem der vier Arduino Unos, sondern direkt von der in Abbildung 10 gezeigten SPI-Verbindung zwischen dem MFRC522 Modul und dem Raspberry. Die Kommunikation wird durch die Python-Bibliothek von pimylifeup (*mfrc522*, 2017/2022), welche auf Github verfügbar ist, erreicht. Die Implementierung der Events von der Bezahlstation kann dem Anhang 5 entnommen werden.

4.1.5 Evaluation RFID-Tag Erkennung

Im Rahmen der Umsetzung wird geprüft, welches der im Abschnitt 3.2.3 RFID-Tags vorgestellten Modelle am besten im Kontext des Brettspiels erkannt wird. Die Herausforderung liegt darin, dass sich zwischen dem RFID-Tag und dem MFRC522-Sensor eine rund 4 mm Dicke Holzplatte befindet. Der Test erfolgt folgendermassen: Das RFID-Tag wird von Spielfeld zu Spielfeld gezogen. Dabei wird er für jeweils vier Sekunden auf dem Feld platziert, bevor er weitergezogen wird. Nach 40 Spielzügen ist der Test beendet. Die untenstehende Tabelle 5 zeigt die Ergebnisse der Untersuchung.

RFID-Tag	Anzahl erkannter Züge	In Prozent
Mifare Classic	40 / 40	100 %
NTag213	20 / 40	50 %
NTag215	12 / 40	30 %

Tabelle 5 Auswertung der Erkennungsrate der RFID-Tags

Aus der Auswertung geht hervor, dass der Mifare Classic Tag jeden einzelnen Zug erkannt hat und somit am besten abschneidet. Dies ist wohl auch auf die Grösse des Tags zurückzuführen. Auffällig ist der Unterschied zwischen dem NTag213 und NTag215: Obwohl der NTag215 robuster wirkt und wesentlich dicker ist als der NTag213 schneidet er in diesem Test schlechter ab. Während dem Test wurde beobachtet, wie die Erkennungsrate des NTag213 und NTag215 zu Beginn des Tests relativ hoch war, sich jedoch bei zunehmender Anzahl Spielzügen stetig verschlechterte, bis schliesslich gar keine Erkennung mehr registriert werden konnte.

Im Hinblick darauf, dass die verwendete Technologie den Spielfluss vereinfachen und nicht behindern soll, wird im weiteren Verlauf dieser Arbeit der Mifare Classic Tag verwendet. Die massiv grösseren Dimensionen des Tags führen zwar zu grösseren Spielfiguren, diese werden dafür aber immer von den Sensoren erkannt. Die Verwendung von einem Tag mit einer schlechten Erkennungsrate wirkt sich negativ auf das Spielerlebnis aus und wird in den meisten Fällen wohl Frustration auslösen.

4.2 Backend (Game Logic)

Diese Komponente bildet die gesamte Spiellogik ab und verfügt dafür über eine http-Schnittstelle inklusive Websockets. Für die Implementierung der Komponente wird die Programmiersprache Golang verwendet, deren Vorteile der einfache Betrieb auf dem Raspberry und der geringe Ressourcenverbrauch sind.

4.2.1 Projekt Struktur

Mit der Anwendung von Clean Architecture und DDD ergibt sich eine übersichtliche Projektstruktur sowie eine klare Kette von Abhängigkeiten im Projekt. Die (Sub-) Domänen werden in einzelne Ordner gegliedert, welche wiederum drei Unterordner besitzen, die sich an der Clean Architecture orientieren. Nachfolgend wird die Ordnerstruktur in Abbildung 32 beispielhaft dargestellt. Dabei wird die gemäss Clean Architecture zentralen Abhängigkeitsregel befolgt. Die Farben entsprechen den im Clean Architecture Diagramm (Abbildung 1) verwendeten Farben.

Der Ordner `api` repräsentiert die Controller Schicht und beinhaltet die REST-Schnittstelle für das UI. Der Ordner `adapter` repräsentiert die Schicht Use Cases und beinhaltet die Services, Repositories und Eventhandler. Die innerste Schicht Entitites wird durch den Ordner `Domain` abgebildet. Dort befinden sich die Aggregates, Entities, Domainevents und Value Objects der Domäne. Die Abhängigkeiten werden von der Clean Architecture vorgegeben. So hat der Controller keine direkte Verbindung zur Domäne, sondern implementiert die Endpunkte mithilfe der vom Adapter zur Verfügung gestellten Funktionen. Die Services im Adapter (Use Cases Schicht) greifen dabei via Repository auf die Domänenobjekte zu und manipulieren deren Status. Da im Rahmen des Prototyps keine Datenbankanbindung implementiert wurde, kommunizieren die Repositories nicht mit einer Datenbank, sondern verwalten die Objekte in-memory.

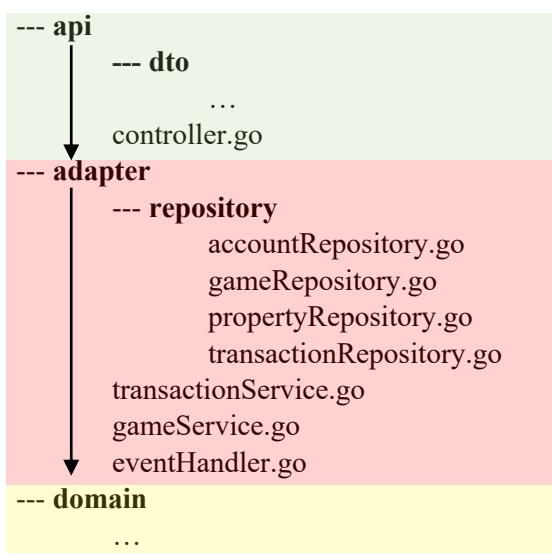


Abbildung 32 Ordnerstruktur nach Clean Architecture

4.2.2 http-Schnittstelle

Die http-Schnittstelle ist der Ein- und Ausgangspunkt der Applikation. Diese wird mithilfe der Bibliothek «Fiber» implementiert und ist in der untenstehenden Abbildung 33 zu sehen. Die Implementation orientiert sich am REST-Architektur Stil und bildet die einzelnen relevanten Ressourcen der Domäne (Game, Players, Properties, Transactions, Cards) ab. Dabei ist anzumerken, dass wie im Abschnitt 2.3 beschrieben die zustandsverändernden Methoden mithilfe einer PATCH, statt einer PUT-Anfrage erfolgen.

RFID-Monopoly 1.0.0 OAS3

This is the backend to provide the game logic for monopoly

games

POST	/games Create a new game	▼
GET	/games/current Get the current game	▼
PATCH	/games/current Patch the information of the current game (e.g. to end it)	▼
PATCH	/games/current/players/{playerId} Patch a player (e.g. the position)	▼
PATCH	/games/current/properties/{propertyIndex} Patch a property (e.g. its owner)	▼
PATCH	/games/current/transactions/current Patch the current transaction (e.g. accept it)	▼
PATCH	/games/current/cards/current Patch the current card (e.g. accept it)	▼

Abbildung 33 http-Endpunkte

4.2.3 Eventing

Aus dem System Design geht hervor, dass die Domänen-Events von zentraler Bedeutung für die Spiellogik sind. Um diese Events innerhalb der Applikation zu versenden und entsprechend zu beobachten, wird ein Messagebus eingesetzt. Dabei handelt es sich um einen internen Messagebus. In diesem Abschnitt wird speziell auf die Implementierung eingegangen, um aufzuzeigen wie eines der Kernelemente von DDD (die Domänenevents) in einer Applikation umgesetzt werden kann.

