HAW HAMBURG

Autoren: Philip Borchert Matrikelnr.: 2237188, Alexander Könemann: 2434660

Dokumentationstyp: arc42

Version: 1.5, Stand 20.01.2022

Abtestat - Vorstellungstermin: 27.01.2022 um 09:00h

Abtestat – Abgabetermin GitLab: 23.01.2022

Project Repository GitLab: *https://git.haw-hamburg.de/abx276/vs-bck*

Modul: Verteilte Systeme – WS 2021/2022

Dozent: Prof. Martin Becke

Praktikumsgruppe: 02

Inhaltsverzeichnis

1) Versionsänderungsdokumentation

- 11.11.2021 V0.1: Alexander Könemann: Alternative Architektur eingefügt (3 Schichten Architektur).
- 13.11.2021 V0.2: Philip Borchert: Layout der Dokumentation überarbeitet.
- 14.11.2021 V0.3: Philip und Alex: Skizzen GUI eingefügt.
- 15.11.2021 V0.4, Philip Borchert und Alexander Könemann: Erste Version aus Zusammenführung Teildokumentation erstellt.
- 17.11.2021 V.05, Philip Borchert:
    - SD0 überarbeitet (ohne Fehlerfälle)
    - UC2 Anpassung Darstellung Controller
    - Stakeholder: Kunde Erwartungen hinzugefügt
    - Stakeholder: Deployment mit Erwartung hinzugefügt
    - Lösungsstrategie: Fairness in Textform hinzugefügt
- 19.11.2021: V0.6: Alexander Könemann
    - Diagramme in technischen Kontext verschoben
    - Openness Kriterien Tabelle eingefügt
    - Neues Kapitel Softwarearchitektur und Verteilungssicht/Middleware eingefügt, um 2 Dokumente zu vermeiden.
    - Anhang hinzugefügt und verworfene Architekturoption MVC Pattern in den Anhang verschoben
    - Erste Version der Diagramme DataLevel (GameModel und Player) eingefügt
- 28.11.21: V0.7 Alex und Philip:
    - Technischer Kontext: Java API hinzugefügt
    - Neue Version der 3-Schichten-Architektur hinzugefügt
- 29.11.21: V0.8 Alex:
    - Sequenzdiagramme GameLoopService, GridCollisionService und MovePlayerService eingefügt
    - Komponentendiagramme Data Level und Processing Level eingefügt
    - Architektur aktualisiert
    - Abbildungsverzeichnis hinzugefügt
    - Zweite Ebene im Inhaltsverzeichnis eingefügt
    - Erster Entwurf des Entwurfs der RPC-Schnittstelle eingefügt. Basis sind die Vorlesungsinhalte
- 03.12.21: V0.9 Alex:
    - Korrektur der Gewinn/Verlust Bedingungen. Ergänzung, was passiert, wenn man in seinen eigenen Schlangenkörper fährt (Spieler verliert) und Ergänzung, falls man versucht eine 180 Drehung zu machen (nicht erlaubt. Spieler kann nicht in seinen eigenen Hals fahren, die Bewegung wird dann durch MovePlayerService korrigiert)
    - Korrektur Klassendiagrame Data Level, Processing Level und User Interface Level

- 05.12.21: V1.0 beide:
    - Diagramme Bausteinsicht in 3 Ebenen eingefügt
    - Aktualisierte Kompaktansicht 3-Schichtenarchitektur
    - Aktualisierte Sequenzdiagramm: CommunicationComponent
    - Entwurf RPC-Schnittstelle aktualisiert
- 08.01.22: V1.1 beide:
    - Entwurf RPC Protokoll fertigestellt für Präsentation Review
- 13.01.22: V1.2 Alex:
    - Bausteinsicht an Code angeglichen: Alle Komponenten außer Middleware
    - Klassendiagramme für UserInterfaceLevel, ProcessingLevel und DataLevel aktualisiert
    - Sequenzdiagramme Spiellogik überarbeitet (Package GameController)
    - RPC Protokoll überarbeitet (Feedback Review: Methodennamen durch Datentypen ersetzt)
    - Termine und Projektplan aktualisiert, Quellen ergänzt
    - Funktionsbeschreibung im Kapitel Lösungsstrategie überarbeitet
- 18.01.22: V1.3 Philip:
    - Sequenzdiagramme hinzugefügt / überarbeitet.
        - SD: Update gamelist
        - SD: Join game
        - SD: Set host game
        - SD: Host game response
        - SD: Middleware ClientStubs
        - SD: Middleware ServerStubs
    - Bausteinsicht überarbeitet
        - GameLoopServer Start Methode hinzugefügt.
    - RPC Protokoll PSH flag Beschreibung hinzugefügt.
- 19.01.22: V1.4 Alex:
    - Abbildungsverzeichnis vervollständigt
    - Readme des Repositories aktualisert, Link zum Repo ins Dokument eingefügt
- 20.01.22: V1.5 Alex | Philip
    - Finales überarbeiten/korrigieren des Dokumentes
    - GameView -> ApplicationStubs -> startServer() hinzugefügt
    - GameView -> ApplicationStubs -> setTronGameMenu hinzugefügt
    - Finale Bausteinsicht eingefügt

2) Abbildungsverzeichnis

3) Einführung und Ziele

Ziel des Projektes: Entwicklung einer Zweispieler Standalone Tron Applikation mit lokalen Multiplayer

Spielprinzip: Jeder Spieler hat eine eigene Sicht, in der er sich bewegen kann. Ein Spieler verliert und scheidet aus, wenn er sein eigener Pfad, den Pfad des Gegenspielers oder den Rand des Spielfeldes berührt.

a) Requirements Overview[1]

- Tron ist ein lokale Mehrspieler Applikation mit maximal 2 Spielern pro Spiel. Ein System kann mehrere Spiele unterstützen.

i. Spielbeschreibung:

- 2 Spieler treten in einem Spiel gegeneinander an und werden als bike bzw. Schlange repräsentiert.
- Das Spielfeld wird durch eine Wand abgegrenzt:

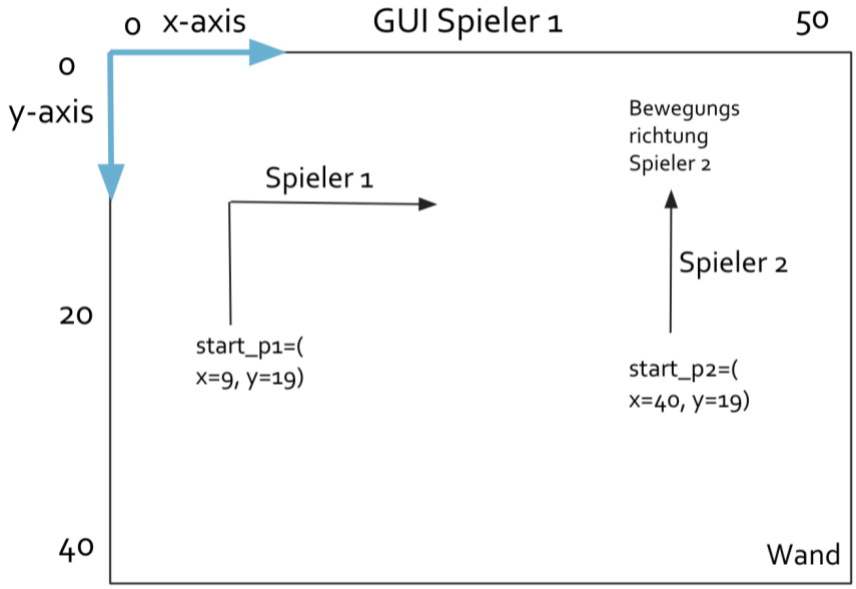


Figure 1: Das Spielfeld des Tron Spiels wird durch ein Gitter repräsentiert (eigene Darstellung)

- Das Spielfeld ist ein Gitter, dass 50 Felder breit (von x=0 bis x=49) und 40 Felder hoch (von y=0 bis y=39) ist
- Die Spieler starten vertikal zentriert: siehe Startpositionen *start_p1* und *start_p2*
- **Verlieren des Spiels:**

[1] Siehe Protokoll Kundenanforderungen

- Wenn ein Spieler in eine Wand fährt bzw. diese berührt, verliert er das Spiel
- Wenn ein Spieler mit seinem Kopf in seinen eigenen Schlangenkörper fährt bzw. diese berührt, verliert er das Spiel
    - Sonderfall: Ein Spieler verliert nicht, wenn er versucht, eine 180 Grad Drehung zu machen, um in seinen eigenen Tail zu fahren. Diese Bewegung ist nicht zulässig, da man sich nicht in den eigenen Hals beißen kann. Die Bewegung wird dann in die Gegenrichtung verkehrt und der Spieler fährt entsprechend geradeaus.
- Wenn ein Spieler den Schlangenkörper des anderen Spielers mit seinem Kopf berührt, verliert er das Siel
- Ein Spieler verliert, falls er das Spiel verlässt.
- Falls sich beide Spieler frontal mit den Köpfen berühren, verlieren beide.

- **Gewinnen des Spiels:**
    - Ein Spieler hat nur dann gewonnen, wenn der andere Spieler verloren hat und er selbst nicht verloren hat
- Sobald ein*e Spieler*in das Spiel gewinnt, wird der Gewinner verkündet.

ii. Spielrepräsentation:

- Beide Spieler*innen spielen in der Prototyp Version auf einem Fenster, in dem beide Motorräder angezeigt werden. In der Abgabeversion hat jeder Spieler im Mehrspieler Modus eine eigene GUI, im lokalen Modus wird ein Fenster für beide Spieler verwendet.

iii. Spielsteuerung:

- Steuerung über Tastatur:

Funktion	**Taste Spieler links (Host)**	**Taste Spieler rechts (Client)**
Bewegung nach oben	w	Pfeiltaste oben
Bewegung nach unten	s	Pfeiltaste unten
Bewegung nach links	a	Pfeiltaste links
Bewegung nach rechts	d	Pfeiltaste rechts

- Ein*e Spieler*in bewegt sich automatisch geradeaus. Der*die Spieler*n kann die Richtung manipulieren, aber nicht zurück.
- Jeder Spieler kann sich selbst einem Spiel über einen Spielraum-Namen zuordnen.

- Nach Beendigung des Spiels kann ein neues Spiel gestartet werden.

iv. Menüführung:

- Der Userflow verläuft über drei verschiedene Views in der GUI des Nutzers. Beim Start der App wird zuerst das Startmenü angezeigt, anschließend gelangt man zur Ansicht des Spielfeldes und im letzten Schritt zum Menü, in dem der Gewinner angezeigt wird. Von diesem kann man wieder zum Startmenü gelangen:

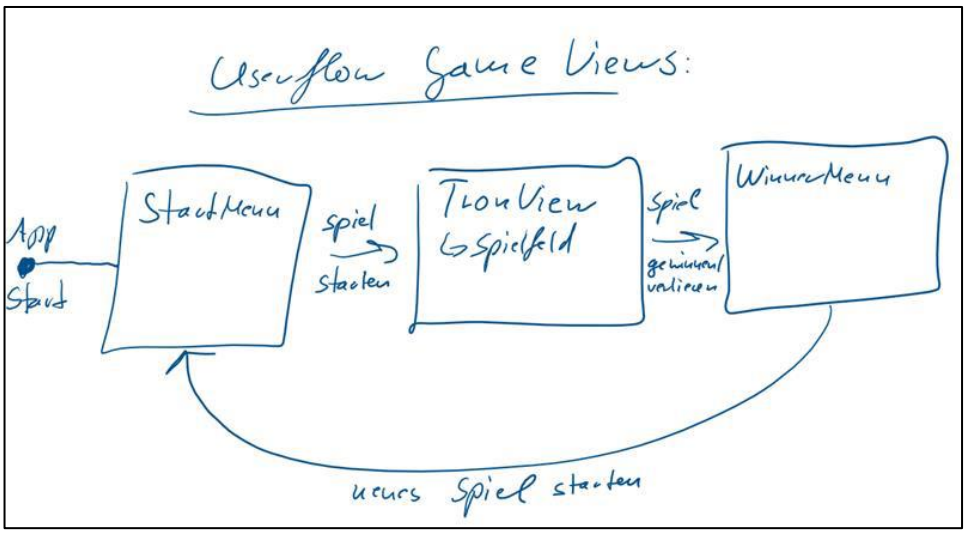


Figure 2: Übersicht der 3 Views des UI (eigene Darstellung)

- Das Startmenü ist bereits für eine spätere Weiterentwicklung des Prototypen vorbereitet und beinhaltet bereits Ansichten zum Beitreten von Spielen, die in der Standalone Applikation ausgeblendet werden:

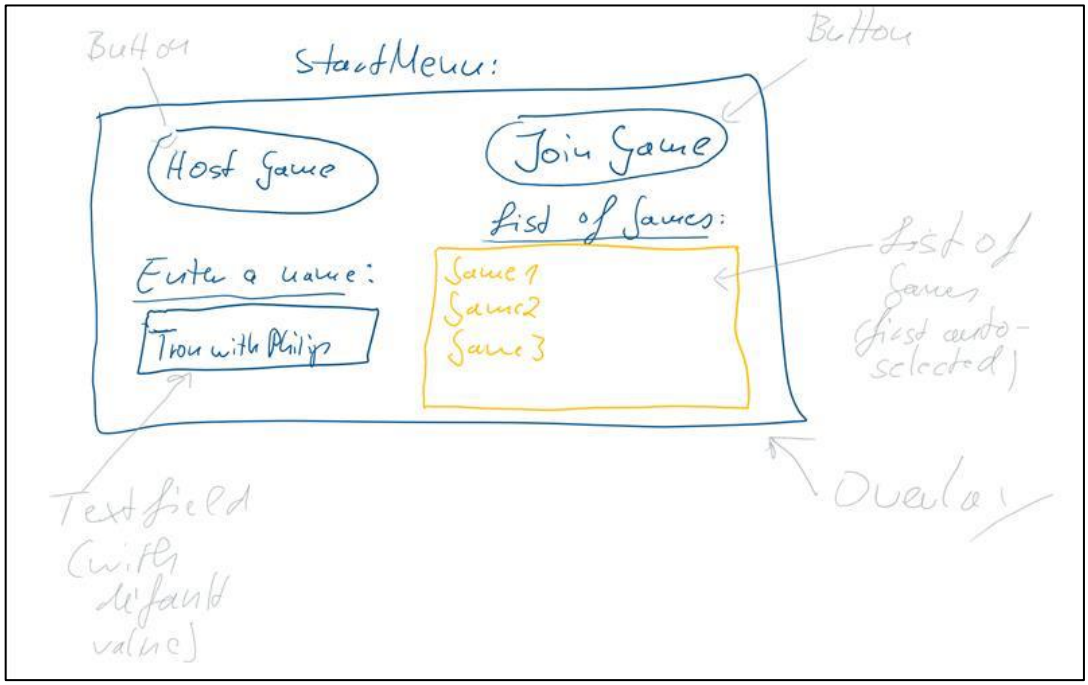


Figure 3: Startmenu der Applikation (eigene Darstellung)

- Die TronView bzw. das Spielfeld wurde bereits in **Error! Reference source not found.** skizziert. Nachdem ein Spieler gewonnen hat, wird ins letzte Menü gewechselt. Dieses zeigt den Gewinner an und gibt dem User die Möglichkeit ins Startmenü zurückzukehren, um ein neues Spiel zu starten:

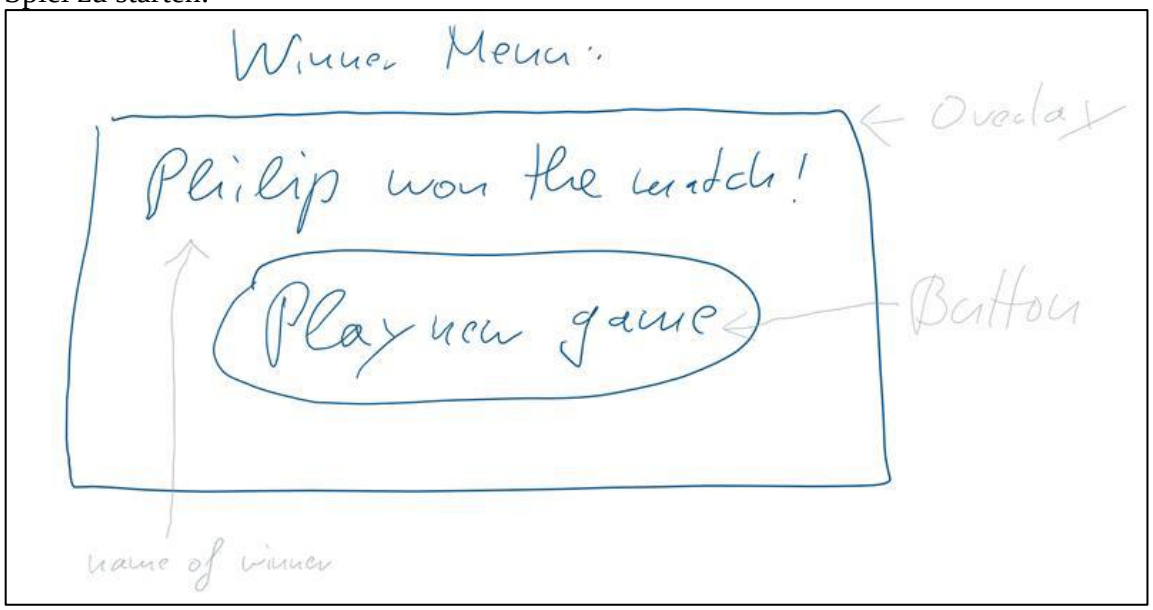


Figure 4: Gewinner Menü der Anwendung (eigene Darstellung)

b) Qualitätsziele

- Das Ziel ist es einen lauffähigen Prototyp für den Kunden zu entwickeln. Das System muss gewährleisten, dass keine Benachteiligung der Spieler z.B. durch unterschiedliche Hardware entstehen darf.

