# tunExponat

# Inhaltsverzeichnis

## Über arc42

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

Template Version 8.2 DE. (basiert auf AsciiDoc Version), Januar 2023

# 1. Einführung und Ziele

Das System soll ein Exponat für eine Messe werden, im Auftrag der Nachwuchsförderung der FHNW. Mit dem Ziel Faszination und Interesse an Technik und Informatik zu wecken, bei der Zielgruppe. Diese beinhaltet Kinder im Alter von etwa 10 Jahren.

## 1.1. Aufgabenstellung

### 1.1.1. Was ist Go Robo?

- "Go Robo" ist eine Spiel auf einem Raspberry
- Es soll Kindern freude und Interesse an der Informatik und Technik geben
- Die Steuerung wird mittels Physikalischen "Karten" mittels einem "Scanner" ausgelöst
- Mittels einer gewissen Komplexität soll wärend dem Spielen, das Informatische denken aufgezeigt werden
- Das Spiel wurde für einen Messestand der Nachwuchsförderung entwickelt

### 1.1.2. Wesentliche Features

- hoher Spassfaktor
- geringe Komplexität
- kurze Spieldauer
- haptische Komponenten
- stabil & transportierbar
- zuverlässiger Betrieb

Genauere Features beschreibungen:

- Steuerung Spielfigur:
  - Das Steuerungssystem mit Richtungskarten ist ein Spielerlebnis, bei dem die Spieler vordefinierte Karten verwenden, um die Bewegungen einer Spielfigur im Spiel zu bestimmen.
- Interaktion Levels:
  - Das Ziel des Leveldesigns fesselnder und herausfordernder Spielabschnitte, die die Spieler motivieren und in das Spiel eintauchen lassen. Integration von Lernelementen, die grundlegende Konzepte der Informatik und Technik vermitteln.
- Case:
  - Stabiles, 3D-gedrucktes Case aus Kunststoff, mit Plexiglasverschluss, mit einer externen Schnittstelle und leicht zugänglicher Hardware. Das Case enthält eine eingebaute RFID-Antenne und einen Raspberry Pi sowie eine ordentliche Verkabelung. Es verfügt über eine externe Schnittstelle für eine einfache Verbindung mit Bildschirmen mittels HDMI Kabel.

Dieses Projekt dient auch dazu die teilnehmenden Studierenden anhand praxisnaher Projekte weiterzubilden.

Genauer Dokumentierten Anforderung: https://fhnw-projecttrack.atlassian.net/wiki/x/1wHPAw

## 1.2. Qualitätsziele

| Qualitätsziele | Beschreibung |
|---|---|
| Akzeptable Spielstärke (Funktionale Eignung) | Die "Levels" so so gestaltet, das der Spiele ins Grübel kommt, aber nicht die lust am Spiel verliert |
| Spassfaktor (Attraktivität) | Das System ist ein Exponat an einer Messe, damit es seinen Job erfüllen kann müssen die Kinder Spass am Spiel haben. |
| zuverlässiges System (Stabilität) | Der Betrieb vom System soll ohne Fehler funktionieren. |
| zuverlässiges System (Leistungseffizienz) | System hat eine vernünftige Antwortzeit |

Aktuelle eingeplante Projekt Qualitätsziele: https://fhnw-projecttrack.atlassian.net/wiki/spaces/IP1223vt1/pages/63898071/Requirements#Qualit%C3%A4tsanforderungen

Im Abschnitt 10 werden diese Ziele konkreter beschrieben, sowie wie deren Erreichung bewertet wird.

## 1.3. Stakeholder

| Rolle | Kontakt/Ansprechsperson | Erwartungshaltung |
|---|---|---|
| Produktowner/Auftraggeber(Nachwuchsförderung FHNW) | Christine Enggist christine.enggist@fhnw.ch | Kommunikation zwischen Nachwuchsförderung und Entwicklungsteam, sowie klare Zielvorstellungen. Sicherstellen, dass das Ziel die Zielgruppe erreicht und das Spiel soll wie in den Anforderungen deklariert funktionieren. |
| Entwicklungsteam (Projekt Gruppe) | Sven Christen | Verantworlich für die technische Umsetzung des Projektes, sowie benötigt auch klare Anweisungen der Richtlinien für den Entwicklungsprozess, mögliche Wartung oder Dokumentationen |

| Rolle | Kontakt/Ansprechsperson | Erwartungshaltung |
|---|---|---|
| Sponsor(FHNW) | sibylle.peter@fhnw.ch | Bieten unterstüzungen und Resourcen, sowie Einschränkungen (z.B. Raspberry PI nutzen). |
| Messe-Exhibitor-Team(Messen) | tunSolothurn, tunBasel | Kommunikation mit Produktowner, sowie legt Einschränkungen fest. |
| Zielgruppe(Nutzer) | Schulkinder und Jugendliche | Ein ansprechendes, lehrreiches, unterhaltsammen Spielerlebnis interessiert. Erwartet ansprechendes Design. Feedback der Endnutzer kann für Zukünftige Updates/ Verbesserungen genutzt werden. |