Für die Implementierung sind die beiden in Abbildung 34 gezeigten Methoden im separaten Package eventing (siehe Anhang 6) von grosser Bedeutung. Mithilfe der FireEvent-Methode kann von überall aus in der Applikation ein Event für einen entsprechenden Kanalausgelöst werden. Diese Methode kreiert in der Zeile 50 den entsprechenden Channel, falls der Kanal noch nicht besteht und versendet den Event an alle registrierten Eventlistener in den Zeilen 58 bis 62. Für die Implementation wird auf

die Bibliothek Monoton zurückgegriffen. Damit andere Komponenten der Applikation in der Lage sind auf diese Events zu reagieren, wird mit der unten abgebildeten Methode RegisterEventHandler die Möglichkeit geboten, für ein entsprechendes Topic zu registrieren. Auch diese Methode prüft die Existenz des Channels mithilfe der Zeile 31 und registriert den Handler in der Zeile 35.

```

44 func FireEvent(channelName ChannelName, event any) {
45
46     fmt.Println("Firing event:")
47     fmt.Println(event)
48
49     name := string(channelName)
50     createChannelIfNotExists(name)
51
52     txID := config.Monoton.Next()
53     ctx := context.Background()
54     ctx = context.WithValue(ctx, bus.CtxKeyTxID, txID)
55
56     b := config.Bus
57
58     err := b.Emit(
59         ctx,
60         name,
61         event,
62     )
63     if err != nil {
64         fmt.Println("ERROR >>>", err)
65     }
66 }

```

```

11 type ChannelName string
12
13 const (
14     GAME_STARTED           ChannelName = "gameStarted"
15     GAME_ENDED             ChannelName = "gameEnded"
16     PLAYER_ON_UNOWNED_FIELD ChannelName = "playerOnUnownedField"
17     PLAYER_ON_OWNED_FIELD ChannelName = "playerOnOwnedField"
18     TRANSACTION_CREATED    ChannelName = "transactionCreated"
19     TRANSACTION_RESOLVED   ChannelName = "transactionResolved"
20     LAP_FINISHED           ChannelName = "lapFinished"
21     GAME_EVENT_WITH_PAYOUT_ACCEPTED ChannelName = "gameEventWithPayoutAccepted"
22     GAME_EVENT_WITH_FEE_ACCEPTED   ChannelName = "gameEventWithFeeAccepted"
23     CARD_DREW               ChannelName = "cardDrew"
24 )

```

```

29 func RegisterEventHandler(handler bus.Handler) string {
30     fmt.Println("Registering handler for channel: " + handler.Matcher)
31     createChannelIfNotExists(handler.Matcher)
32     id := uid.NewString()
33     fmt.Printf("Handler received id %s\n", id)
34
35     config.Bus.RegisterHandler(id, handler)
36
37     return id
}

```

Abbildung 34 Implementierung DDD-Eventing

Um sicherzustellen, dass in der gesamten Applikation konsistente Channel-Namen verwendet werden, wird ein Enum erstellt, welches alle vorhandenen Channels speichert. Damit sind die Grundlagen für das Austauschen von Domänenevents gegeben und die unterschiedlichen Subdomänen können auf diese Funktionalität zurückgreifen. Abbildung 35 zeigt, wie nun das Transaction-Aggregate für relevante Geschäftsfälle ein Domänenevent auslösen kann. Sowohl bei der Erstellung (Zeile 44), als auch beim Akzeptieren (Zeile 38) wird ein Domänenevent ausgelöst.

```

35 func (t Transaction) Accept() {
36
37     t.accepted = true
38     eventing.FireEvent(eventing.TRANSACTION_RESOLVED, newTransactionResolvedEvent(t.id))
39 }
40
41 func NewTransaction(id string, recipientId string, senderId string, amount int) *Transaction {
42
43     transaction := &Transaction{id: id, recipientId: recipientId, senderId: senderId, amount: amount}
44     eventing.FireEvent(eventing.TRANSACTION_CREATED, newTransactionCreatedEvent(transaction.recipientId, transaction.senderId, transaction.amount))
45
46     return transaction
47 }
48

```

Abbildung 35 Implementierung Domänen-Event Transaction Aggregate

Um einen entsprechenden Eventhandler zu implementieren, wird mithilfe des in Abbildung 36 auf Zeile 29 gezeigten Matchers ein Eventhandler für den entsprechenden Channel erstellt. Zeilen 25 bis 27 implementieren die Logik, welche ausgeführt wird, sobald ein Event im Channel `transcation_resolved` übermittelt wird.

```

21 func startTransactionResolvedHandler() {
22
23     eventing.RegisterEventHandler(bus.Handler{
24         Handle: func(ctx context.Context, e bus.Event) {
25             transactionResolved := e.Data.(domain.TransactionResolvedEvent)
26             fmt.Println("Transaction #{transactionResolved.TransactionId} is resolved, check for pending actions to trigger\n")
27             GetCurrentGame().ResolvePendingPropertyTransfer(transactionResolved.TransactionId)
28         },
29         Matcher: string(eventing.TRANSACTION_RESOLVED),
30     })
31 }
```

Abbildung 36 Implementierung EventHandler

Wie im Kontextdiagramm (Abbildung 23) gezeigt, müssen jedoch gewisse Events auch aktiv nach aussen via Websocket propagiert werden. Hier zeigen sich erneut die Vorteile von DDD, indem die bereits bestehenden Events wiederverwendet werden können. Die untenstehende Abbildung 37 zeigt wie zuerst die für die Aussenwelt interessanten Domänenevents definiert werden (Zeile 26). Für jeden Channel wird anschliessend ein entsprechender Eventhandler erstellt, der das Domänenevent den bestehenden Websocket-Verbindungen propagiert. Die damit erstellte Funktionalität kann für weitere Domänen-Events erweitert werden und verbindet das Backend mit dem Frontend.

```

26 var EXTERNAL_CHANNELS = []ChannelName{PLAYER_ON_UNOWNED_FIELD, TRANSACTION_CREATED, TRANSACTION_RESOLVED, CARD_DREW, LAP_FINISHED}
27
28 func registerWebsocket(app *fiber.App) fiber.Router {
29     return app.Get( path: "/ws", websocket.New(func(c *websocket.Conn) {
30
31         for i := range EXTERNAL_CHANNELS {
32
33             RegisterEventHandler(bus.Handler{
34                 Handle: func(ctx context.Context, e bus.Event) {
35                     err := c.WriteJSON(e.Data)
36                     if err != nil {
37                         fmt.Println(err)
38                         return
39                     }
40                 },
41                 Matcher: string(EXTERNAL_CHANNELS[i]),
42             })
43         }
44     }))
45 }
```

Abbildung 37 Implementation Websocket zur Verteilung von Domänen-Events

4.2.4 Beispiel-Implementation

Das Zusammenspiel der Domänenevents und die Einfachheit von DDD kann am besten anhand eines konkreten Ablaufs eines Spielszenarios aufgezeigt werden. Nachfolgend wird das Vorrücken auf ein Grundstücksfeld anhand des im Anhang 1 gezeigten UML-Aktivitätsdiagramm beschrieben. Sobald die spielende Person die Spielfigur auf dem Spielbrett positioniert und diese von einem RFID-Sensor erkannt wurde, wird die Position via REST-Schnittstelle verändert. Je nachdem, ob das Grundstück bereits von einer anderen Person gekauft wurde oder nicht, wird ein player on owned property oder player on unowned propertyEvent generiert und versendet.

Die Finance Subdomäne implementiert einen entsprechenden Eventhandler für das player on owned property Event und kreiert eine Transaktion zwischen der auf dem Feld gelandeten Person und der besitzenden Person des Grundstücks über den zu bezahlenden Betrag. Das entsprechende Transaction Aggregate löst im Konstruktor ein Transaction created event aus. Dies wird in der Applikation propagiert und via WebSocket ans Frontend übermittelt. Dort werden die entsprechenden Informationen der Transaktion angezeigt (siehe Anhang 7) und es wird darauf gewartet, dass die betroffene Person die Zahlkarte an der Zahlstation präsentiert. Dadurch wird eine http-Anfrage ausgelöst. Sobald der Finance Kontext die Transaktion akzeptiert und die entsprechenden Kontenstände angepasst hat, wird ein transaction resolved event ausgelöst.