Zur **Unterstützung** des **Softwareentwicklungsprozesses** werden verschiedene Softwaretools eingesetzt. Übersicht der Tools nach Einsatzzweck:

- **Kommunikation**: Microsoft Teams
    - Das Tool MS Teams wurde ausgewählt, da dort neben Meetings auch Dateien und Nachrichten im Projektteam ausgetauscht werden können
- **Design**: draw.io und Lucidchart
    - Für die Erstellung von UML2.0 und Sequenzdiagrammen wird draw.io und Lucidchart.com eingesetzt.
- **Versionskontrolle**: GitLab
    - GIT als Versionskontrolle ist das gängigste Tool in der Softwareentwicklung. Auch wir sind mit GIT entsprechend aufgestellt. Mit Hilfe von GIT sind alle Projektentwicklungen und -Änderungen jederzeit einsehbar.
- **Dokumentation**: Word und OneDrive
    - Microsoft Word in einem geteilten OneDrive Folder.
- **Ticketing**: Trello
    - Mit Hilfe eines Trello Boards haben wir jederzeit einen Überblick über den gesamten Projektverlauf und den aktuellen Stand. Durch die Organisation der Tickets in Sprints (mit Deadlines) können wir effizient und transparent das Projekt vorantreiben. Das Board ist unter folgenden Link zu erreichen: *https://trello.com/b/LWeeNzIt/vs-kanban-board*

c) Stakeholder

Rolle/Name	**Kontakt**	**Erwartungen**
Kunde	Repräsentiert durch: Martin Becke	• Entwicklung des Spieles „TRON“ • Entwicklung in der Programmiersprache JAVA • Entwicklung eines Prototyps als eigenständige Version • Feste Methode für Dokumentation • Dokumentation passt zum Code • Code passend zu Dokumentation • Lauffähige Version bis zum Ende des Semesters • Nach Absprache festgelegte Projektbestandteile werden durch Entwickler geliefert. Diese werden im Seminar an der HAW verhandelt (Donnerstag vormittags) • Kundenfeedback wird durch regelmäßige Review Termine

		(mindestens 4) eingeholt • Finale Übergabe des Produktes und Dokumentation bis spätestens 23.01.22
Entwickler	Philip Borchert Alexander Könemann	• Kundenfeedback zu den entwickelten Inkrementen je Review Termin • Regelmäßiger Austausch untereinander (durch Jour Fixe jeden Dienstag) • Erlangung der PVL im Modul Verteilte Systeme • Verbesserung der Fähigkeiten zur Entwicklung und Dokumentation von verteilten Systemen durch praktische Anwendung und Kundenfeedback • Zeitbudget PVL laut Modulhandbuch: o 12 Stunden Praktikum o 132 Stunden Eigenarbeit für das Modul: o 50% der Eigenarbeit wird für die Vor- und Nachbereitung der Vorlesung aufgebracht. o Es verbleiben 78 Stunden Projektarbeit verteilt auf 14 Wochen: 5,6 Stunden pro Woche je Teammitglied
Support	Daniel Sarnow	• Technischer Support und Bereitstellung einer view-library in Java • Erwartet freundliche und frühzeitige Meldung von Unterstützungsbedarf seitens der Entwickler • Erwartet eine klare Formulierung der Problemstellung seitens
Deployment	Philip Borchert	• Der Protoyp wird auf einem PC für den Kunden zur Übergabe bereitgestellt. Für weiteres deployment ist der Kunde selbst verantwortlich (.jar Datei, Projekt Repository und Dokumentation werden ausgeliefert).

d) Randbedingungen

Versionierungsrichtlinien:

- Jede Änderung ist mit {Datum}{Name}{Kurzbeschreibung} zu protokollieren.
- Nach jeder Äderung ist die Versionsnummer zu inkrementieren.

4) Nebenbedingungen

e) Technical Contraints

Constraint bzw. Einschränkung	**Beschreibung**
Java als Entwicklungssprache	Im Rahmen des Kurses wird Java als Programmiersprache empfohlen. Weiterhin wird technischer Support nur bei dieser Sprache angeboten. Die UI wird mittels JavaFX realisiert.
Geringe Hardwareanforderung	Die Applikation sollte auf jedem Handelsüblichen Notebook funktionieren. Z.B. Macbook Air mit Intel i3 Prozessor. Die Steuerung erfolgt über die Computertastatur.
Plattformunabhängig	Das Programm soll alle gängigen Betriebssysteme unterstützen (Bsp. Windows/ Mac)

f) Organizational Contraints

Constraint bzw. Einschränkung	**Beschreibung**
Entwicklerteam	Das Team besteht nur aus Alexander Könemann und Philipp Borchert. Der Arbeitsaufwand soll fair verteilt werden und beide Teilnehmer sollen durch gegenseitigen Austausch Know-how austauschen. Dies kann durch Feedback in den regelmäßigen Meetings oder Reviews erfolgen.
Zeitplanung	Der 27.01.2022 stellt die letzte Möglichkeit der erfolgreichen Übergabe des Projektes an den Kunden dar. Die Review Termine sind generell optional, werden jedoch alle wahrgenommen.
Vorgehensmodell	Die Entwicklung erfolgt nach einem iterativen und inkrementellen Vorgehensmodell. Für jeden Meilenstein werden Inkremente geplant und

	umgesetzt. Zur Dokumentation der Architektur wird arc42 verwendet.

5) Kontextabgrenzung

g) Business Kontext

Zur Veranschaulichung der möglichen Nutzungsszenarien durch den User wurden Use-Case-Diagramme erstellt:

UC0 – Tron Spiel

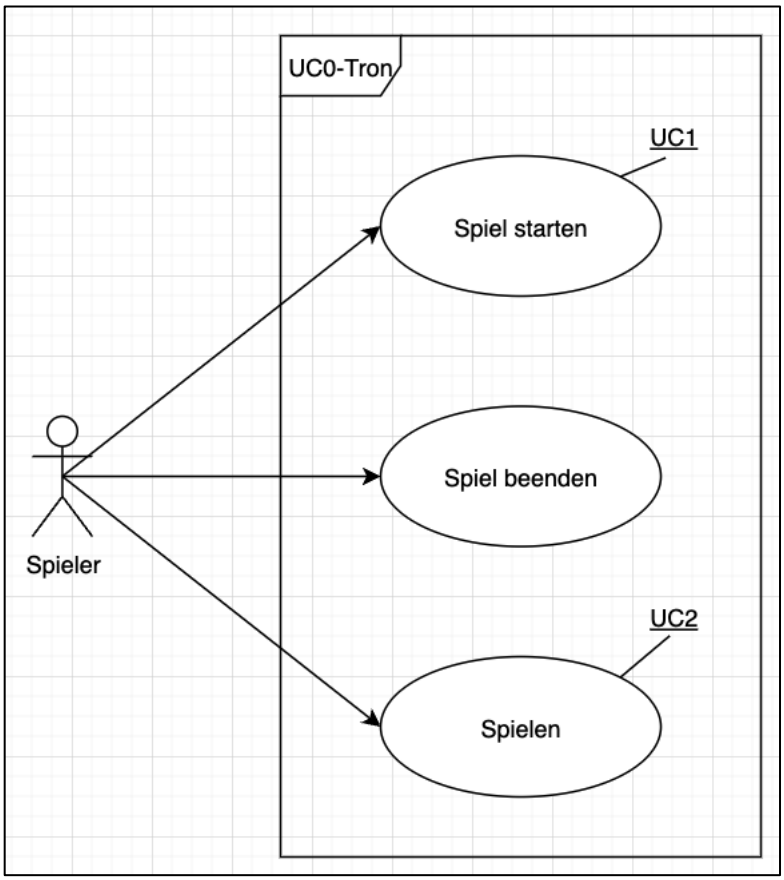


Figure 5: Veranschaulichung der Anwendungsfälle durch ein Use-Case-Diagramm (eigene Darstellung)

UC1 - Spiel starten: 2 Optionen

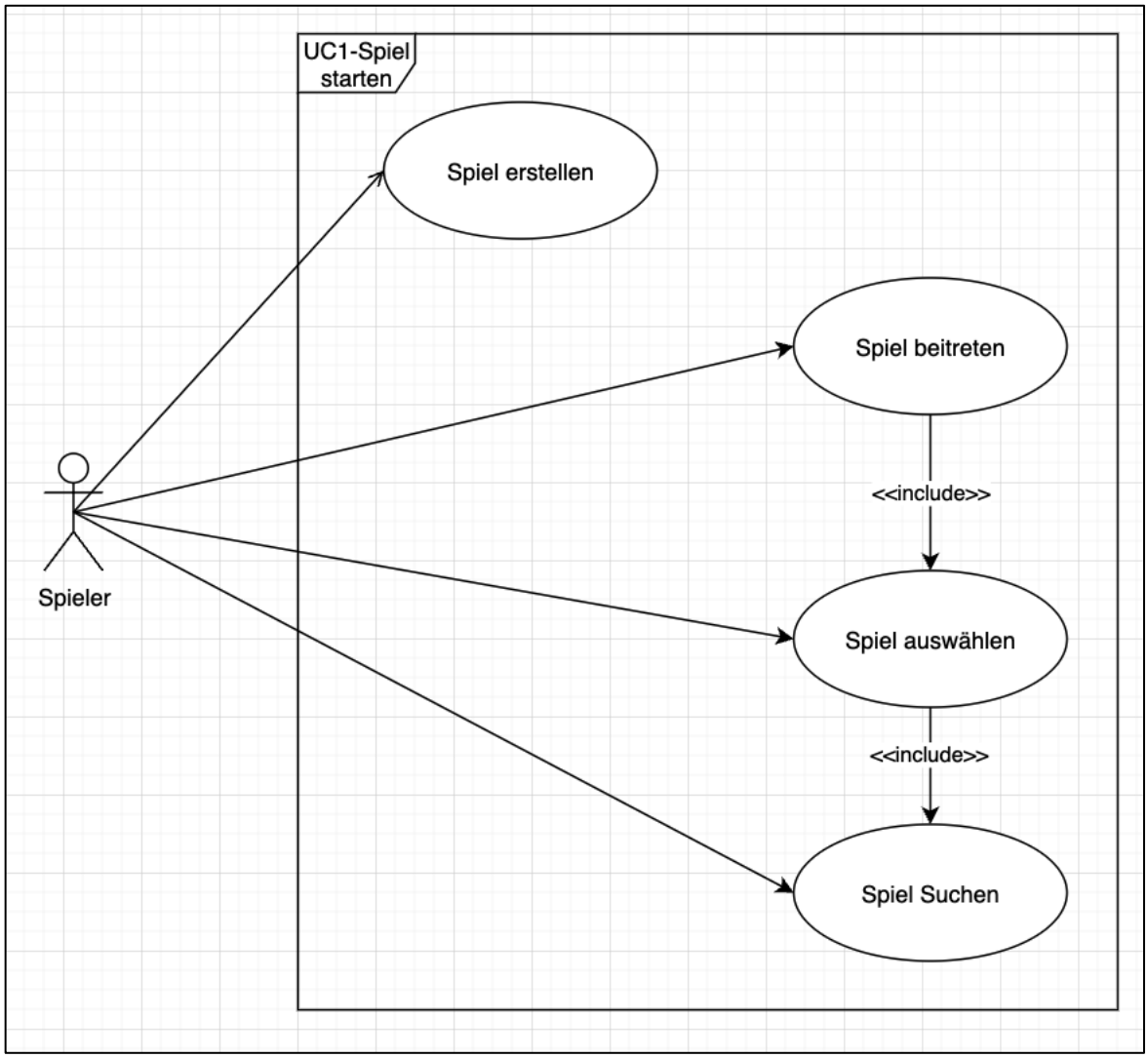


Figure 6: UC1 (eigene Darstellung)

UC2 - Spielen

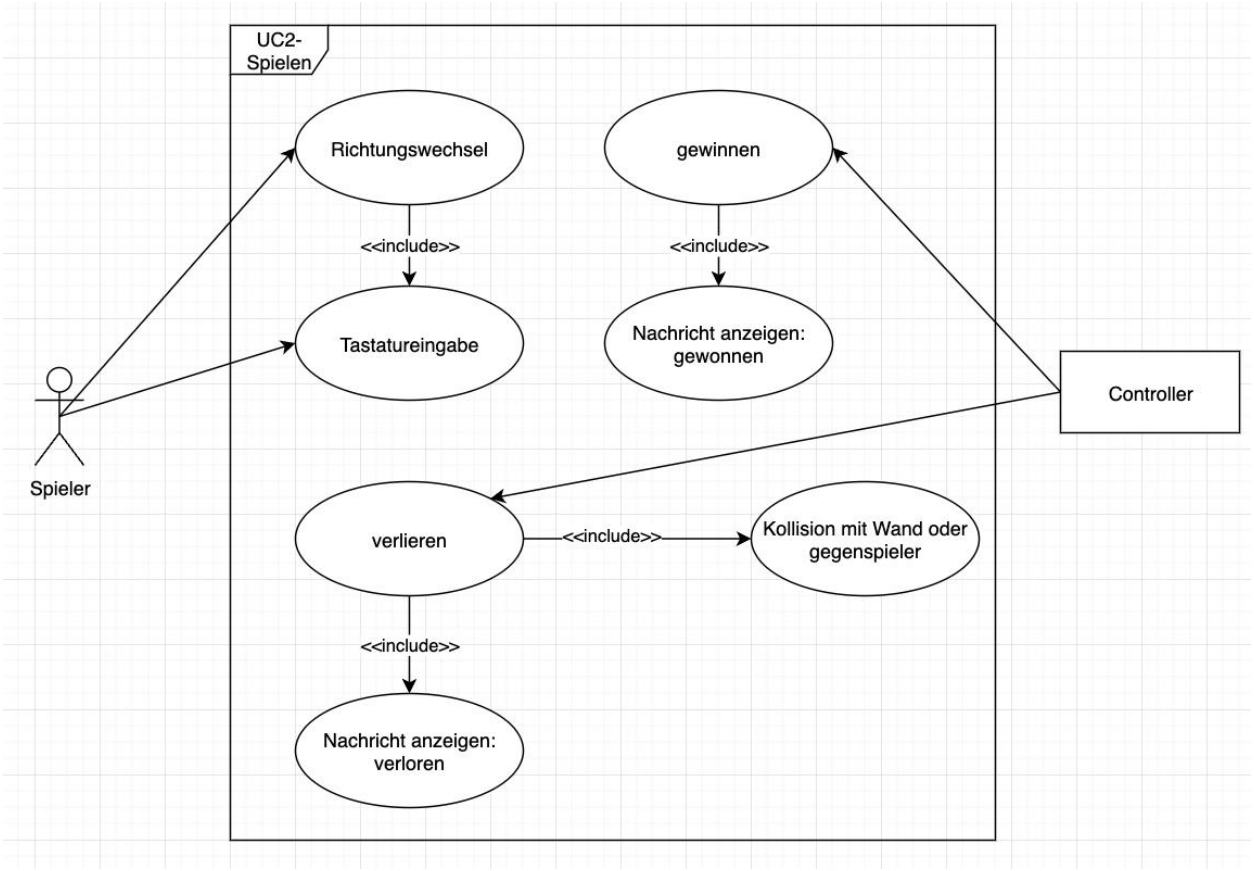


Figure 7: UC2 (eigene Darstellung)

AD0-Tron (UC0|AD0): Aktivitätsdiagramm App starten/beenden

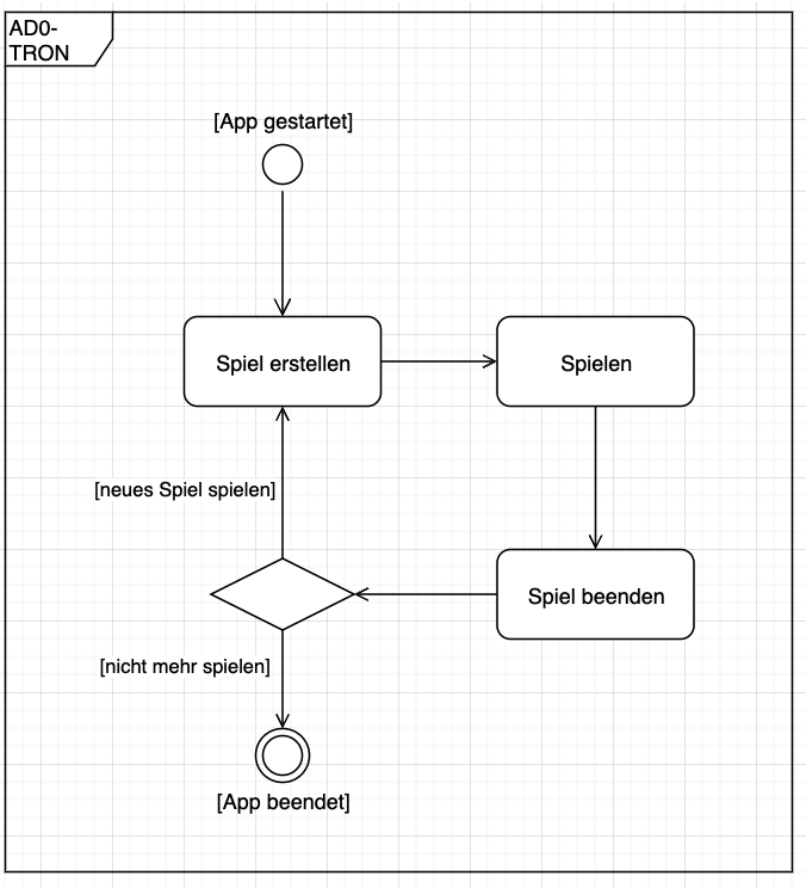


Figure 8: AD0 (eigene Darstellung)

AD1 – ein Spiel starten/erstellen/beitreten (UC1|AD1)

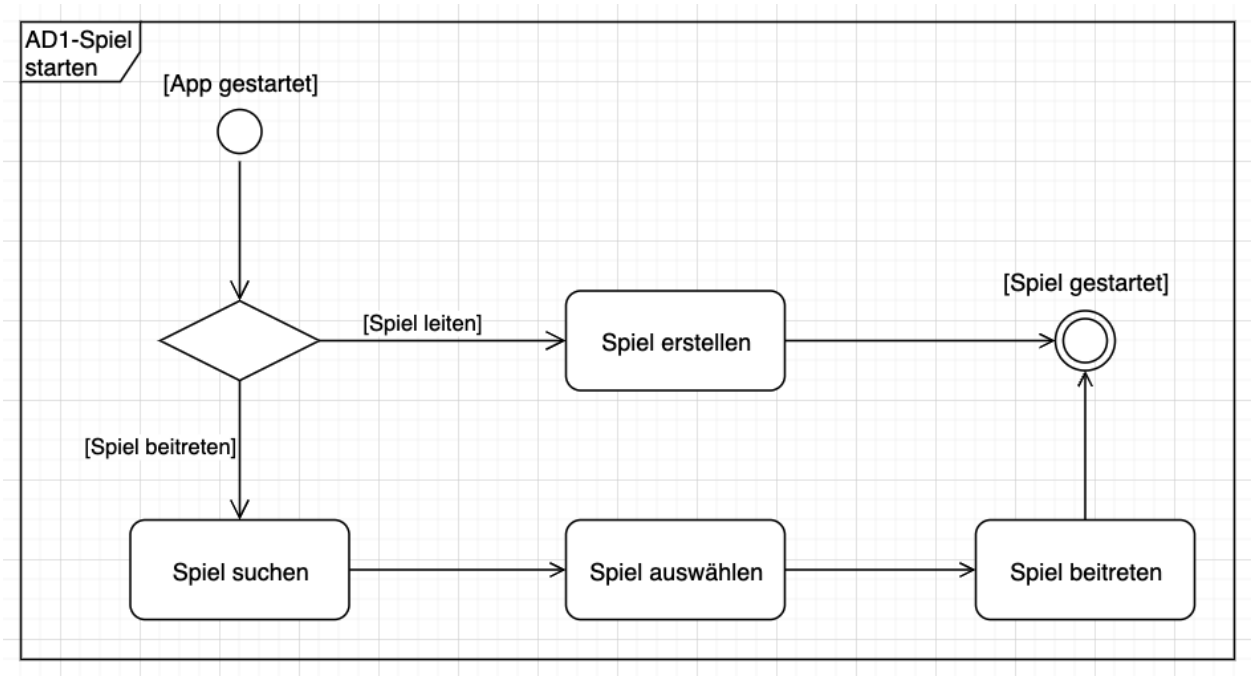


Figure 9: AD1 (eigene Darstellung)

AD2 – ein Spiel spielen (UC2|AD2)

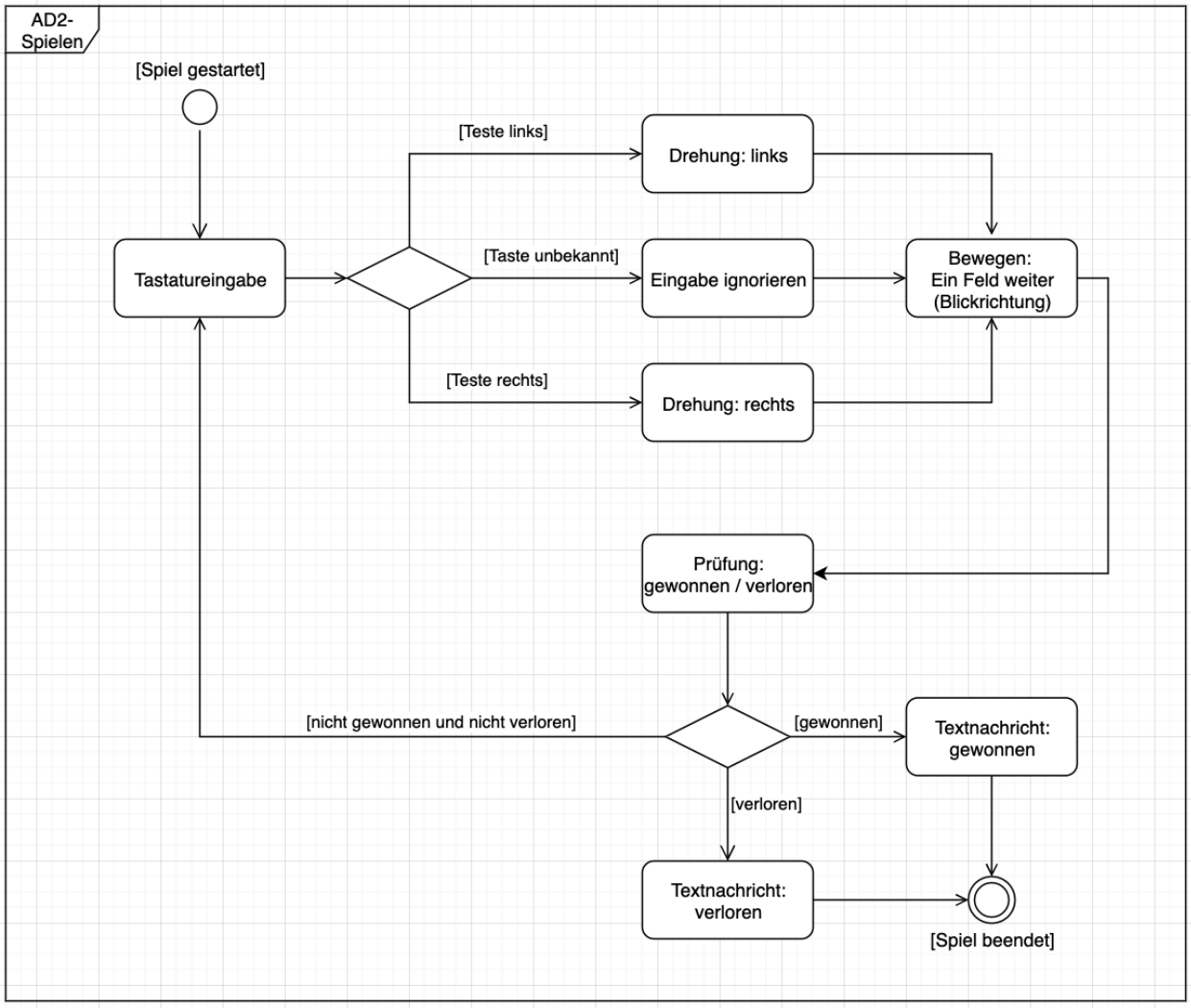


Figure 10: AD2 Spiel starten (eigene Darstellung)

Der zeitliche **Ablauf eines Spieles** entsprechend AD2 bzw. Use Case U3 aus **Error! Reference source not found.** wird mit dem folgendem Sequenzdiagramm veranschaulicht:

F0 (UC0|AD0|F0) Userflow App Nutzung mit Flowchart Diagramm:

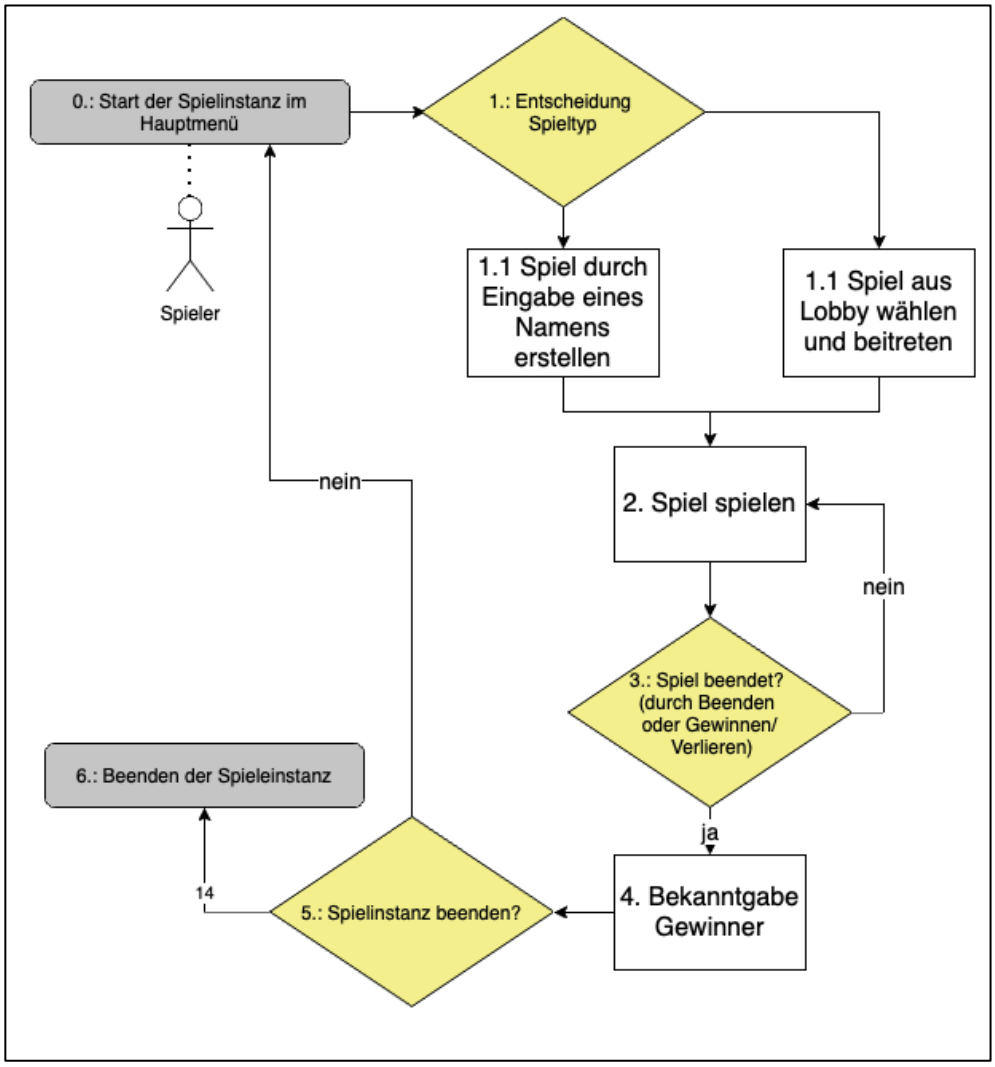


Figure 11: Der Spieleprozess dargestellt als Flowchart Diagramm (eigene Darstellung)

h) Technical Kontext

Übersicht der Systembestandteile des Systems[2]:

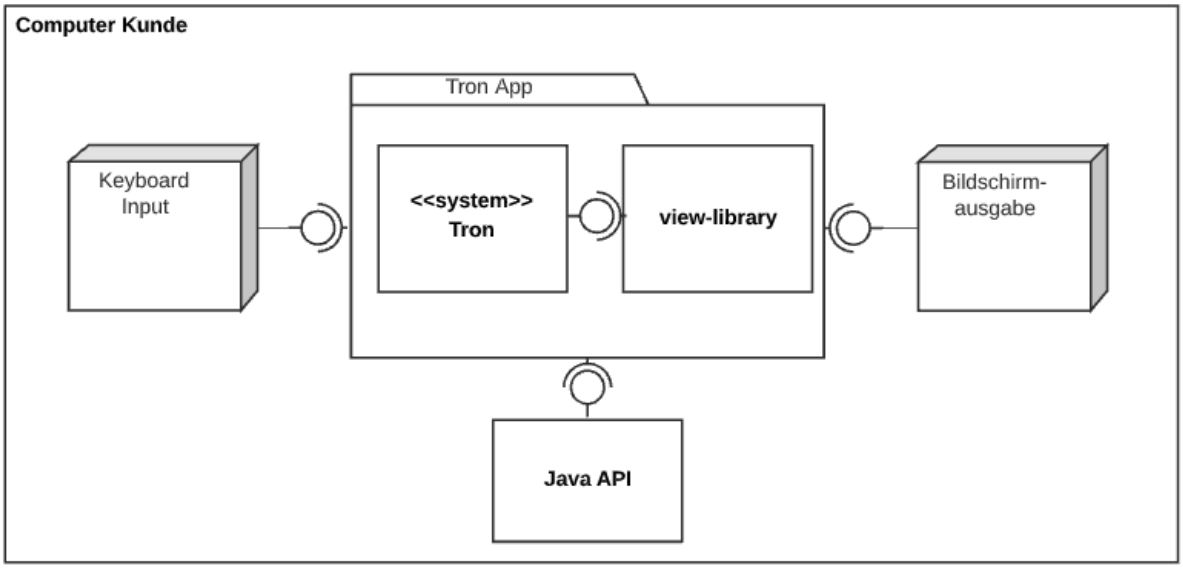


Figure 12: Übersicht Systembestandteile (eigene Darstellung)

[2] *https://lucid.app/lucidchart/07620ce8-1c7d-4ec0-9a34-277bd4b69adb/edit?viewport_loc=492%2C-204%2C3440%2C2269%2CHWEp-vi-RSFO&invitationId=inv_b6f3db2d-4edd-4ce2-99ed-078feb203df4*

SD0 (UC2|AD2|SD0) – Ablaufsemantik Spielsequenz

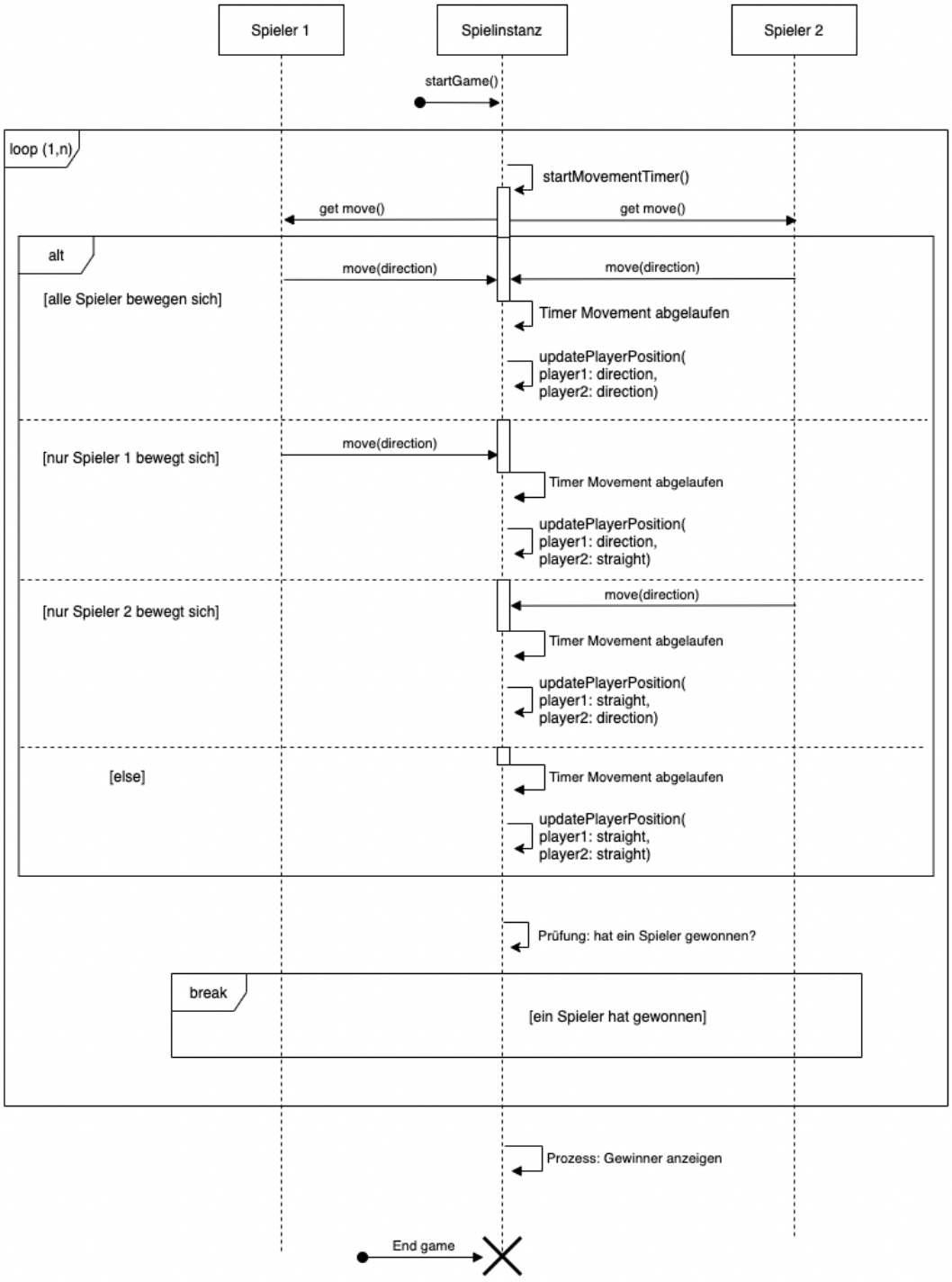


Figure 13: Ablauf der zeitlichen Abfolge eines Spieles dargestellt im Sequenzdiagramm (eigene Darstellung)