Im Falle eines player on unowned property event, reagiert das Frontend auf dieses Ereignis und stellt die Eckdaten des Grundstücks dar (siehe Anhang 7). Entscheidet sich die spielende Person für den Kauf des Grundstücks, wird eine entsprechende http-Anfrage ausgelöst. Der angestrebte Zustand ist das Property mit der ownerId der spielenden Person, welche am Zug ist. Dies führt zu einem pendenten Grundstücktransfer, wobei anschliessend eine Transaktion erstellt wird. Dadurch wird wiederum die Zahlungsaufforderung wie im obigen Fall auslöst. Der einzige Unterschied ist, dass für den Abschluss des Grundstücktransfers die Core Domain auf das transaction resolved event reagiert und beim Erhalt des Events den neuen Eigentümer persistiert. Das Frontend navigiert beim Erhalt eines transaction resolved events via WebSocket wieder zurück auf die Spielübersicht.

Wie Anhang 1 zeigt, wird die Spiellogik durch das konsequente Propagieren von Domänenevents in kleine und einfache Programmteile aufgeteilt. Statt einer starren Aneinanderreihung der Spiellogik in einer einzelnen Komponente, welche für mehrere Aufgaben gleichzeitig zuständig ist, wird der Programmfluss durch die Verwendung von Events lose gekoppelt und stark vereinfacht. Dies führt zu klaren Zuständigkeiten innerhalb des Programmcodes. Damit zeigt sich ein weiterer Vorteil von DDD: Es ist einfacher, sich an das Single Responsibility Pattern zu halten.

4.3 Frontend

Im folgenden Abschnitt wird darauf eingegangen, wie die Domänen-Events im Frontend verarbeitet werden und wie somit ein fliessender Spielfluss entsteht. Zudem werden einige der implementierten Ansichten anhand eines Spielzuges gezeigt. Das Frontend wird mithilfe von Svelte und SvelteKit umgesetzt. Dabei handelt es sich um ein aufsteigendes Frontend-Framework, welches sich durch Performance und eine gute Developer-Experience auszeichnet.

4.3.1 Websocket Implementation

Wie Anhang 4 zeigt, erbt jeder Event vom BaseEvent. Dadurch stellt jeder Event die Information zur Verfügung, um welchen Event Typ es sich handelt. Die untenstehende Abbildung 38 zeigt, wie die eingehenden Events des Websockets verarbeitet werden. Der entsprechende Event Typ wird in den Zeilen 11 und 12 ausgelesen. Damit kann im Anschluss je nach Event eine Navigation zur entsprechenden Seite vorgenommen werden. Um die Komplexität bezüglich State-Management im Frontend möglichst tief zu halten, werden die Daten, welche angezeigt werden sollen, mittels der URL in Form von Query-Parametern übermittelt. Dieser Mechanismus kann ohne grossen Aufwand mit weiteren Ansichten und Events erweitert werden.

```
7  onMount(async () => {
8    const socket = new WebSocket('ws://localhost:3000/ws');
9
10   socket.onmessage = (backendEvent: MessageEvent<any>) => {
11     const eventData = JSON.parse(backendEvent.data);
12     switch(eventData.Type.split('.')[1]){
13
14       case "TransactionCreatedEvent":
15         let transaction = eventData
16         goto(` /transactions/${transaction.Id}?recipientId=${transaction.RecipientId}&
17           senderId=${transaction.SenderId}&amount=${transaction.Amount}` )
18         break
19       case "PlayerOnUnownedFieldEvent":
20         goto(` /properties/${eventData.PropertyIndex}?name=${eventData.PropertyName}&
21           &propertyPrice=${eventData.PropertyPrice}&housePrice=${eventData.HousePrice}&
22           &hotelPrice=${eventData.HotelPrice}&revenueNormal=${eventData.RevenueNormal}&
23           &revenueOneHouse=${eventData.RevenueOneHouse}&revenueTwoHouses=${eventData.RevenueTwoHouses}&
24           &revenueThreeHouses=${eventData.RevenueThreeHouses}&revenueFourHouses=${eventData.RevenueFourHouses}&
25           &revenueHotel=${eventData.RevenueHotel}&buyerId=${eventData.PlayerId}` )
26         break
27       case "CardDrewEvent":
28         goto(` /card-event?title=${eventData.Title}&text=${eventData.Text}` )
29         break
30       case "TransactionResolvedEvent":
31       case "PlayerDataUpdatedEvent":
32       case "AccountDataUpdatedEvent":
33         goto("/game")
34         break
35     }
36   });
37 });

});
```

Abbildung 38 Websocket-Implementation im Frontend

4.3.2 Beispiel-Implementation Grundstück-Kauf

Nachfolgend wird der Ablauf zum Kauf eines Grundstücks im Frontend gezeigt. Nach dem Starten des Spiels (Anhang 7) wird die in Abbildung 39 dargestellte Hauptansicht dargestellt. Darauf befinden sich alle relevanten Spielinformationen. Jede teilnehmende Person verfügt über einen Kontostand, eine Anzahl an «Du kommst aus dem Gefängnis frei Karten» und eine Anzahl an Grundstücken. Darüber hinaus ist ersichtlich, welche spielende Person gerade an der Reihe ist, in dem die entsprechende Anzeige gelb hinterlegt wird.

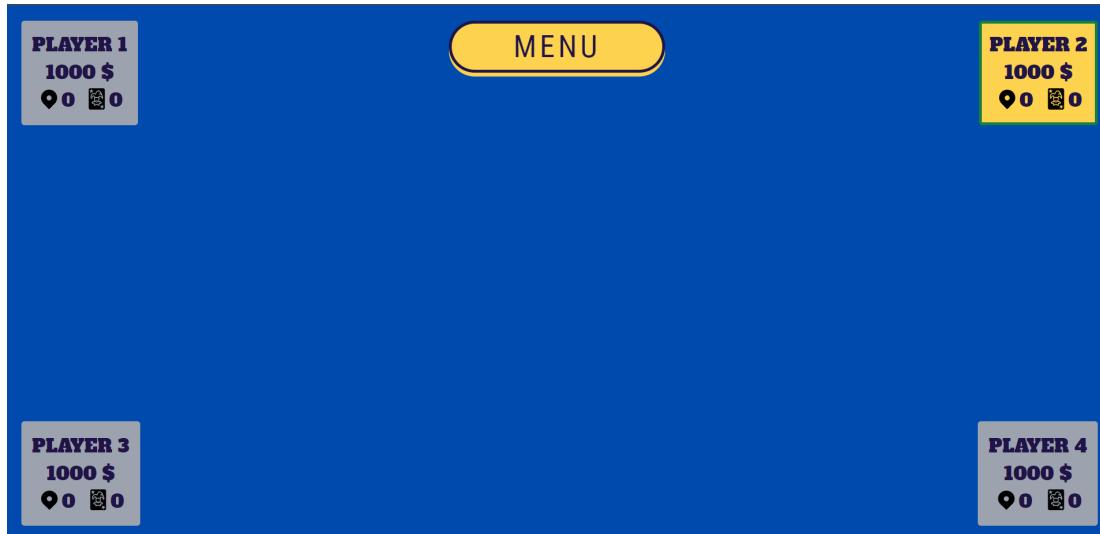


Abbildung 39 Hauptansicht Frontend

Sobald eine Spielfigur auf einem Feld ohne besitzende Person landet, wird gemäss dem UML-Aktivitätsdiagramm in Anhang 1 ein «player on unowned property event» ausgelöst. Dadurch zeigt das Frontend die in Abbildung 40 gezeigte Ansicht. Beim Klick auf den BUY-Button wird ein entsprechender Patch-Request ausgelöst, welcher das angezeigte Grundstück mit der PlayerId der aktuellen Person als OwnerId übermittelt. Die daraufhin erstellte Transaktion löst einen «transaction created event aus».