6) Lösungsstrategie

i) Projektplan[3] zur Erarbeitung der Inkremente:

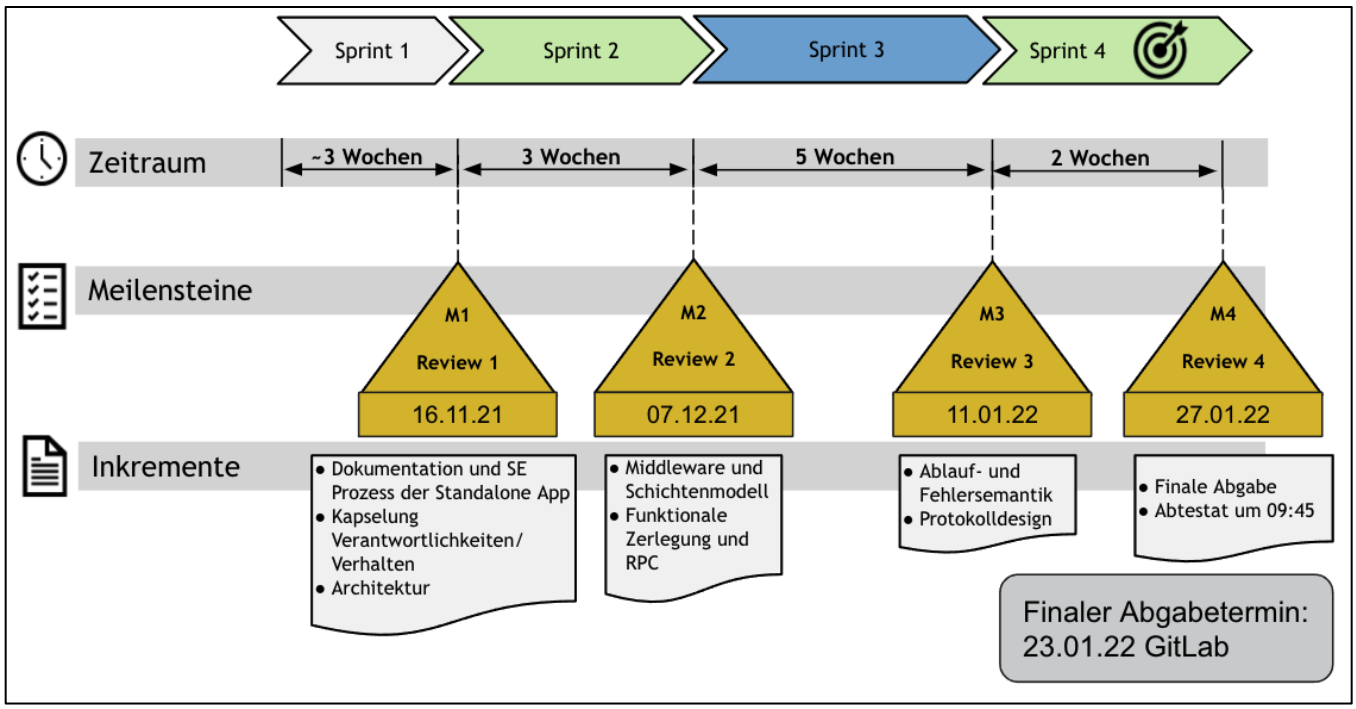


Figure 14: Projektplan zur Erreichung des Projektzieles (eigene Darstellung)

j) Projektmanagementmethodik:

- Entwicklung nach adaptierter SCRUM Methodik[4].
- Aufteilung des Projektes in 4 Sprints zur Generierung festgelegter Inkremente
- Festlegung der Inkremente durch Entwicklerteam in Absprache mit dem Kunden
- Nach jedem Sprint wird das Inkrement zusammen mit dem Kunden reviewed. Evtl. nötige Änderungen werden dann im nächsten Sprint erledigt
- Daily Meetings werden durch einen wöchentlichen Jour Fixe (dienstags) erfüllt. Inhalt: Diskussion Zwischenergebnisse, Änderungen und Probleme
- Organisation des Backlogs, Sprint Backlogs, WIP und abgeschlossene Themen mit Trello Kanban Board.

[3] Projektplan einsehbar unter: *https://docs.google.com/presentation/d/1BIZMIhBbEzaA7x2pHdnqUC6iriMMlvO233gPfCFWmNc/edit?usp=sharing*

[4] Entsprechend Scrum Guide nach Schwaber, Sutherland (2020). Verfügbar unter: *https://scrumguides.org/index.html*, abgerufen am 15.12.2021.

Beschreibung Fairness

Das Spiel wird intervallweise aktualisiert. Somit hat jeder Spieler dieselben Eingabevoraussetzungen. Die Länge eines Intervalls wird nach Fertigstellung des Prototyps ermittelt bzw. festgelegt.

k) Funktionsbeschreibung:

Funktion	Methodensignatur[5]	Rolle	Use Case	Vorbedingung	Nachbedingung
Spiel erstellen	hostGame(gameName, localGame)	Spieler	UC0, UC1	App installiert, Kein Spiel gestartet	Spiel erstellt, Spiel gehostet
Spiel beitreten	joinGame(gameName)	Spieler, gameMenu	UC1	Spiel gesucht und ausgewählt	Spiel beigetreten und gestartet
Spiel suchen	requestGameList()	Spieler, GameMenu		Gehostete Spiele existieren	Liste mit Spielen wird angezeigt
Spiel beitreten	joinGame(hostIp, hostPort)	Spieler, GameMenu		Spiel aus Liste ausgewählt	Host wird informiert, dass spiel zu starten
Spiel spielen	gameLoopService.start()	Spieler, ApplicationStubs	UC0, UC2	Spiel gestartet, 2 Spieler vorhanden	Spiel beendet
Spiel beenden	sendPlayerLost(playerLeftLost, playerRightLost)	Spieler	UC0	Spiel gestartet	Spiel beendet
Richtung wechseln	onKeyPressed(keyEvent):MovingDirection	Spieler	UC2	Tastatureingabe	Motorrad ändert Richtung
Spieler bewegen	movePlayer(PlayerPositionOnGrid, MovingDirection): List<Coordinate>	Spieler	UC2, AD2	Keine Tastatureingabe	Motorrad bewegt sich in gleicher Richtung weiter
Spiel gewinnen	checkCollisionWithGrid(Coordinate):false, checkCollisionWithPlayer(List<Coordinate>, List<Coordinate>): true	GameController, UserInterface	UC2	Bewegung des Spielers, Spiel gestartet	Gewinner und Verlierer angekündigt
Kollision mit Wand	checkCollisionWithGrid(Coordinate):true	GameController	UC2	Spieler befindet sich eine Zelle von der Wand entfernt und bewegt sich in Richtung Wand	Spieler verliert Spiel, Gegenspieler gewinnt Spiel
Kollision mit Gegenspieler	checkCollisionWithPlayer(List<Coordinate>, List<Coordinate>): true	GameController	UC2	Spieler befindet sich eine Zelle vom Gegenspieler entfernt und bewegt sich in Richtung Gegenspieler	Spieler verliert Spiel, Gegenspieler gewinnt Spiel
Gewinner anzeigen	showWinner(winner): Overlay	View	UC2	Gewinner und Spieler werden übermittelt	Gewinner und Spieler werden angezeigt
Spiel-veränderung loggen	Logger.info(String)	View	Logger	Ereignis ist eingetreten	Relevante Information geloggt

[5] Entsprechen der Signatur in der Komponentensicht der Applikation: *https://lucid.app/lucidchart/d10df9dd-49d5-46b0-9831-c8c097554942/edit?viewport_loc=693%2C48%2C2219%2C1114%2CHWEp-vi-RSFO&invitationId=inv_072965e4-3dca-4759-9b98-50b25fea16b5*

7) Softwarearchitektur

I) Gewählte Softwarearchitektur Option B: 3 Schichtenarchitektur[6]

Als **Softwarearchitektur Option B** wird eine Entwicklung in Anlehnung an das **3-Schichten-Modell** verwendet. Diese hat den Vorteil, dass die GameView, die später auf verschiedene Applikationen verteilt wird, nur mit einer Komponente gekoppelt ist. Die verworfene Architekturoption A nach dem MVC Pattern sowie eine Begründung für die Entscheidung findet sich im Anhang.

Darstellung der Softwarearchitektur:

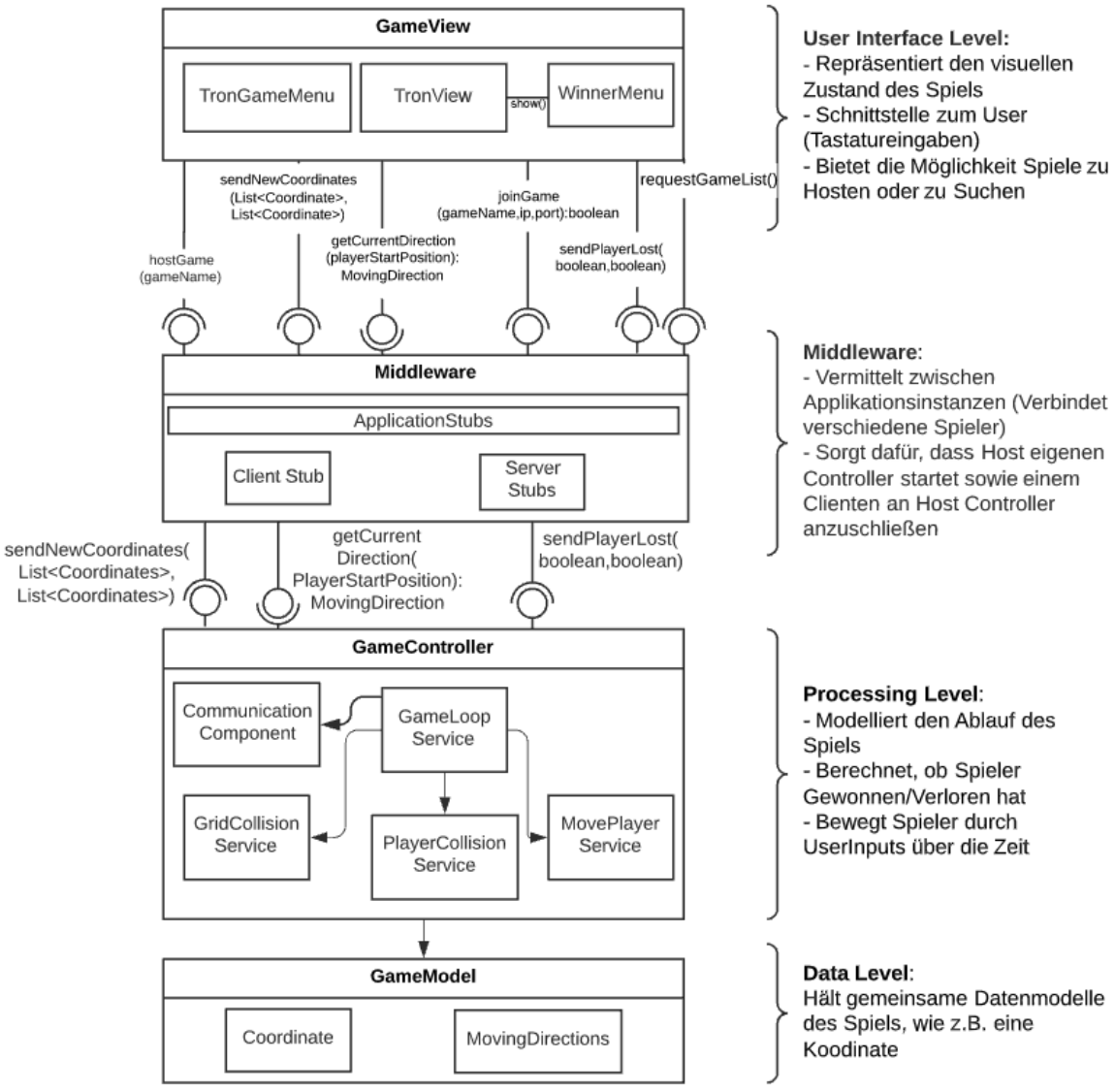


Figure 15: Die Applikation wird in einer 3-Schichten-Architektur entwickelt (eigene Darstellung)

[6] Diagramm einsehbar unter: *https://lucid.app/lucidchart/d10df9dd-49d5-46b0-9831-c8c097554942/edit?invitationId=inv_072965e4-3dca-4759-9b98-50b25fea16b5*

8) Verteilungssicht/ Design Middleware

m) Transparenzziele Kunde:

Kriterium	Beschreibung	Kundenanforderung	Umsetzung
Access	Hide differences in data representation and how an object is accessed.	Der Kunde bemerkt keinen Unterschied, ob die Verarbeitung seiner Eingaben auf seinem lokalen Rechner oder entfernt umgesetzt werden.	Mittels RPC können lokale Methoden aufgerufen werden, die auf einer entfernten implementiert sein könnten. Je nachdem, ob der Spieler Host ist oder einem Spiel joined. Dies wird jedoch vor dem User versteckt.
Location	Hide where an object is located.	Der Kunde hat keine Kenntnis darüber, wo das Spiel ausgeführt wird (GameController).	Jeder Spieler startet seine eigene GUI. Der Host startet zusätzlich einen GameController, zu dem sich der Mitspieler verbindet. Die Verbindung erfolgt über die Suche in der Spielelobby im Spielmenü.
Relocation	Hide that an object may be moved to another location while in use.	Nicht erforderlich	Nicht erforderlich
Migration	Hide that an object my move to another location.	Nicht erforderlich	Nicht erforderlich
Replication	Hide that an object is replicated.	Keine besonderen Anforderungen	Nicht erforderlich
Concurrency	Hide that an object may be shared by several independent users.	Spiele sollen sich nicht beeinflussen. Das bedeutet, die Performance eines Spieles wird nicht durch andere Spiele gestört sowie die Ergebnisse der anderen Spiele nicht angezeigt.	Jedes Spielerpaar agiert autonom (horizontale Skalierung). Der Spieler, der ein Spiel hostet, übernimmt den Rechenaufwand, den Spielverlauf zu berechnen.
Failure	Hide the failure and recovery of an object.	Kleinere Fehler < 1 sek sind gegenüber dem Kunden zu kompensieren und nicht anzuzeigen. Größere Fehler führen zu einem nicht gewerteten Spiel und zumindest einen Hinweis für den Entwickler	Konsolenausgabe von größeren Fehlern. Sollte ein Spielteilnehmer keine Tastenanschläge übermitteln, fährt der Spieler in der gleichen Richtung weiter.

n) Anforderungen Skalierung

Gemeinsame Nutzung von Ressourcen: Maximale Auslastung der Rechner von 80%
Anforderung Skalierung:

- Geographische Skalierung: Loopback oder LAN
- Administrative Skalierung: 1 Administrator
- Die Skalierungsart für Problemgröße mindestens 32 Spiele parallel laufen zu lassen kann frei gewählt werden (horizontale oder vertikale Skalierung)
- 80% Auslastung der CPU

o) Skalierung der Problemgröße

- Für die Skalierung der Applikation gilt es die Anzahl der parallel ausgeführten Spiele zu maximieren bzw. zu verteilen, ohne dabei die Performance einer Instanz zu verringern
- Der Ansatz zur Skalierung folgt dem Ziel, die Spiele auf verschiedenen Computer/Nodes zu verteilen (horizontale Skalierung) indem jedes Spielerpaar einen gemeinsamen GameController inkl. GameModel teilt. Jeder Spieler startet sein eigenes User-Interface, jedoch startet nur der Host eines Spieles den GameController mitsamt GameModel. Der einzige Mitspieler (Client) verbindet sich über Lan oder Loopback zu dem Spiel des Hosts. Die Datenübertragung wird über RPC realisiert.

- 2 Nodes/Computer umfassen 2 GUIS und 1 Instanz des Spieles. **Skalierungsschema**:

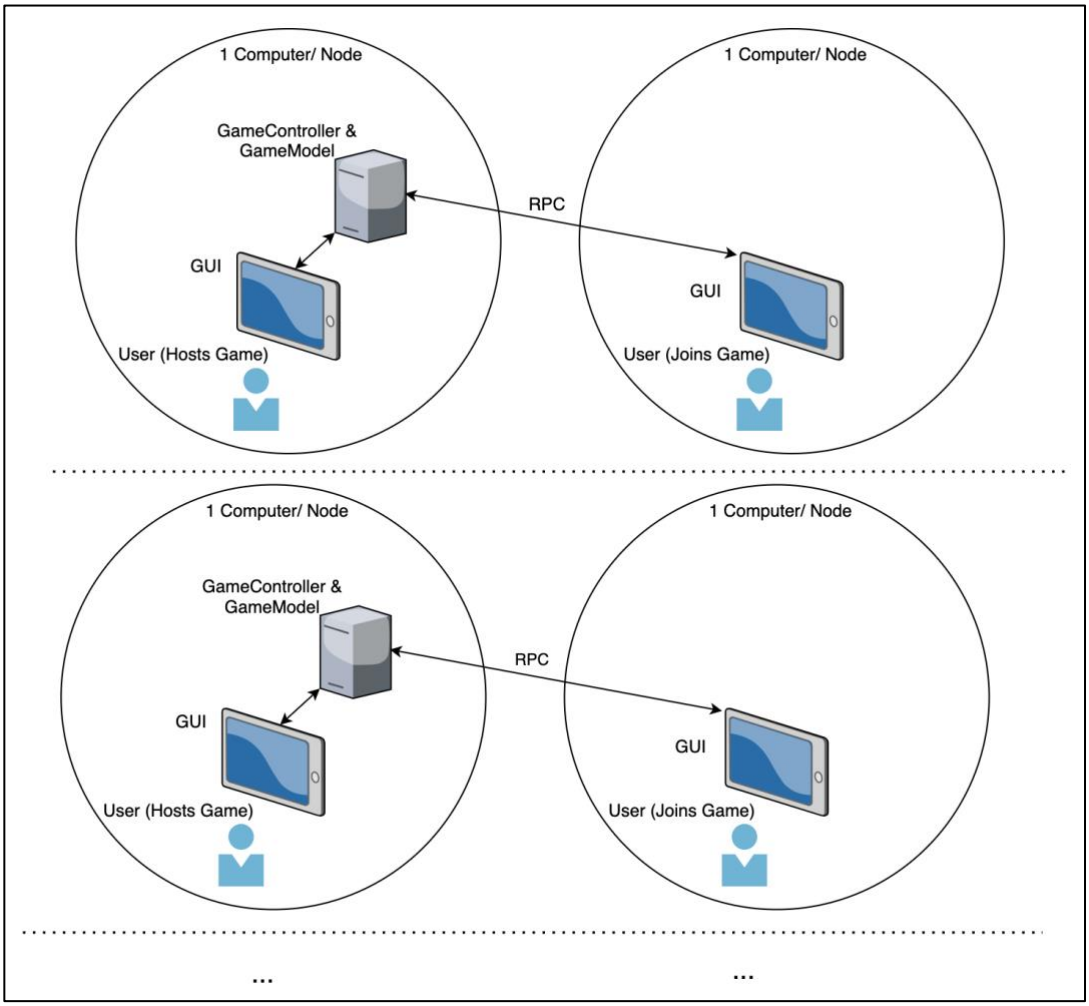


Figure 16: Skalierungsschema der App: Horizontale Skalierung durch Peer-to-Peer Ansatz (eigene Darstellung).

p) Entwurf RPC-Schnittstelle

- Als Referenzliteratur für den Entwurf der Middleware bzw. RPC Schnittstelle wird der Ansatz nach Tanenbaum verwendet[7].
- RPC übersetzt den Funktionsaufruf in eine Nachricht und suggeriert einen lokalen Funktionsaufruf, der ggf. auf einer entfernten Node ausgeführt wird.
- Es gibt **zwei wesentliche Kommunikationsarten:**
    - Zum Suchen/Beitreten von Spielen oder zum Hosten von Spielen: Mittels **Broadcast** wird das **UDP Protokoll** verwendet, die Antwort per TCP. 1 zu N Verbindung je nach Anzahl der Spiele und Spieler in der Spielelobby.
    - Beim Spiel besteht eine 1:1 TCP-Verbindung zwischen zwei Instanzen (nicht blockierend). Eine Instanz fungiert als Host, die ein Spiel anbietet und ein Client, der dem Spiel beitritt. Es gilt das first-come first-served Prinzip: Es kann sich nur ein Client zum Host verbinden und der Erste wird angenommen.
- Durch Marshalling werden die Funktionsparameter in eine Nachricht verpackt. Bei TCP werden die Daten (z.B. Liste mit Koordinaten als Integer) in einen TCP Bytestrom übersetzt und müssen vom Empfänger entpackt werden. Bei UDP werden die Daten in einem Datagramm verwendet.
- **Inhalt der Kommunikation** (spezifiziert in Enum Klasse RPCMethodName.java):
    - **Kommunikation zwischen Spielinstanzen vor Spielstart** (Game Host mit Game Client) Struktur: <enum>:<MethodenName>:<Rückgabewert>:
        - REQUESTGAMELIST: requestgameList(): void.
        - ADDGAMETOGAMELIST: addGameToGameList(): void
        - JOINGAME: joinGame(): void
        - SETCONNECTIONTOHOST: setConnectionToHost(): void
        - STARTGAME: startGame(): void
    - **Kommunikation** zwischen **Game Host** und **Game Client während des Spiels**:
        - REQUESTCURRENTDIRECTION: requestCurrentDirection(): void
        - SENDCURRENTDIRECTION: sendCurrentDirection(PlayerPositionOnGrid, MovingDirection): void
        - SENDCURRENTCOORDINATES: sendCurrentCoordinates(Coordinate, Coordinate): void
        - SENDPLAYERLOST: sendPlayerLost(boolean, boolean): void

[7] Tanenbaum (2020), S. 173ff.

- Ablauf der Kommunikation: RPC Kommunikationspattern
- Design Middleware:
    - Siehe Bausteinsicht Middleware
    - In der Middleware wird keine register Methode realisiert, da diese bei dem Ansatz der horizontalen Skalierung nicht benötigt wird: Es gibt immer nur zwei Kommunikationspartner. Hier wird von dem Standard in der Referenzliteratur nach Tanenbaum abgewichen.
- Kommunikationsart:
    - Es wird eine transiente Kommunikation zwischen Server und Client über Sockets realisiert. Das bedeutet, dass Nachrichten nicht persistent verschickt werden (es wird kein Kommunikationsserver eingesetzt), sondern die Gegenseite die Verantwortung über die Verarbeitung der Nachricht übernimmt.
    - Durch den Einsatz einer TCP-Verbindung ist eine reihenfolgengesicherte vollständige Übertragung gewährleistet, solange keiner der Kommunikationspartner abstürzt.
    - Das Protokoll wurde von Prof. Martin Becke im Praktikum reviewed. Es wurde jedoch kritisiert, dass durch TCP (möglicherweise) "Datenstau" entstehen kann, wenn Pakete langsam bestätigt werden. Dadurch entsteht eine anhaltende Verzögerung für den Remote Spieler. Daher haben wir uns entschieden das Push (PSH) flag zu setzen und damit den Nagle Algorithmus zu deaktivieren, um das Problem zu kompensieren.
    - Asynchrone Kommunikation durch Verzicht auf Antwort. Die Anfrage von neuen Koordinaten löst eine Aktualisierung aus: Der Client wird angestoßen eine neue Bewegung an den Host zu senden. Für diese Sendung hat er einen Gametick Zeit (deltaTime im GameLoop, z.B. 120ms. Wird in der game.properties Datei spezifiziert). Falls die Antwort nicht im Zeitfenster ankommt, wird ein default value für die Bewegung angenommen(MovingDirection.Straight). Der Spieler geht dann geradeaus. So kann das Spiel auch noch fortgesetzt werden, wenn eine Bewegung nicht übertragen wurde.
- Struktur der **Kommunikation TCP**:
    - Umfasst die **Send und Receive** Aspekte der Middleware
    - Jedes Paket hat eine feste Länge, welche in den ersten 4 Bytes des Packages als Integer kodiert ist (header) und Big-Endian formatiert ist. Der Body umfasst ein JSON Package, welches UTF-8 formatiert ist. Das gesamte Paket wird als Bytestrom versendet.

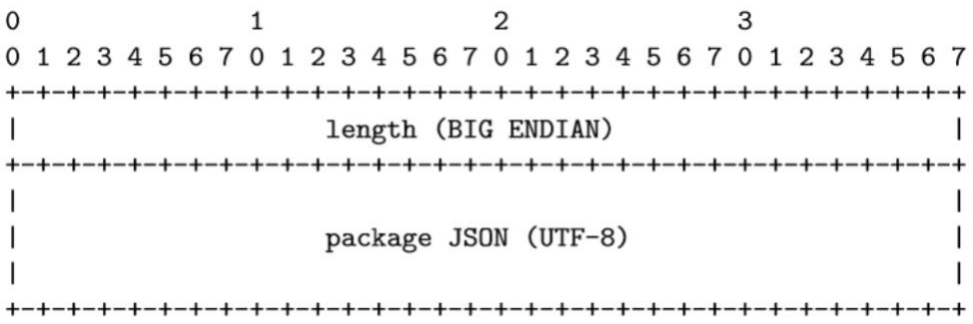


Figure 17: Paketstruktur mit Big Endian header und UTF-8 formatiertem JSON Package.

- Package-body Struktur:
  - Umfasst die **Marshalling und Unmarshalling** Aspekte der Middleware.
  - Generelle **JSON Struktur**:

    {

    „SOURCE_IP“: String,

    „SOURCE_PORT“: integer,

    „METHOD_NAME“: String,

    „PARAM“: Object, (Methodenspezifisch)

    „DISCONNECT“: boolean

    }
  - Default: Parameter werden entsprechend der JSON Spezifikation nach RFC4627[8] durch die eingesetzte Bibliothek JSON.simple[9] umgewandelt:
- Die Struktur des JSON Keys „PARAM“ wird abhängig übergebenen Typ der RPC Methode unterschiedlich serialisiert:
  1. Typ **null**:
     - „PARAM“: null.
     - Wird bei den Methoden requestgameList(), joinGame(), setConnectionToHost(), startGame(), requestCurrentDirection(). verwendet:
  2. Typ **String**:

[8] Siehe *https://www.ietf.org/rfc/rfc4627.txt*

[9] Siehe *https://github.com/fangyidong/json-simple*

- „PARAM“: String
- Wird bei den Methoden addGameToGameList(gameName), sendCurrentDirection(PlayerPositionOnGrid, MovingDirection) verwendet:

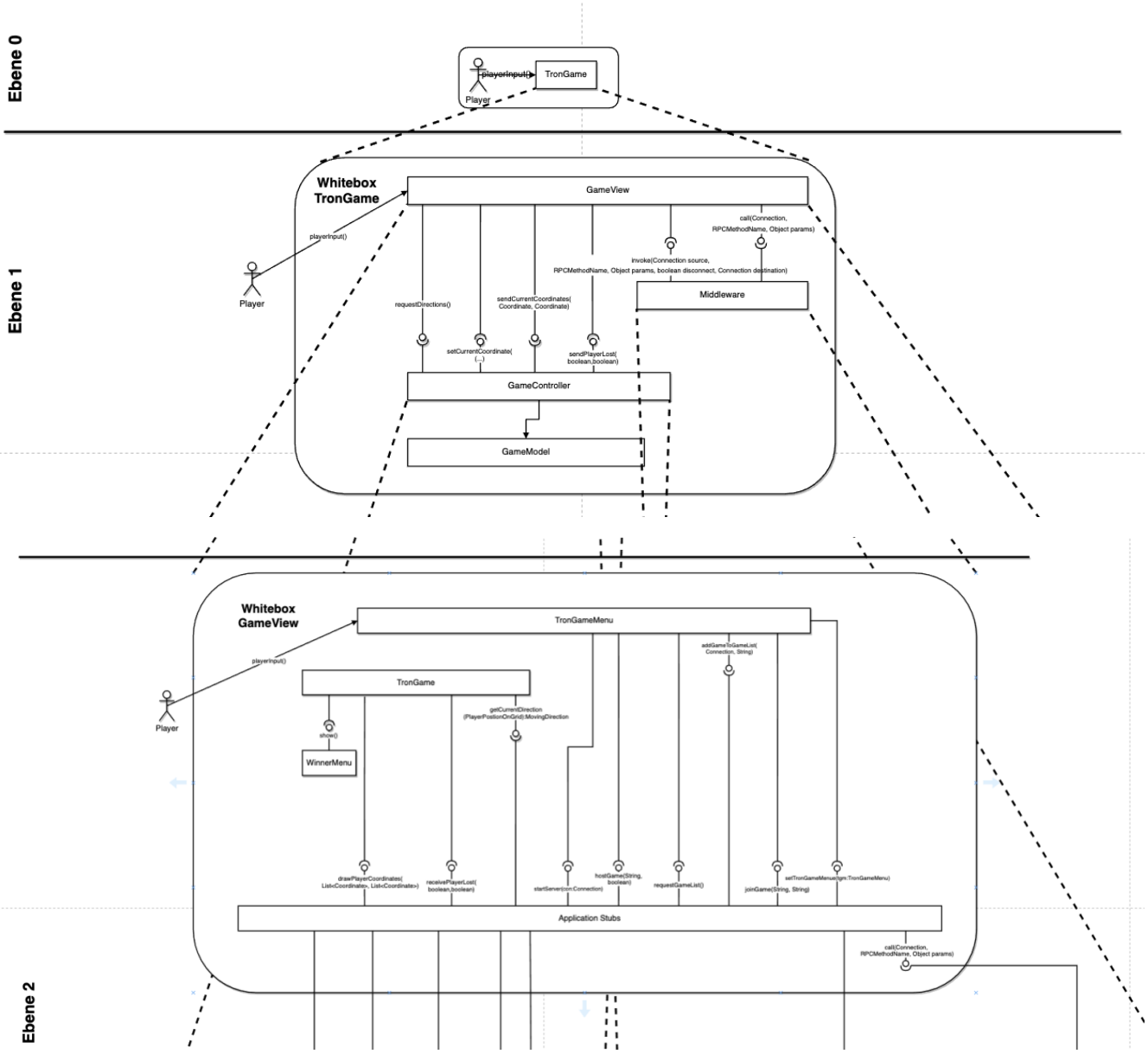
3. Typ **ArrayList<Coordinate>:**

   - „PARAM“: ArrayList<List<Integer>> wird übergeben.
     Diese beinhaltet zwei List<Integer>, die erste mit der aktuellen Koordinate für den linken Spieler und die Zweite für den rechten Spieler. In einer List<Integer> entspricht der erste Wert der X-Koordinate des entsprechenden Spielers und der zweite Wert der Y-Koordinate des Spielers.
   - Wird bei der Methode sendCurrentCoordinates(Coordinate, Coordinate) verwendet.

4. Typ: **ArrayList<Boolean>:**

   - „PARAM“: ArrayList<Boolean> wird übergeben. Es beinhaltet zwei Booleans, der Erste gibt an, ob der linke Spieler verloren hat und der Zweite, ob der rechte Spieler verloren hat.
   - Wird bei der Methode sendPlayerLost(boolean, boolean) verwendet.

9) Bausteinsicht



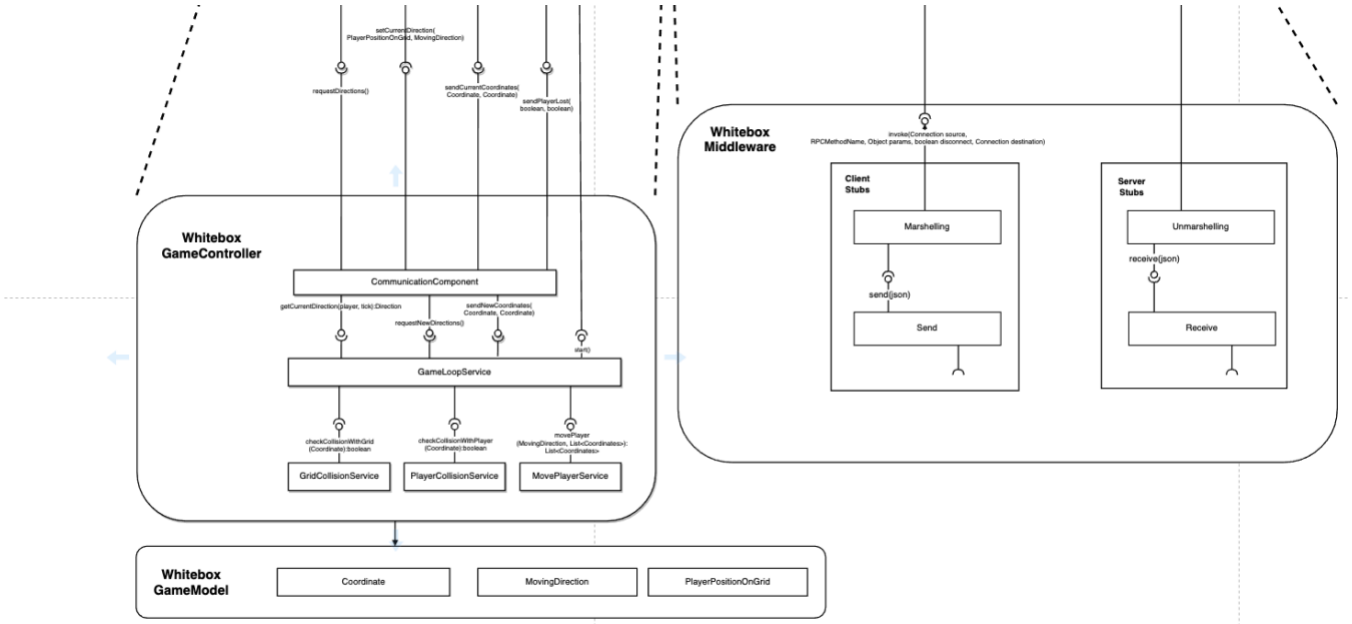


Figure 18: Bausteinsicht mit den Leveln 0,1 und 2 (eigene Darstellung).