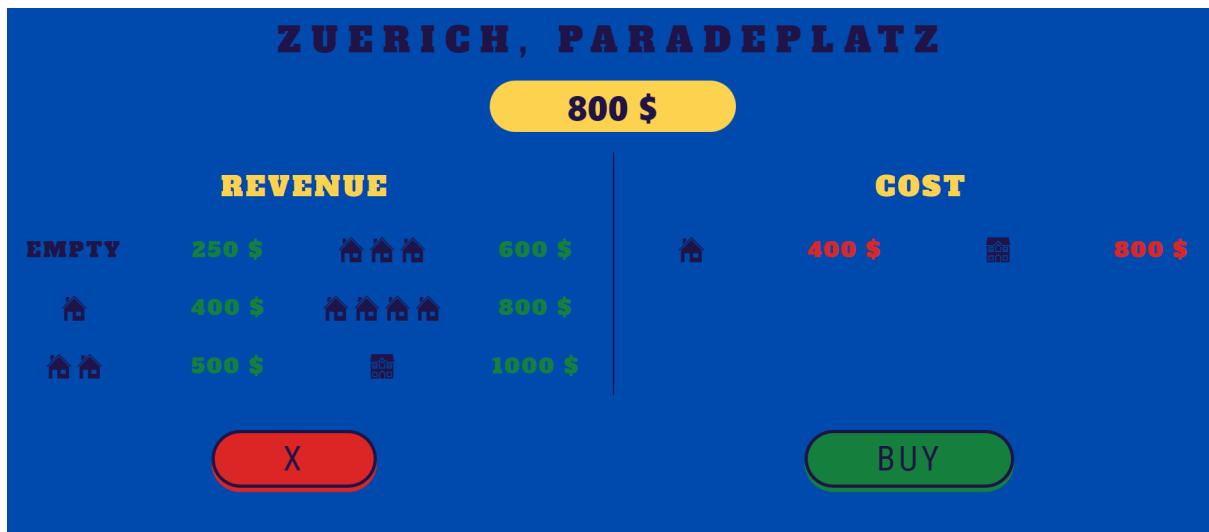


Abbildung 40 Anzeige der Grundstück-Informationen im Frontend

Beim Empfang dieses Events navigiert das Frontend automatisch zur Transaktions-Ansicht, welche in Abbildung 41 zu sehen ist. Von dieser Ansicht wird erst wegnavigiert, wenn ein entsprechendes «transaction resolved event» empfangen wird. Damit ist dieser Vorgang abgeschlossen und es wird wieder die Hauptansicht angezeigt.

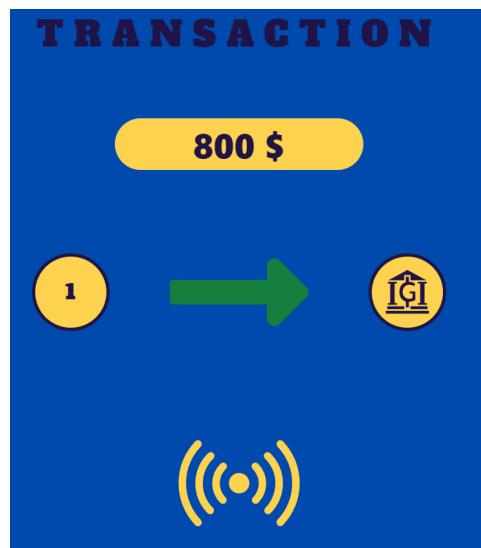


Abbildung 41 Transaktions-Ansicht im Frontend

5 Evaluation

Aufgrund des beschränkten Umfangs dieser Arbeit kann keine umfassende Evaluation mit mehreren spielenden Personen vorgenommen werden. In diesem Abschnitt werden die Möglichkeiten und Limitationen des Systems anhand eines Testspiels zwischen Zwei Personen dokumentiert.

Wie aus dem Plan des Prototyps (Abbildung 25) und der physischen Umsetzung (Abbildung 29) hervorgeht, verfügt das System über einen einzigen Bildschirm. Befinden sich mehrere spielende Personen rund um das Spielbrett ist die Ausrichtung des Bildschirms nicht für jede Person passend. Für die Personen, welche sich am Kopf und am Fusse des Spielbretts befinden, könnte das System die Anzeige automatisch drehen, sobald die Person am Zug ist. Für die Personen auf der Seite ist jedoch keine Softwareseitige Lösung denkbar und das Spielerlebnis ist sicherlich beeinträchtigt. Als Alternative wäre es möglich, die Anzeige auf einem Smartphone vorzunehmen. Dazu müssten sich die Teilnehmenden vorab mit dem Smartphone einloggen. Eine weitere Möglichkeit wäre ein quadratisches Display, bei dem die Anzeige automatisch in die Richtung der jeweiligen Person gedreht wird.

Eine weitere Limitation des vorgestellten Systems stellen die gekauften Grundstücke dar. Anders als in der klassischen Version, erhalten spielende Personen beim Kauf eines Grundstückes keine Karte, welche den Besitz repräsentiert. Das System signalisiert erst beim Betreten eines Grundstücks, ob dies bereits einer Person gehört. Eine Möglichkeit das Spielbrett auf dem Bildschirm darzustellen und die Grundstücksbesitzende anzuzeigen fehlt. Hinzu kommt, dass der erstellte Prototyp durch die Verwendung von öffentlich zugänglicher Hardware über grosse Dimensionen verfügt, obwohl nur 16 statt der üblichen 40 Spielfelder umgesetzt wurden. Für eine umfassendere Version mit mehr Feldern scheint proprietäre Hardware zwingend notwendig.

Die Reaktionszeit der erarbeiteten Lösung ist sehr gut. Wie aus dem Abschnitt 4.1.2 hervorgeht, musste sogar eine künstliche Pause eingebaut werden, damit RFID-Tags nicht zu schnell vom System registriert werden und erst beim Absetzen der Figur ein Event auslösen. Das Zusammenspiel zwischen dem physischen Spielbrett und dem System resultiert in einem flüssigen Spielablauf, welcher optimal durch das System unterstützt wird. Eine weitere Observation aus dem Testspiel ist, dass durch die Verwendung von zusätzlichen Audio-Effekten und Animationen auf dem Bildschirm ein noch immersiveres Spielerlebnis erreicht werden könnte.

Abschliessend ist festzuhalten, dass sich RFID sehr gut eignet, um ein intelligentes Spielbrett zu erstellen. Die Sensoren und Tags lassen sich einfach in die physische Welt integrieren und bieten eine zuverlässige Erkennungsrate.

6 Schlussfolgerung

Die vorliegende Arbeit beschäftigte sich mit der Frage, wie Technologie sinnvoll in das Brettspiel Monopoly integriert werden kann, um das Spielerlebnis positiv zu beeinflussen. Für die Beantwortung wurde eine passende Technologie evaluiert und ein entsprechender Prototyp erstellt.

Die Evaluation einer geeigneten Technologie hat gezeigt, dass sich RFID sehr gut für die Verwendung in Brettspielen eignet. Dies vor allem weil sich die RFID-Sensoren und RFID-Tags relativ unauffällig in die physische Welt integrieren lassen. Nebst der geeigneten Technologie hat diese Arbeit auch spezifische Komponenten evaluiert und getestet. Die Ergebnisse zeigen, dass die MFRC522 Sensoren zusammen mit den Mifare Classic Tags eine zuverlässige Erkennungsrate bei verhältnismässig kleinen Dimensionen erreichen.

Anhand des erstellten Prototyps wurde ansatzweise aufgezeigt, wie sich das Spielerlebnis durch die Integration von Technologie verbessert. Durch das Zusammenspiel zwischen dem intelligenten Spielbrett und der erstellten Software entsteht eine immersive Spielwelt, in der das Bewegen von Spielfiguren eine direkte Reaktion des Systems auslösen. Mithilfe des Prototyps konnten jedoch auch gewisse Limitationen aufgezeigt werden. So würde ein grösseres Spielfeld durch die hohe Anzahl an RFID-Sensoren wohl erhebliche Dimensionen annehmen. Hinzu kommt, dass die Ausrichtung des Displays nicht für alle beteiligten Spielerinnen und Spieler optimal ist.

Im Rahmen dieser Arbeit wurde zudem aufgezeigt, wie DDD und Clean Architecture angewendet werden kann, um eine Domäne in klar definierte Subdomänen zu unterteilen und entsprechende Komponenten zu designen. Die dadurch entstandene Anwendung ist erweiterbar, einfach zu verstehen und verwendet die Sprache der Brettspiel-Domäne. Die vorgestellte Architektur bietet die Möglichkeit, die Spiellogik und das GUI im Internet zu betreiben. Das physische Spielbrett würde dann die Positionsveränderungen via Internet kommunizieren. Dadurch entsteht eine Vielzahl von spannenden Möglichkeiten, wie Brettspiele der Zukunft, unabhängig vom Standort, miteinander kommunizieren können. Die Architektur erlaubt die Implementierung von komplexen Spielvorgängen.

Abschliessend ist festzuhalten, dass durch die Integration von RFID oder Technologie im Allgemeinen in Brettspiele viel Potenzial in Bezug auf ein immersiveres Spielerlebnis entsteht. Diese neue Generation von Brettspielen verbindet die soziale Interaktion zwischen den Spielenden und den schnelllebigen Welten von Computerspielen. Zukünftige Arbeiten sollten sich jedoch vermehrt darauf konzentrieren, die positiven und negativen Auswirkungen dieser neuen Generation von Brettspielen zu untersuchen und zu qualifizieren.

7 Eigenständigkeitserklärung

I hereby declare that I have prepared the present work independently and have used nothing other than the specified aids. All text sections, citations or contents of other authors used have been explicitly marked as such.

Thalwil, 22.01.23

R. Schüller

8 Literaturverzeichnis

- 2N RFID-Badge Mifare Classic 1k RFID Badge, 13.56 MHz. (o. J.). Abgerufen 18. Dezember 2022, von <https://www.brack.ch/2n-rfid-badge-mifare-classic-1k-rfid-badge-311097>
- 50pcs Ntag213 Nfc Tags 13.56mhz Iso14443a. (o. J.). Fruugo. Abgerufen 18. Dezember 2022, von https://www.fruugoschweiz.com/50pcs-ntag213-nfc-tags-1356mhz-iso14443a-nfc-sticker-ntag-213-all-nfc-phone/p-72222006-145105587?language=en&ac=ProductCasterAPI&gclid=CjwKCAiAkfucBhBBEiwAFjbkr5nT1XFhCQEZQYDvA7Y3bOXYESefIKIOViuYF70XVvB0MktYBpqeFRoCp4wQAvD_BwE
- Ahmad, S., Umirzakova, S., Jamil, F., & Whangbo, T. K. (2022). Internet-of-things-enabled serious games: A comprehensive survey. *Future Generation Computer Systems*, 136, 67–83. <https://doi.org/10.1016/j.future.2022.05.026>
- Arduino-rfid-PN5180. (2022). [C++]. playfultechnology. <https://github.com/playfultechnology/arduino-rfid-PN5180> (Original work published 2018)
- Balboa, M. A. (2022). MFRC522 [C++]. <https://github.com/miguelbalboa/rfid> (Original work published 2012)
- Banks, F. (2022, Oktober 26). *Domain-Driven Design: What is it and how do you use it?* <https://blog.airbrake.io/blog/software-design/domain-driven-design>
- Bohn, J. (2004). *The Smart Jigsaw Puzzle Assistant: Using RFID Technology for Building Augmented Real-World Games.*
- Connell, C. (2015, Februar 6). *What's The Difference Between Measuring Location By UWB, Wi-Fi, and Bluetooth?* Electronic Design. <https://www.electronicdesign.com/technologies/communications/article/21800581/whats-the-difference-between-measuring-location-by-uwb-wifi-and-bluetooth>
- Donovan, R., & Au-Yeung, J. (2020, März 2). *Best practices for REST API design.* Stack Overflow Blog. <https://stackoverflow.blog/2020/03/02/best-practices-for-rest-api-design/>
- Floerkemeier, C., & Mattern, F. (2006). *Smart Playing Cards – Enhancing the Gaming Experience with RFID.* 10.
- Hallek, P. (2019, August 20). Precise Realtime Indoor Localization With Raspberry Pi And Ultra-Wideband Technology (Decawave.... Medium. <https://medium.com/@newforestberlin/precise-realtime-indoor-localization-with-raspberry-pi-and-ultra-wideband-technology-decawave-191e4e2daa8c>

- Han, J., Kim, K., Jung, K., & Lee, K.-O. (2010). RFID-BASED DIGITAL BOARD GAME PLATFORMS. *Computing and Informatics*, 29, 1141–1158.
- Hinske, S., & Langheinrich, M. (2007). *An RFID-based Infrastructure for Automatically Determining the Position and Orientation of Game Objects in Tabletop Games*. 21.
- In-Depth: What is RFID? How It Works? Interface RC522 with Arduino*. (2018, Juli 30). Last Minute Engineers. <https://lastminuteengineers.com/how-rfid-works-rc522-arduino-tutorial/>
- Joy-it RB-LCD7-2 (Display). (o. J.). Abgerufen 11. Dezember 2022, von <https://www.digitec.ch/de/s1/product/joy-it-rb-lcd7-2-display-elektronikmodul-12247799?supplier=406802>
- Konkel, M., Leung, V., Ullmer, B., & Hu, C. (2004). Tagaboo: A collaborative children's game based upon wearable RFID technology. *Personal and Ubiquitous Computing*, 8, 382–384. <https://doi.org/10.1007/s00779-004-0302-y>
- Kosa, M., & Spronck, P. (2018, August). *What Tabletop Players Think about Augmented Tabletop Games: A Content Analysis*. <https://doi.org/10.1145/3235765.3235782>
- Lee, W., Woo, W., & Lee, J. (2005). TARBoard: Tangible Augmented Reality System for Table-top Game Environment. *Personal Computing*, 5, 0–4.
- Lima, S. V., & Sarinho, V. T. (2019). *Developing an Open Hardware RFID Architecture for Electronic Board Games*. 4.
- Magerkurth, C., Memisoglu, M., Engelke, T., & Streitz, N. (2004). Towards the next Generation of Tabletop Gaming Experiences. *Proceedings of Graphics Interface 2004*, 73–80.
- Martin, R. C. (2012, August 13). *Clean Coder Blog*. <https://blog.cleancoder.com/uncle-bob/2012/08/13/the-clean-architecture.html>
- Martin, R. C. (2017). *Clean Architecture: A Craftsman's Guide to Software Structure and Design* (1st Aufl.). Prentice Hall Press.
- Masse, M. (2011). *REST API design rulebook: Designing consistent RESTful web service interfaces*. O'Reilly Media, Inc.
- Mfrc522. (2022). [Python]. Pi My Life Up. <https://github.com/pimylifeup/MFRC522-python> (Original work published 2017)
- MIRACLE TECH (Regisseur). (2020, Juli 22). *Multiple RC522 arduino rfid 2 (10 EA RC522 Arduino RFID demonstration video)*. <https://www.youtube.com/watch?v=2565HpoITOY>

Miskovic, J. (2022, Oktober 19). *When To Use PATCH vs. PUT in Professional REST APIs – Josip Miskovic*.

<https://josipmisko.com/posts/patch-vs-put-rest-api>

Monopoly Game Rules. (o. J.). Abgerufen 1. Januar 2023, von

<https://www.ultraboardgames.com/monopoly/game-rules.php>

NTag215 Mini Rfid NFC Tag. (o. J.). aliexpress.com. Abgerufen 18. Dezember 2022, von https://de.aliexpress.com/item/1005002713268743.html?src=ibdm_d03p0558e02r02&sk=&aff_platform=&aff_trace_key=&af=&cv=&cn=&dp=

Playful Technology (Regisseur). (2017, Dezember 1). *Escape Room Object Placement Puzzle using Arduino/RFID*. https://www.youtube.com/watch?v=f_f_5cL0Pd0

Playful Technology (Regisseur). (2018, Dezember 5). *RFID Roundup!* <https://www.youtube.com/watch?v=98GXrixOM4c>

PN532 NFC/RFID controller, Adafruit. (o. J.). Abgerufen 15. Dezember 2022, von <https://www.distrelec.ch/en/pn532-nfc-rfid-controller-adafruit-364/p/30129234>

Rdm6300 ID Kartenleser Modul Rfid RF Modul Uart Serieller Ausgang. (o. J.). Fruugo. Abgerufen 15. Dezember 2022, von https://www.fruugoschweiz.com/rdm6300-id-kartenleser-modul-rfid-rf-modul-uart-serieller-ausgang/p-127472603-267478941?language=de&ac=ProductCasterAPI&asc=pmax&gclid=Cj0KCQiAqOucBhDrARIsAPCQL1ZJzEy0ikvN4iD8iFtkB5jUiS1yWKIdlbRqaJEeXVVYXOvPMqqpxpcaAuTwEALw_wcB

RFID RC522 Reader Writer. (o. J.). Abgerufen 15. Dezember 2022, von <https://www.rajguruelectronics.com/ProductView?tokDatRef=MTI5Mg==&tokenId=NTI=&product=RC522%20RFID%20READER%20WRITER>

Rose, K., Eldridge, S., & Chapin, L. (2015). The internet of things: An overview. *The internet society (ISOC)*, 80, 1–50.