10) Klassendiagramme

q) UserInterfaceLevel: GameView[10]

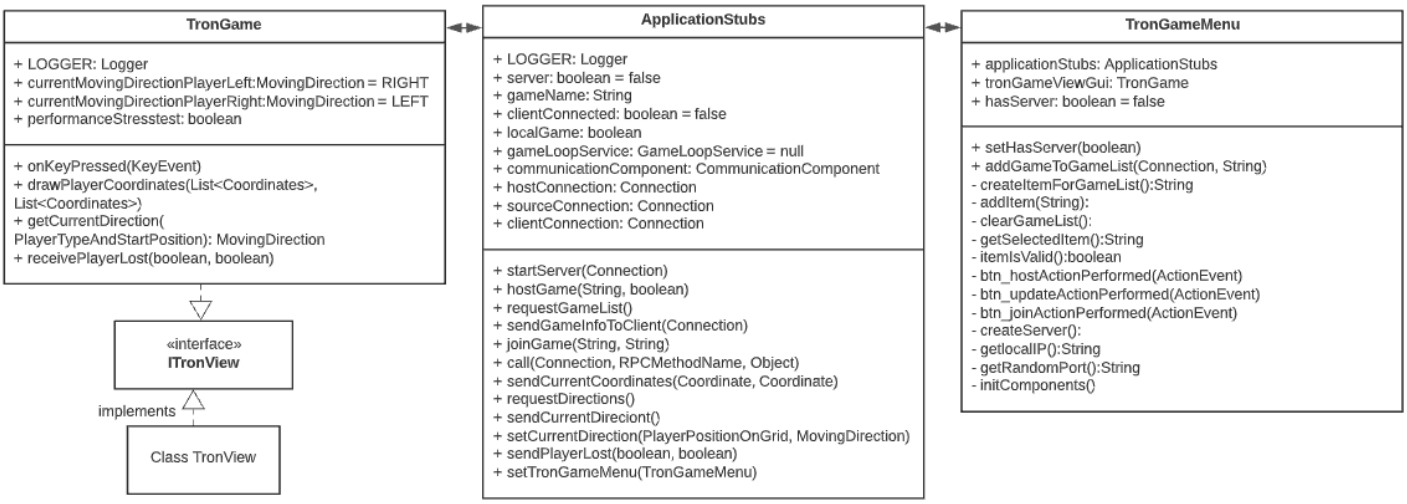


Figure 19: Klassendiagramme UserInterface Level (eigene Darstellung).

[10] https://lucid.app/lucidchart/d3754edd-24a5-4263-9275-b88f0e3fdd75/edit?invitationId=inv_aefd0125-25e2-44a9-8aac-12108e2d59f6

r) Processing Layer: GameController

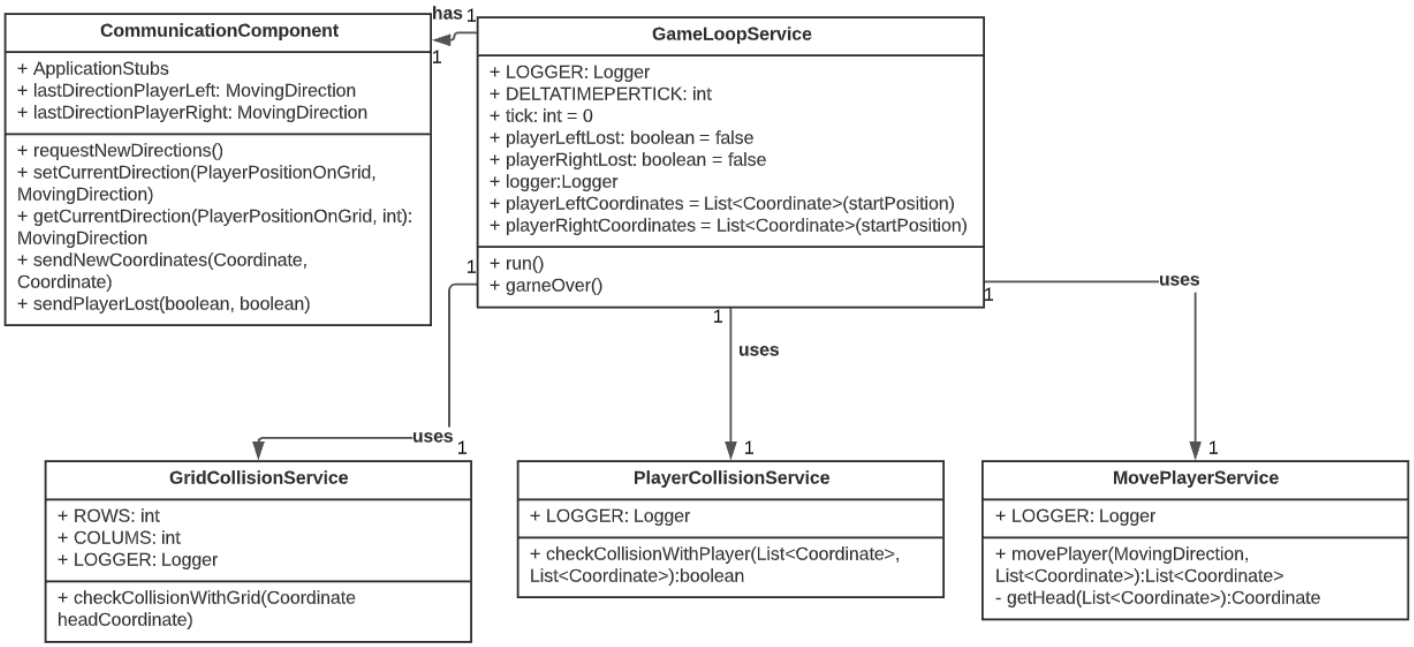


Figure 20: Klassendiagramme Processing Layer (eigene Darstellung).

s) DataLevel: MovingDirection, RPCMethodName, PlayerPositionOnGrid und Coordinate

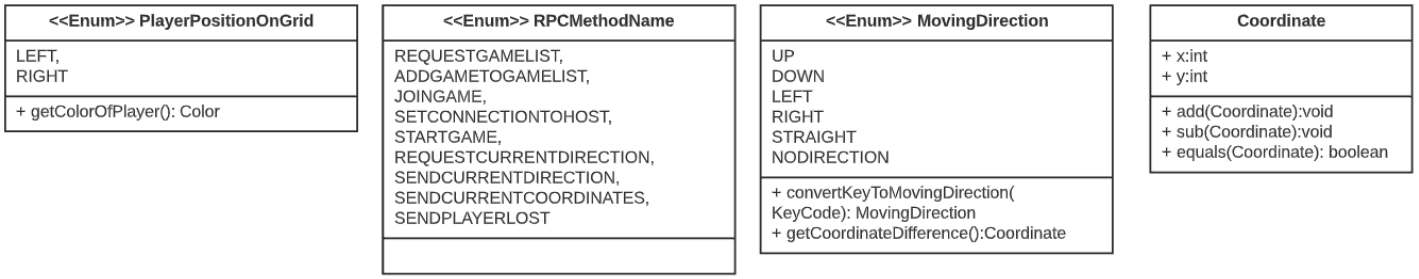


Figure 21: Klassendiagramme Data Level (eigene Darstellung).

11) Laufzeitsicht

t) GameLoopService: Ablaufsemantik eines Zuges

Input: MovingDirection beider Spieler

Output: List<Coordinate> beider Spieler, boolean playerLost (einer oder beide Spieler)

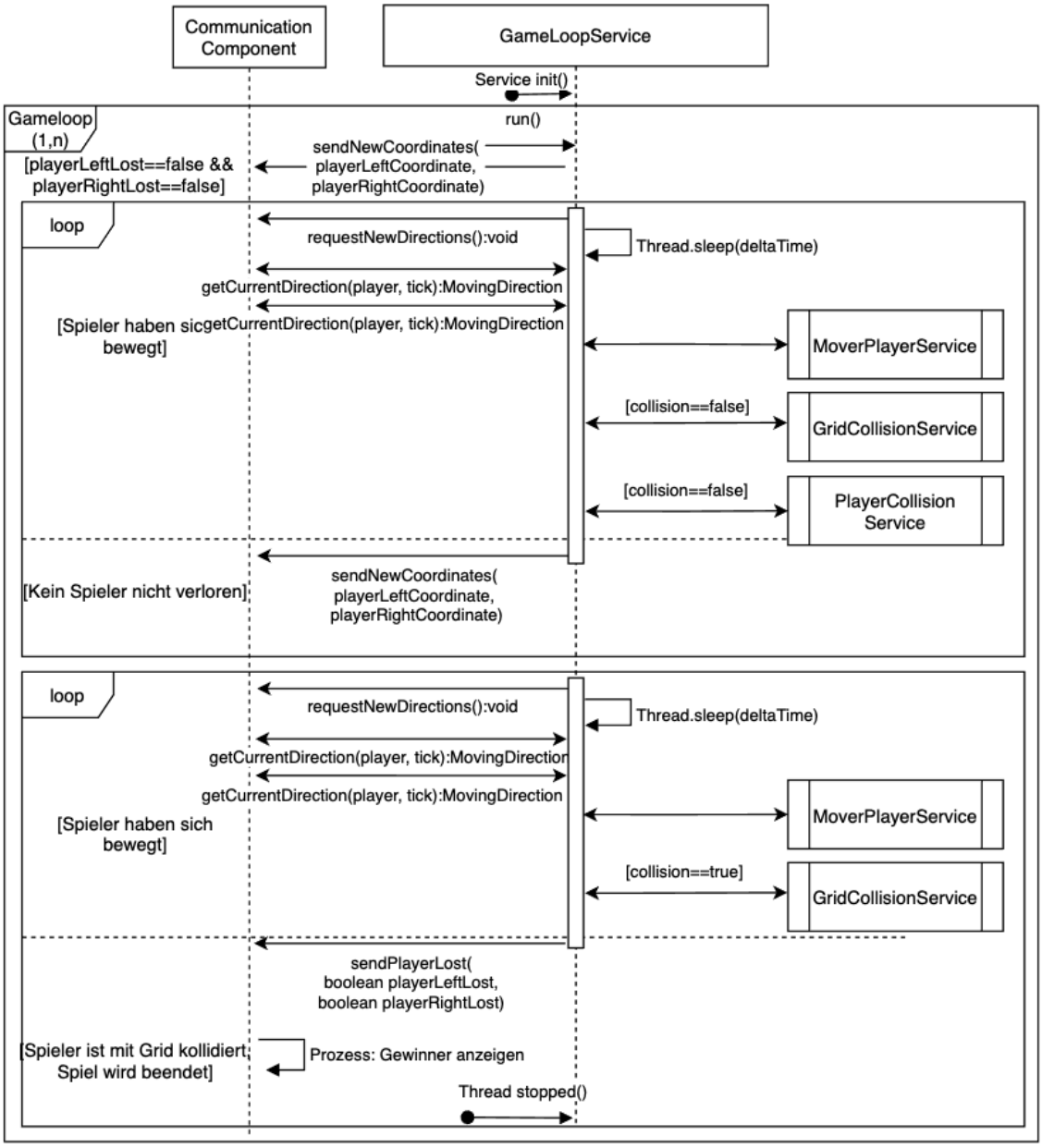


Figure 22: Ablaufsemantik eines Zuges als Sequenzdiagramm im GameLoopService (eigene Darstellung).

u) GridCollisionService: Kollissionspüfung mit Spielfeldrand

Input: Coordinate(x,y)

Output: Boolean (Player ist kollidiert/ nicht kollidiert)

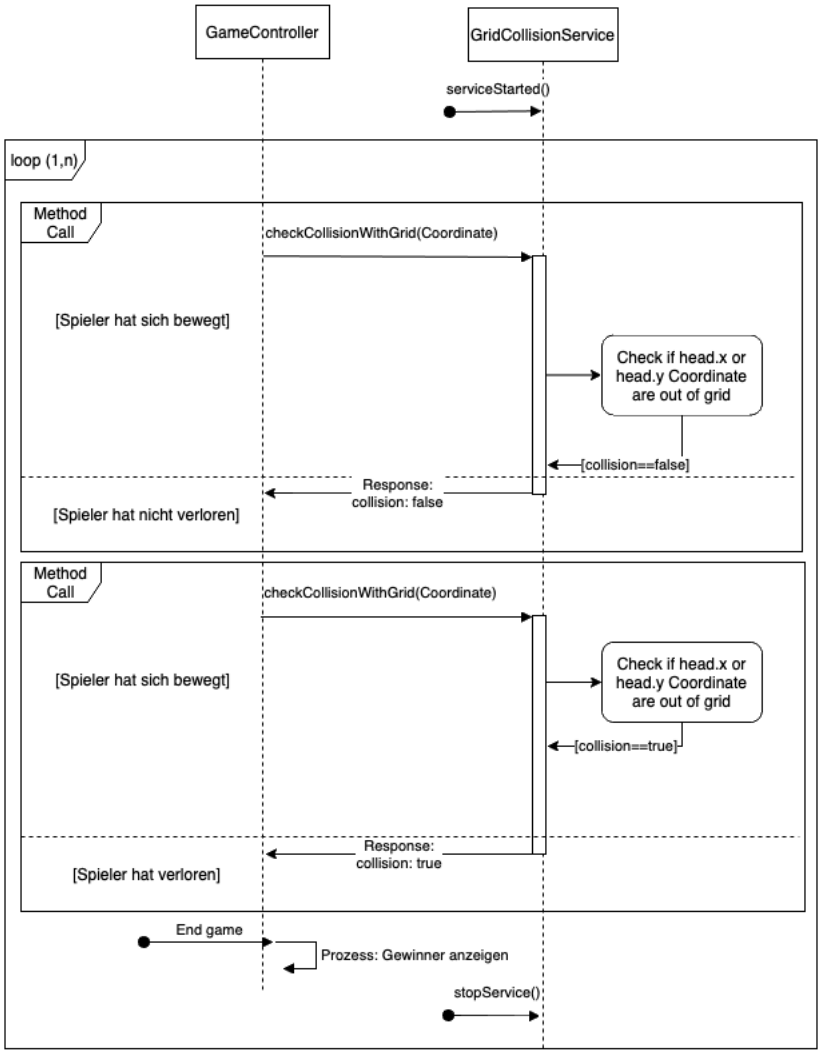


Figure 23: Kollisionsprüfung mit Spielfeldrand als Sequenzdiagramm im GridCollisionService (eigene Darstellung).

v) MovePlayerService: Ablaufsemantik Bewegung eines Spielers

Input: MovingDirection, List<Coordinate>

Output: List<Coordinate>

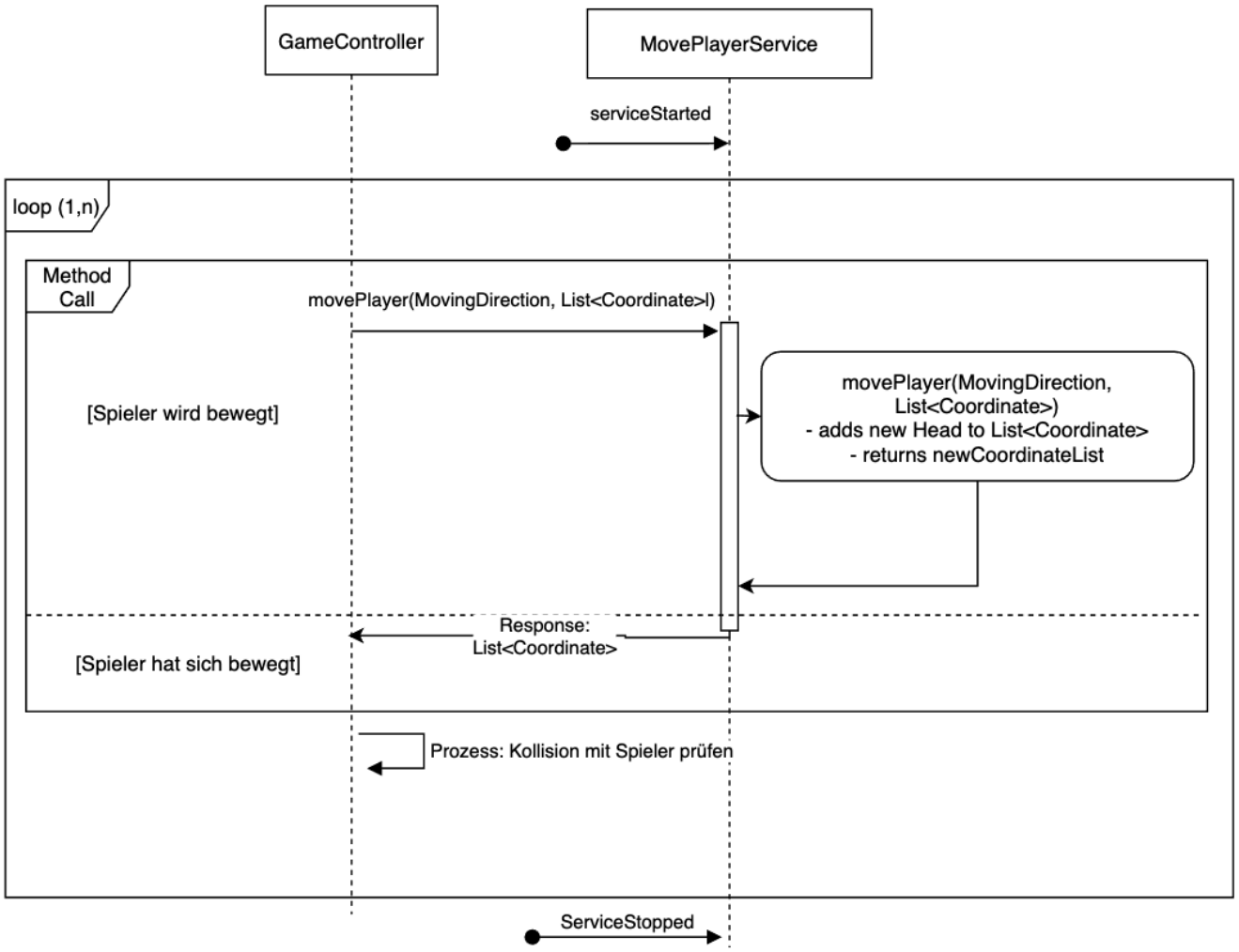


Figure 24: Ablaufsemantik der Bewegung eines Spielers als Sequenzdiagramm im MovePlayerService (eigene Darstellung).