Silverio-Fernández, M., Renukappa, S., & Suresh, S. (2018). What is a smart device? - A conceptualisation within the paradigm of the internet of things. *Visualization in Engineering*, 6(1), 3. <https://doi.org/10.1186/s40327-018-0063-8>

Sivakami, N. (2018). *COMPARATIVE STUDY OF BARCODE, QR-CODE AND RFID SYSTEM IN LIBRARY ENVIRONMENT*. 1(1), 5.

Tan, J., & Rau, P.-L. P. (2015). A Design of Augmented Tabletop Game Based on RFID Technology. *Procedia Manufacturing*, 3, 2142–2148. <https://doi.org/10.1016/j.promfg.2015.07.353>

Technavio. (2020, Januar 14). How Technology is Upgrading the Board Game Industry. *How Technology Is Upgrading the Board Game Industry*. <https://blog.technavio.org/blog/technology-board-game-industry>

Tilkov, S. (2021, Februar 3). *Is Domain-driven Design overrated?* <https://www.innoq.com/en/blog/is-domain-driven-design-overrated/>

Tracking movement of game pieces on a board. (2014, Februar 13).

<https://forums.raspberrypi.com/viewtopic.php?t=69430>

Vasconcelos, R. (2022, Oktober 7). *Clean Architecture: The concept behind the code*. DEV Community.

<https://dev.to/rubemfsv/clean-architecture-the-concept-behind-the-code-52do>

Vernon, V. (2013). *Implementing Domain-Driven Design* (1st Aufl.). Addison-Wesley Professional.

Vernon, V. (2016). *Domain-driven design distilled*. Addison-Wesley Professional.

Want, R. (2006). An Introduction to RFID Technology. *IEEE Pervasive Computing*, 5(1), 25–33.

<https://doi.org/10.1109/MPRV.2006.2>

Xystec Aktiver USB-3.0-Hub (USB A). (o. J.). Abgerufen 18. Dezember 2022, von

<https://www.digitec.ch/de/s1/product/xystec-aktiver-usb-30-hub-usb-a-dockingstation-usb-hub-18184958?supplier=4270853>

9 Abbildungsverzeichnis

Abbildung 1 The Clean Architecture Diagram (Martin, 2012).....	6
Abbildung 2 Kamera-Vorrichtung für das Tracking von Spielfiguren	13
Abbildung 3 Nahfeld-RFID Kommunikation (Want, 2006, S. 26).....	14
Abbildung 4 RDM6300 mit Antenne (Rdm6300 ID Kartenleser Modul Rfid RF Modul Uart Serieller Ausgang, o. J.), MFRC522 (RFID RC522 Reader Writer, o. J.), PN532 (PN532 NFC/RFID controller, Adafruit, o. J.), (arduino-rfid-PN5180, 2018/2022).....	18
Abbildung 5 Konfiguration eines MFRC522 mit Arduino Uno	20
Abbildung 6 Konfiguration von vier MFRC522 mit Arduino Uno	21
Abbildung 7 RFID Badge (2N RFID-Badge Mifare Classic 1k RFID Badge, 13.56 MHz, o. J.).....	21
Abbildung 8 Ntag213 (50pcs Ntag213 Nfc Tags 13.56mhz Iso14443a, o. J.)	21
Abbildung 9 NTag215 (NTag215 Mini Rfid NFC Tag, o. J.)	21
Abbildung 10 Konfiguration von einem MFRC522 mit Raspberry Pi 3B+	23
Abbildung 11 Konfiguration der LEDs mit dem Arduino Uno	24
Abbildung 12 (links) LC Display von Joy-IT welcher im Rahmen dieses Projekts verwendet wird (Joy-it RB-LCD7-2 (Display), o. J.).....	25
Abbildung 13 (rechts) Aktiver USB-HUB (Xystec Aktiver USB-3.0-Hub (USB A), o. J.).....	25
Abbildung 14 UML-Use-Case Diagramm	26
Abbildung 15 Entitäten Sensor Subdomäne.....	28
Abbildung 16 Player und Account Entität	28
Abbildung 17 Aggregate Game.....	29
Abbildung 18 Board Value Object.....	29
Abbildung 19 Entitäten und Value Objects für das Modellieren der Grundstücke	30
Abbildung 20 Entitäten und Value Objects rund um GameEvents.....	30
Abbildung 21 Entitäten Finance Subdomäne.....	31
Abbildung 22 UML-Komponenten-Diagramm interne Events	32
Abbildung 23 UML-Komponenten-Diagramm externe Events.....	33
Abbildung 24 UML-Deployment-Diagramm	34
Abbildung 25 Monopoly-Spielbrett Prototyp	35
Abbildung 26 Innenansicht Prototyp (nicht massstabsgetreu).....	36
Abbildung 27 Innenansicht Prototyp (Konstruktion RFID, nicht massstabsgetreu).....	37
Abbildung 28 Prototyp Innenansicht.....	38
Abbildung 29 Prototyp Aussenansicht.....	38
Abbildung 30 Code-Ausschnitt Kommunikation mit MFRC522	39
Abbildung 31 Python-Code zur Umwandlung der eingehenden seriellen Nachrichten in http-Anfragen	40

Abbildung 32 Ordnerstruktur nach Clean Architecture	42
Abbildung 33 http-Endpunkte.....	43
Abbildung 34 Implementierung DDD-Eventing.....	44
Abbildung 35 Implementierung Domänen-Event Transaction Aggregate	44
Abbildung 36 Implementierung EventHandler	45
Abbildung 37 Implementation Websocket zur Verteilung von Domänen-Events.....	45
Abbildung 38 Websocket-Implementation im Frontend.....	47
Abbildung 39 Hauptansicht Frontend	48
Abbildung 40 Anzeige der Grundstück-Informationen im Frontend.....	49
Abbildung 41 Transaktions-Ansicht im Frontend.....	49

10 Tabellenverzeichnis

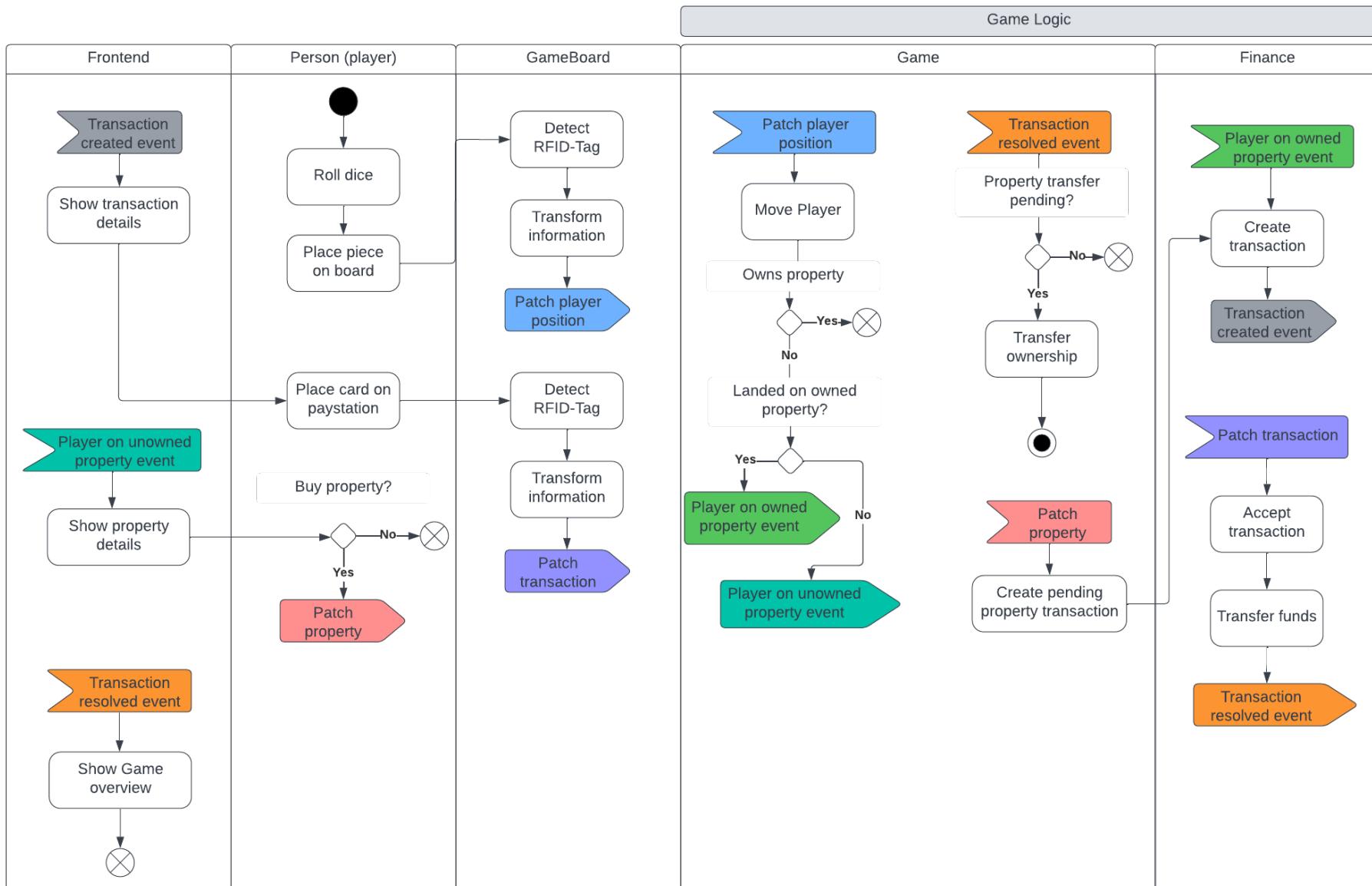
Tabelle 1 Übersicht Felder Monopoly	16
Tabelle 2 Pin Layout MFRC522 mit Arduino Uno (In-Depth, 2018)	20
Tabelle 3 RFID-Tags.....	21
Tabelle 4 Pin Layout RFID RC522 mit Raspberry Pi 3B+.....	23
Tabelle 5 Auswertung der Erkennungsrate der RFID-Tags.....	41

11 Anhang

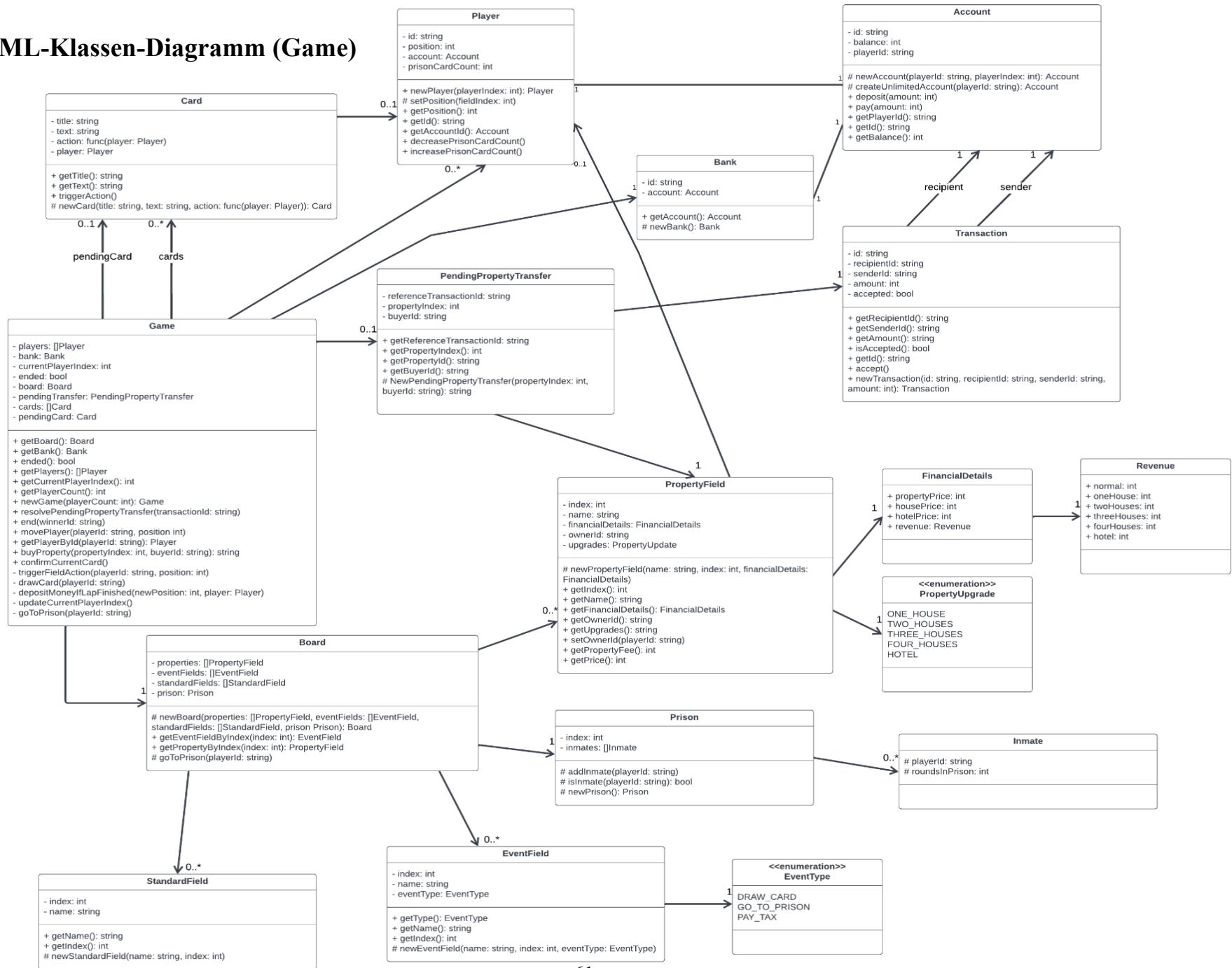
11.1 Anhangsverzeichnis

Anhang 1 UML-Aktivitätsdiagramm	60
Anhang 2 UML-Klassen-Diagramm (Game).....	61
Anhang 3 UML-Klassen-Diagramm (Sensor)	62
Anhang 4 UML-Klassen-Diagramm (Events)	62
Anhang 5 Code Bezahlstation	63
Anhang 6 Ordnerstruktur	64
Anhang 7 Frontend.....	65