w) Communication Component: Ablaufsemantik eines Gameloops

Input: PlayerPositionOnGrid, tick:Int, MovingDirection
Output: MovingDirection, Coordinate

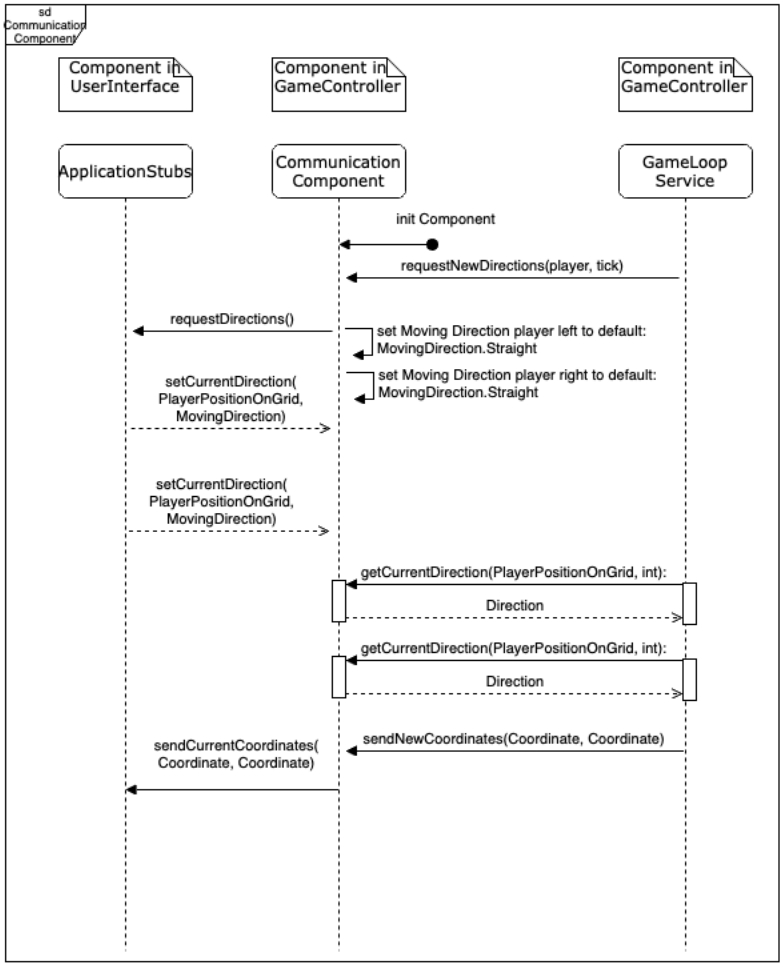


Figure 25: Ablaufsemantik eines GameLoops als Sequenzdiagramm im CommunicationComponent (eigene Darstellung).

x) SearchGame

Input: Button aktivität
Output: Liste von Hosts

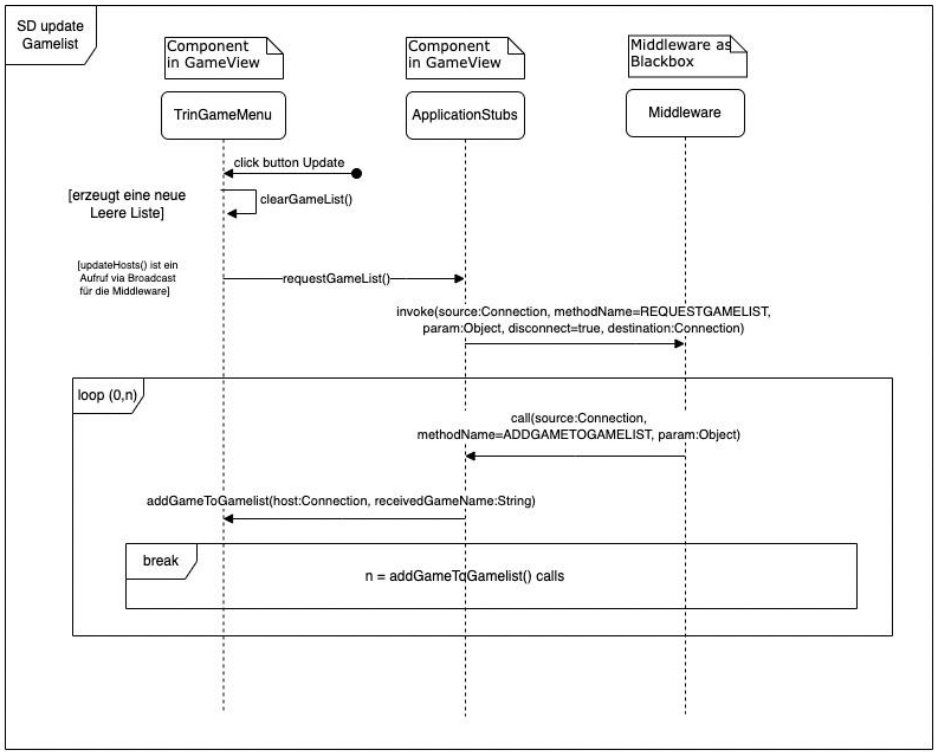


Figure 26: Ablaufsemantik einer Spielsuche als Sequenzdiagramm (eigene Darstellung).

y) Join game

Input: Button Aktivität (Spiel wird aus Spielliste ausgewählt und der join Game Button angeklickt)

Output: Verbindung zum Spiel

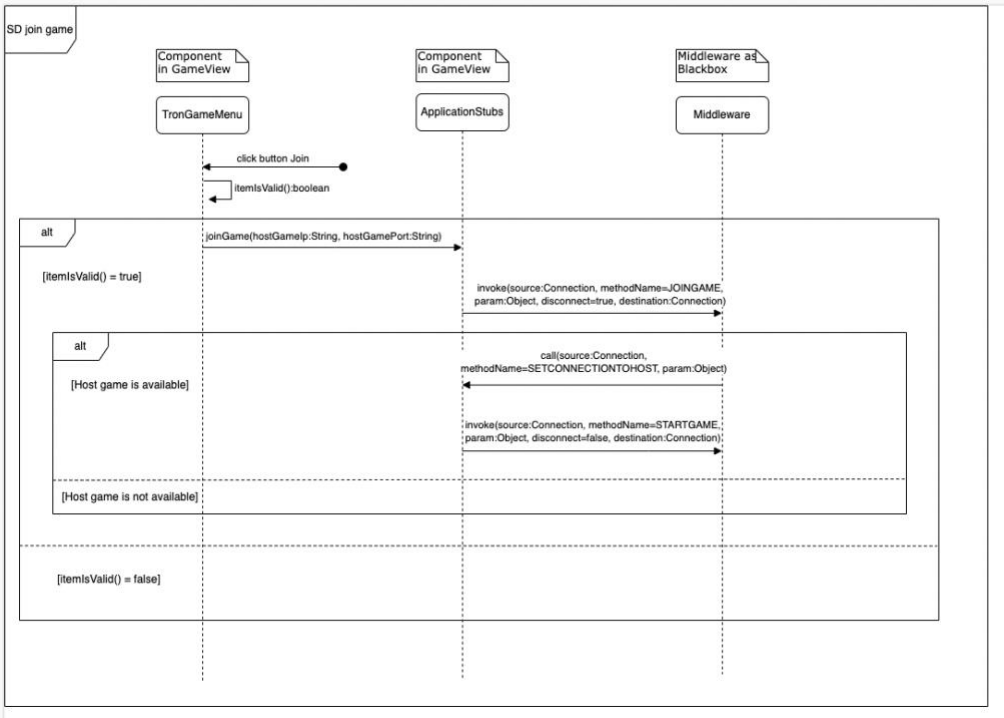


Figure 27: Ablaufsemantik Spielbeitritt (joinGame) als Sequenzdiagramm (eigene Darstellung).

z) Set host game

Input: Button Aktivität (Der Host Game Button wurde im Menü angeklickt)

Output: Game als Host, dass auf einen Mitspieler wartet. (Außer im lokalen Modus. Dann kann direkt ein Spiel gestartet werden)

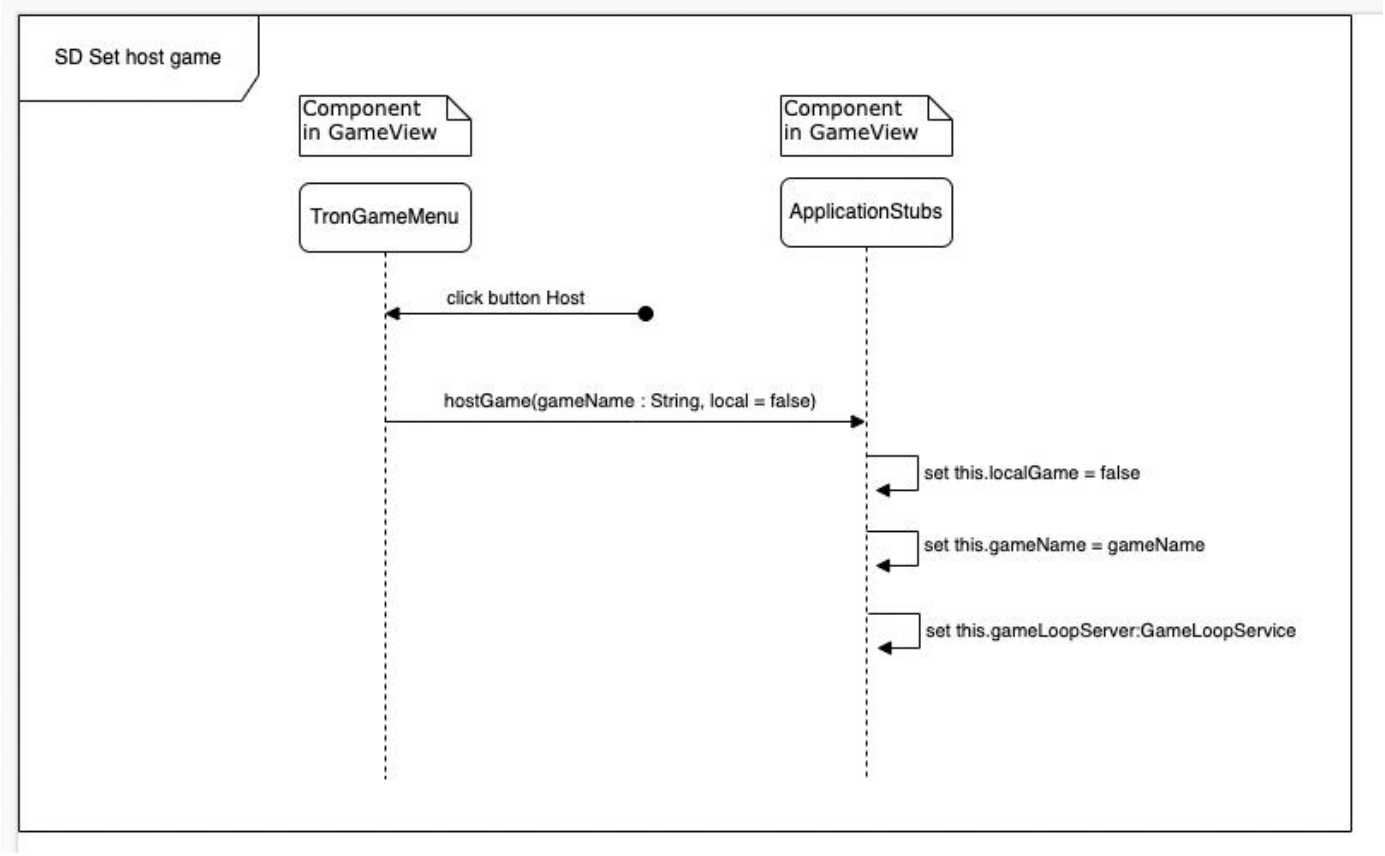


Figure 28: Ablaufsemantik hosten eines Spieles (hostGame) als Sequenzdiagramm (eigene Darstellung).

aa) Host game response

Input: Method call

Output: Response or run game

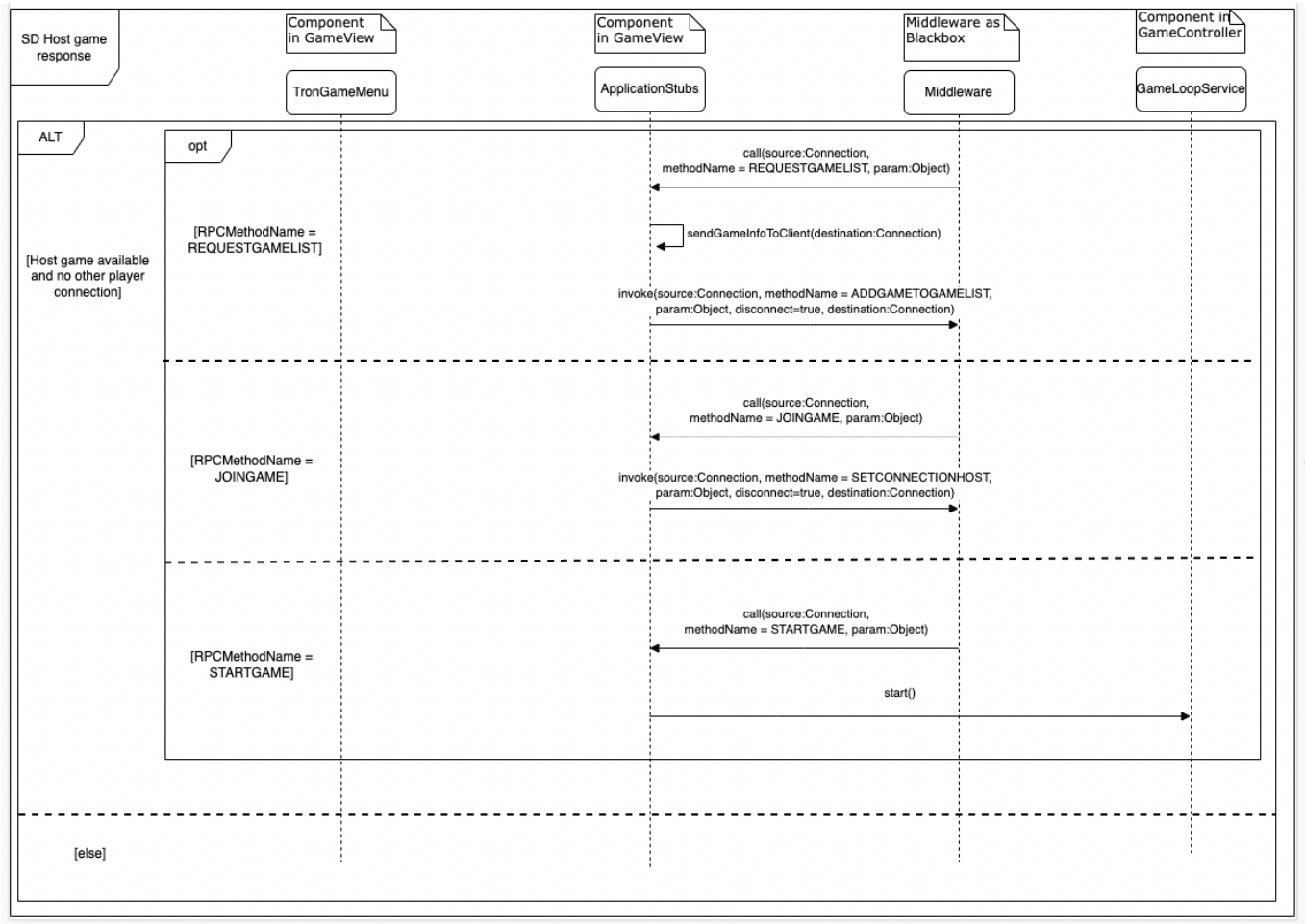


Figure 29: Ablaufsemantik Finden, Beitreten und Starten eines Spieles als Sequenzdiagramm (eigene Darstellung).