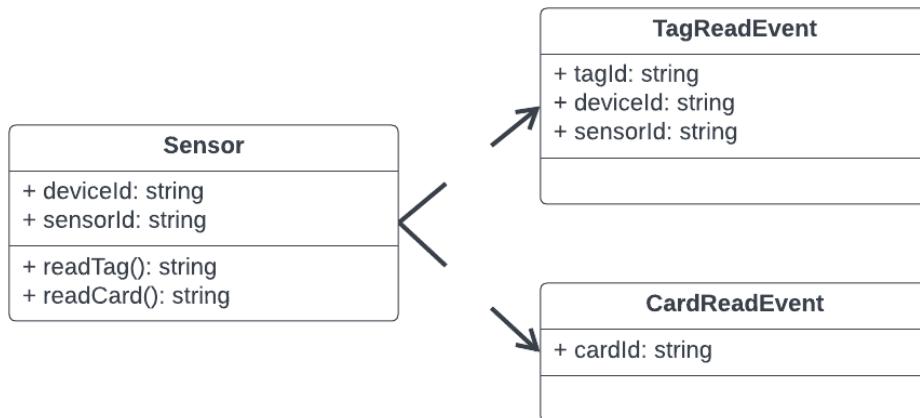
Anhang 1 UML-Aktivitätsdiagramm



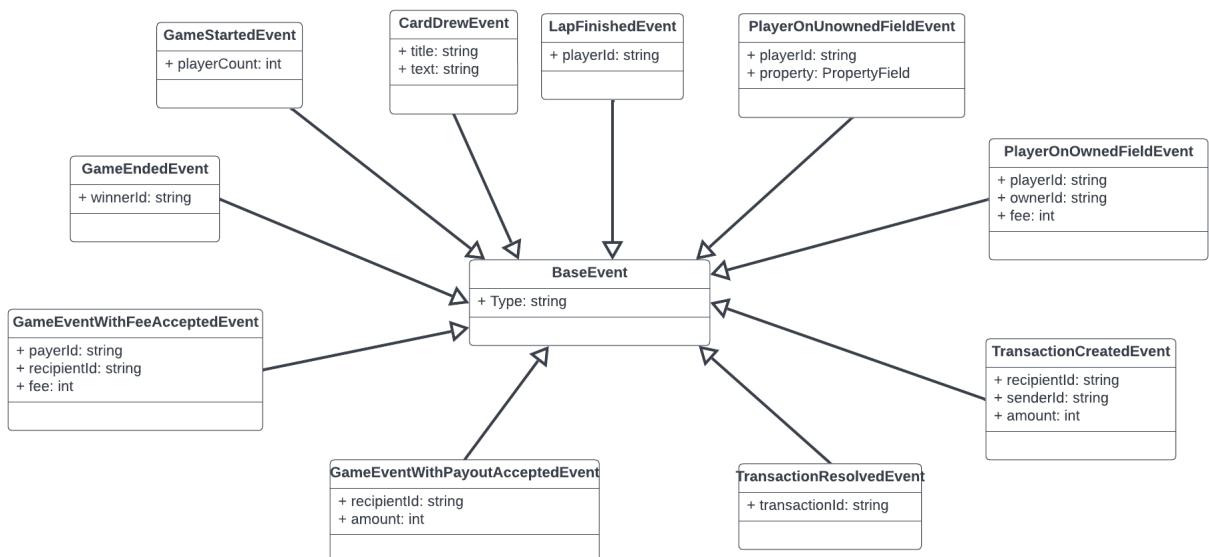
Anhang 2 UML-Klassen-Diagramm (Game)



Anhang 3 UML-Klassen-Diagramm (Sensor)



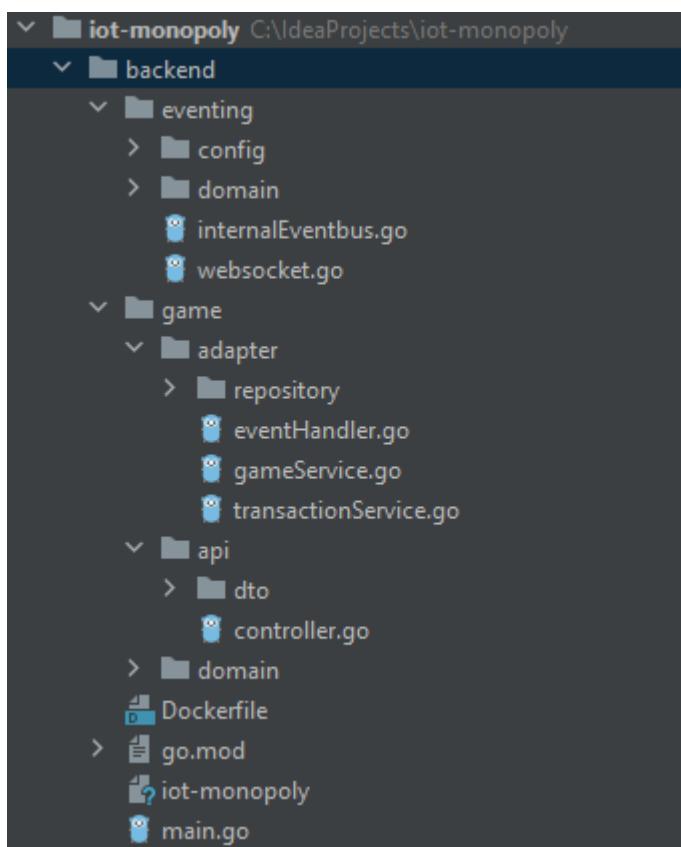
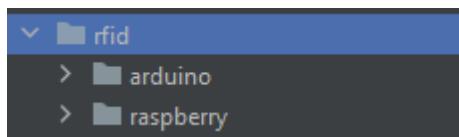
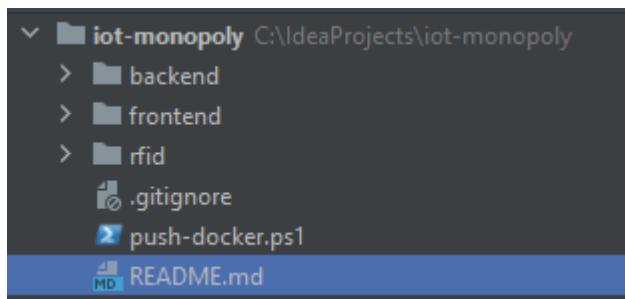
Anhang 4 UML-Klassen-Diagramm (Events)



Anhang 5 Code Bezahlstation

```
28     continue_reading = True
29
30     tagIdToAccountIdMap = {"33-A8-8A-10": "Account_1", "1304-B6-1A": "Account_2",
31                         "43-F1-E70E": "Account_3", "A3-D9-350F": "Account_4"}
32
33     def map_tag_id_to_account_id(tag_id):
34         account_id = tagIdToAccountIdMap[tag_id]
35         print("mapped tagId: " + tag_id + " to accountId: " + str(account_id))
36         return account_id
37
38     # Capture SIGINT for cleanup when the script is aborted
39     def end_read(signal,frame):
40         global continue_reading
41         print ("Ctrl+C captured, ending read.")
42         continue_reading = False
43
44     # Hook the SIGINT
45     signal.signal(signal.SIGINT, end_read)
46
47     # Create an object of the class MFRC522
48     MIFAREReader = MFRC522.MFRC522()
49
50     # This loop keeps checking for chips. If one is near it will get the UID and authenticate
51     while continue_reading:
52
53         # Scan for cards
54         (status,TagType) = MIFAREReader.MFRC522_Request(MIFAREReader.PICC_REQIDL)
55
56         # If a card is found
57         if status == MIFAREReader.MI_OK:
58
59             # Get the UID of the card
60             (status, uid) = MIFAREReader.MFRC522_Anticoll()
61             tag_id = "{}-{}-{}-{}".format(uid[0], uid[1], uid[2], uid[3])
62             try:
63                 requests.patch("http://localhost:3000/games/current/transactions/latest",
64                                 data={"accepted": True, "senderId": map_tag_id_to_account_id(tag_id)})
65             except Exception:
66                 print("Request failed")
```

Anhang 6 Ordnerstruktur



Anhang 7 Frontend



PLAYER 1
1000 \$
📍 0 🏡 0

MENU

PLAYER 2
1000 \$
📍 0 🏡 0

PLAYER 3
1000 \$
📍 0 🏡 0

PLAYER 4
1000 \$
📍 0 🏡 0

ZUERICH, PARADEPLATZ

800 \$

REVENUE

EMPTY	250 \$	🏡🏡🏡	600 \$
🏡	400 \$	🏡🏡🏡🏡	800 \$
🏡🏡	500 \$	🏡	1000 \$

COST

🏡	400 \$	🏡🏡	800 \$
---	--------	----	--------

X

BUY

TRANSACTION

800 \$



ESCAPE FROM PRISON

KEEP THIS CARD AND ESCAPE FROM
PRISON NEXT TIME

OK

TAX BILL

**YOU RECEIVED A BILL FOR THE
FEDERAL TAXES OF 200 \$**

OK

YOU INHERITED

**YOU'RE MENTIONED IN THE
TESTAMENT OF YOUR AUNT. YOU
RECEIVE 100 \$.**

OK