bb) Middleware ClientStubs

Input: Method call

Output: RPC

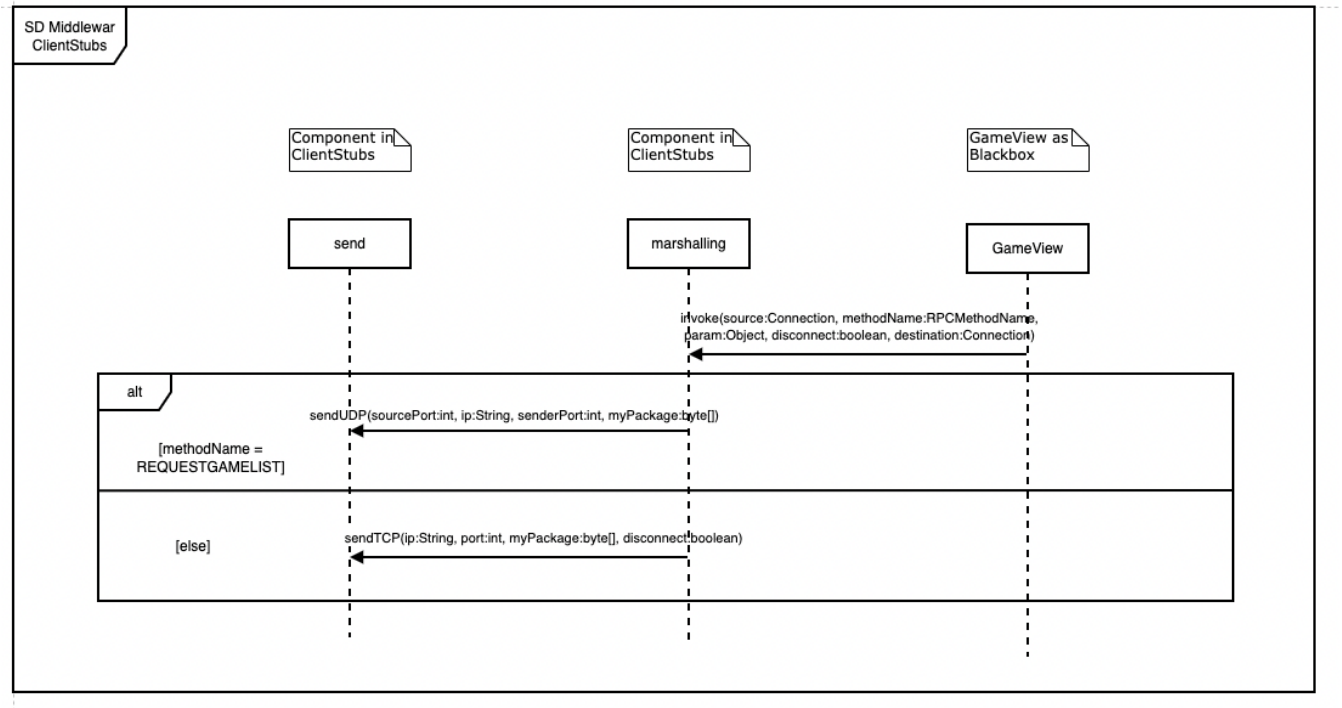


Figure 30: Ablaufsemantik invoke eines RPC und Nutzung und send via UDP oder TCP als Sequenzdiagramm (eigene Darstellung).

cc) Middleware ServerStubs

Input: RPC

Output: Method call

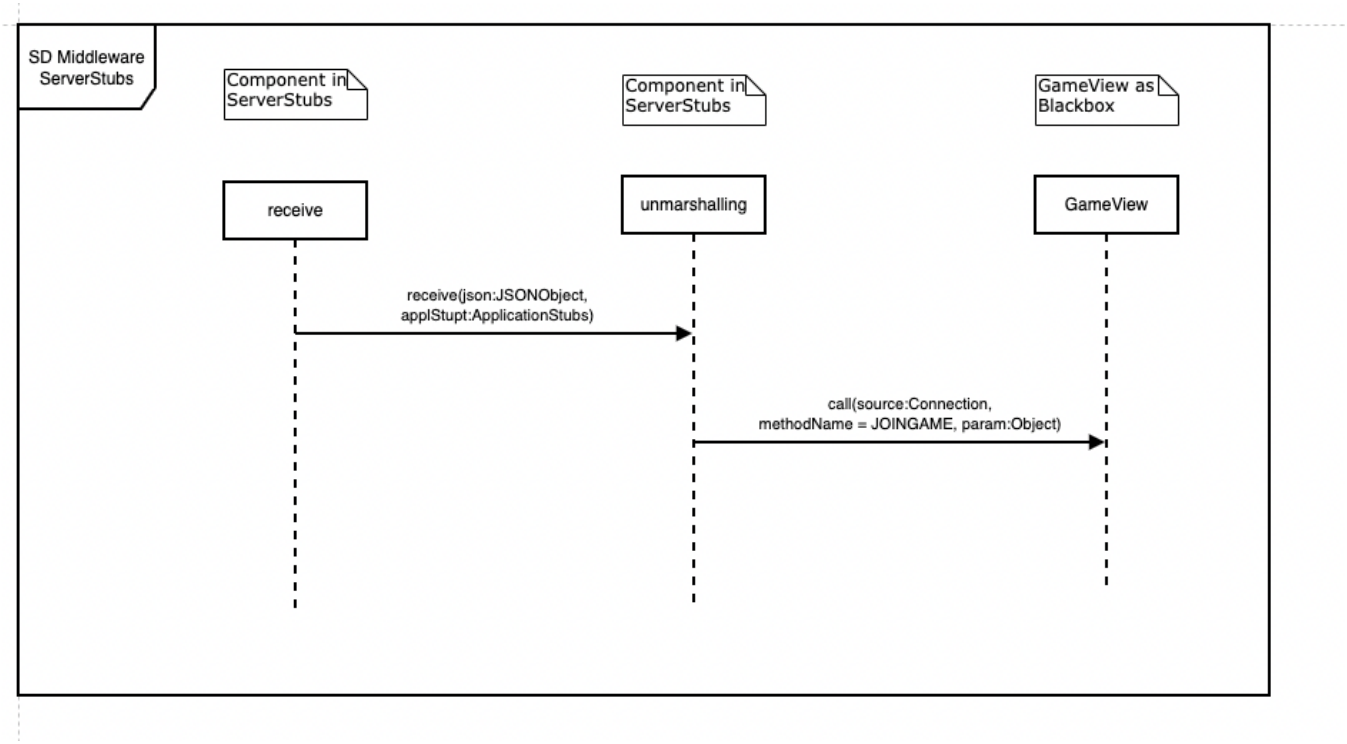


Figure 31: Ablaufsemantik receive und unmarshalling als Sequenzdiagramm (eigene Darstellung).

v. Spiel leiten

Input: Spiel Name (String)
Output: GameController

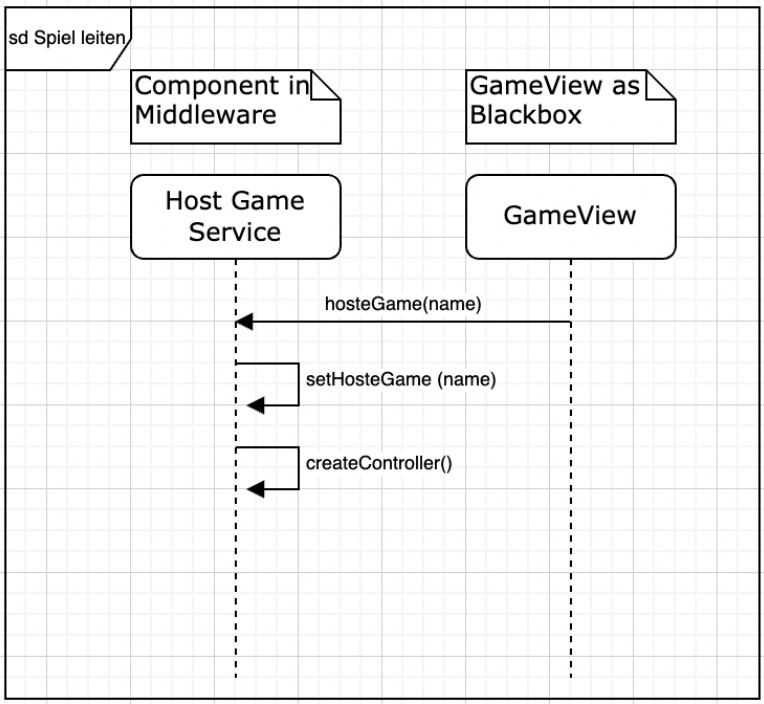


Figure 32: Ablaufsemantik ein Spiel zu leiten als Sequenzdiagramm (eigene Darstellung).

vi. Spiel veröffentlichen

Input: Client IP und Port
Output: Host Session mit Name, IP und Port

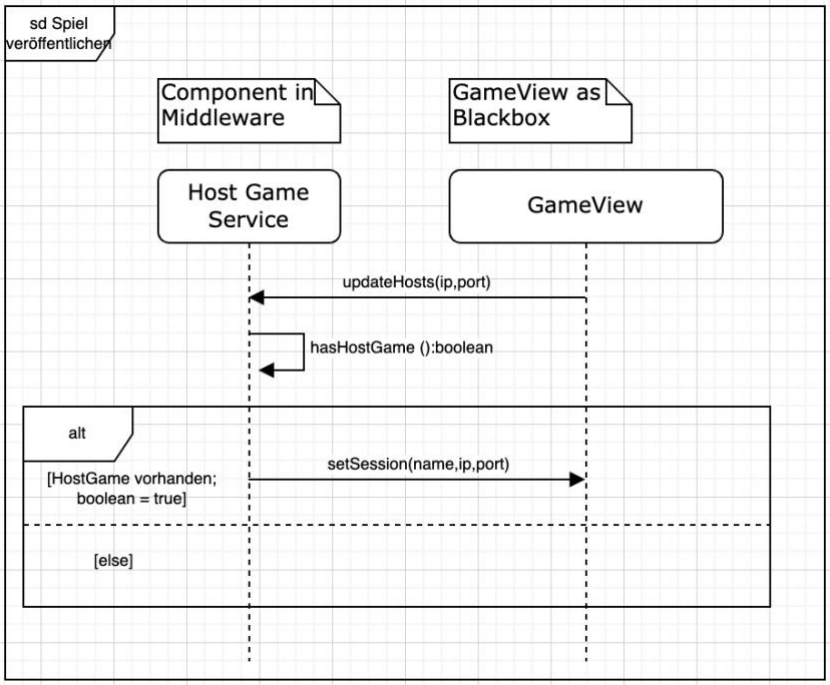


Figure 33: Ablaufsemantik Spielveröffentlich im lokalen Netzwerk für andere Spielinstanzen als Sequenzdiagramm (eigene Darstellung).

vii. Spiel beitreten

Input: Spiel Name

Output: Spiel Name

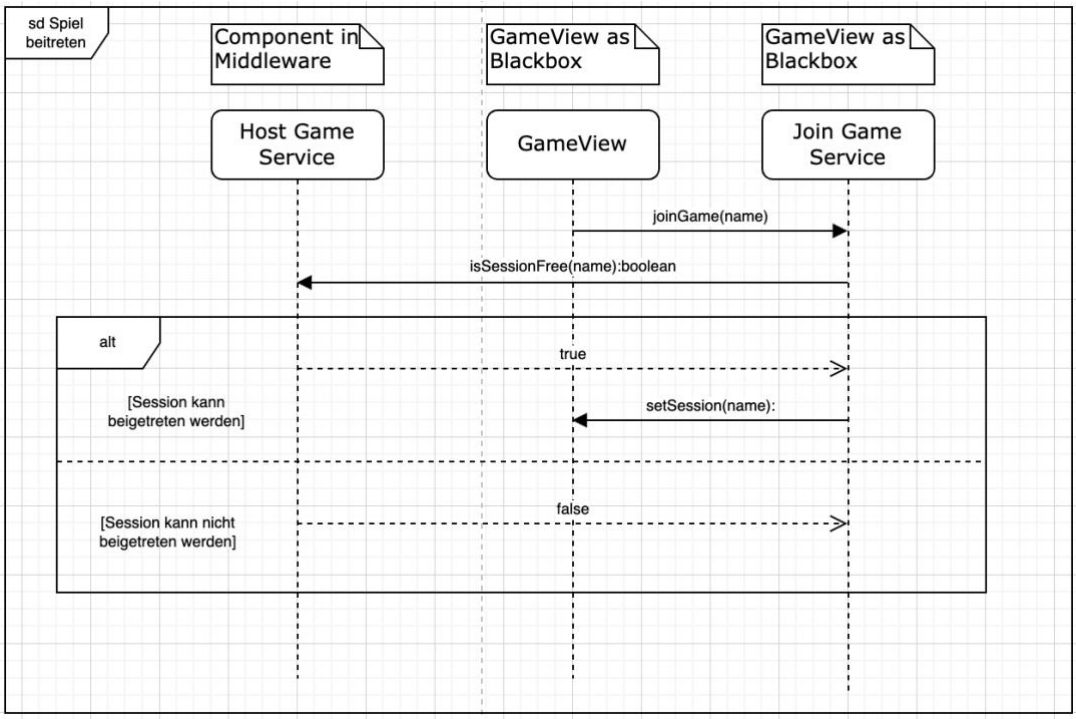


Figure 34: Ablaufsemantik Spielbeitritt als Sequenzdiagramm (eigene Darstellung).

12) Anhang

Verworfene Softwarearchitektur Option A: MVC-Pattern[11]

Als **Softwarearchitektur Option A** wird eine Entwicklung in Anlehnung an das **Model-View-Controller Pattern** nicht verwendet. Diese eignet sich generell besonders, da es eine User Interface gibt, die auf ein gemeinsames Model basieren (Spielstand, aktuelle Position). Da die View jedoch mit dem Model und dem Controller verbunden ist, verursacht es Probleme bei der Verteilung. Aus diesem Grund wurde die 3-Schichten-Architekutr verwendet. Eine Veranschaulichung ist in der folgenden Abbildung gegeben:

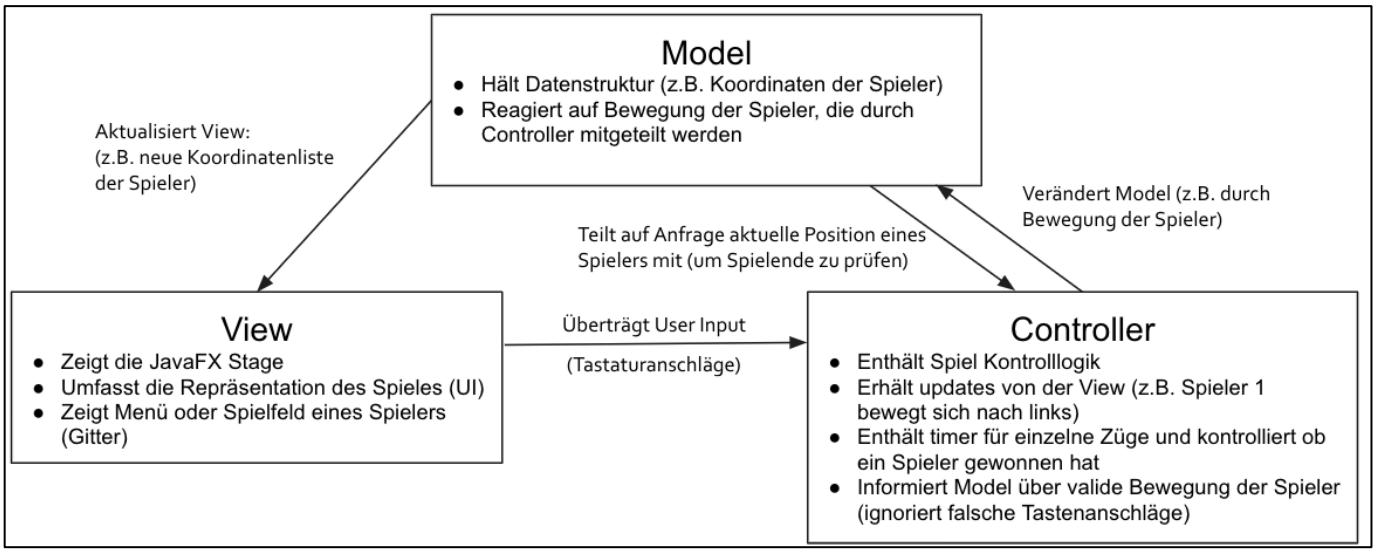


Figure 35: Option für die Architektur der Software mit MVC Pattern (eigene Darstellung).

Mit dem folgenden Klassendiagramm kann das MVC Pattern zur Entwicklung der Applikation umgesetzt werden:

- Der *GameController* in Abbildung 6 übernimmt in der Architektur die wesentliche Steuerung des Spielflusses. Nach Ablauf einer festgelegten Zeit wird eine Bewegung der Spieler über *movePlayer()* veranlasst. Das *GameModel* übernimmt die Verarbeitung der Bewegung und entscheidet, ob das Spiel durch den Zug beendet wurde (return boolean signalisiert Spielende)

- Das *GameModel* hält alle Informationen über den aktuellen Spielzustand inkl. der Koordinaten der Spieler. Nach jeder Modellveränderung (z.B. durch Bewegung) wird die *GameView* aktualisiert und die aktuelle Spielsituation durch die Koordinatenliste der Motorräder wird visualisiert.

- Die *GameView* übernimmt stellt die Schnittstelle zum Nutzer dar. Dieser wird visuell über den aktuellen Spielzustand informiert und kann durch Tastatureingaben das Spiel beeinflussen. Diese Eingaben werden an den *GameController* weitergeleitet und dort interpretiert. Falsche

[11] MVC Diagramm einsehbar unter: *https://docs.google.com/presentation/d/1BIZMIhBbEzaA7x2pHdnqUC6iriMMlvO233gPfCFWmNc/edit?usp=sharing*

Tastatureingaben werden ignoriert und bei mehreren Bewegungen innerhalb eines Zeitschrittes wird nur die erste Eingabe berücksichtigt.