Aktuelle Doku für Stakeholder: https://fhnw-projecttrack.atlassian.net/wiki/spaces/IP1223vt1/pages/63898052/Produktvision#Stakeholders

# 2. Randbedingungen

## 2.1. Technische Randbedingungen

| Randbedingung | Erläuterung |
|---|---|
| Programmierung in Java 17 | Vorgabe gemäss der Projektschiene. |
| Verwendung der Library Pi4J | Vorgabe gemäss der Projektschiene. |
| Betrieb auf Raspberry Pi | Das ganze System soll zentral über einen Rpi gesteuert werden, dabei dürfen können Komponenten daran angeschlossen werden. Vorgabe gemäss der Projektschiene. |
| Haptische Komponenten | Das System soll mit haptischen Komponenten gebaut werden. |

## 2.2. Organisatorischen Bedingungen

| Randbedingung | Erläuterung |
|---|---|
| Team | Definiert durch Projektschiene; Sven Christen, Merlin van der Kolk, Simy Melinda Tran, Nick Rindlisbacher, (Jan Schöni), Andrin Martig, Dean Bregenzer |
| Coaching | Als Unterstützung von der Projektschiene zur Verfügung gestellt; Thekla Müller (IP1), (Lucia Di Caro (IP1-2)), Stefan Vetter(IP2) |
| Zeitplan | Definiert durch FHNW; Beginn der Entwicklung am 11.09.2023, erster lauffähiger Prototyp 15.11.2023, Fertigstellung Juni 2024. |
| Datenablage | Gemäss Projektschiene sollen alle Informationen rund um das Projekt im dazugehörigen Confluence abgelegt werden. |
| Konfigurations- und Versionsverwaltung | Im Gitlab welches von der Projektschiene zur Verfügung gestellt wurde. |
| Zielgruppe | Kinder im alter von ca. 10 Jahren. |

## 2.3. Konventionen

| Konvention | Erläuterung |
|---|---|
| Architekturdokumentation | Terminologie und Gliederung nach dem deutschen arc42-Template in der Version 6.0 |
| Kodierrichtlinien für Java | Java Coding Conventions von Sun/Oracle, geprüft mit Hilfe von CheckStyle |
| Sprache (Deutsch vs. Englisch) | Java Code und Bennenung von Komponenten in Englisch. Diagramme und Texte auf Deutsch |

## 2.4. Randbedingung des Lauffähigen Produktes

Innerhalb welcher Rahmenbedingungen funktioniert das Produkt zuverlässig?

Auf das Wesentliche begrenzen:

- Steckdose in 1-m Nähe
- Stromkabel
- Operation in Raumtemperatur
- Mikro HDMI auf HDMI
- Monitor
- Lüftung / Dichtheit
- Registrierte Bewegungskarten (Pro Richtung eine, sowie eine Reset und NextLevel) *(Bei Karten Scannung max. Karten Abstand zu Scanner 0-60 mm)

Für aktuelle Coding Konvention siehe: https://gitlab.fhnw.ch/ip12-23vt/ip12-23vt_tun-exponat/docu/-/blob/main/coding_conventions.adoc?ref_type=heads#user-content-verwendete-konventionen

# 3. Kontextabgrenzung

Hier wird das Umfeld vom Exponat beschrieben.

## 3.1. Fachlicher Kontext

Das Spiel wird gegen keinen Gegner gespielt, der Spieler oder die Spielerin muss eigenständig die gegebenen Levels lösen.

**System: Go Robo**

| Kommunikationsbeziehung | Eingabe | Ausgabe |
|---|---|---|
| Level erstellung | Neue / veränderte .csv Datei (mit Format: LEVELSCHWIERIGKEITSNUMMER_LEVELNUMMER) | Ein Neues / verändertes Level |
| Spielkonfigurationen | Veränderungen der Werte in der app.properties Datei | Verändertes Spielverhalten nach den Konfigurationen |
| Level-Fortschritt | Signal/Event, dass das aktuelle Level erfolgreich abgeschlossen wurde (Battery wurde eingesammelt) | Starten des nächsten Levels |
| Processing Library (Fremdsystem) | System Logik | Grafische darstellung des Spiels über die Fremdsystem oberfläche Processing "angebunden" |
| Pi4j Library (Fremdsystem) | Eingabe der erkannten Phyischen Elemente | Event auslösung für dieses Phyischen Elemente im Spiel |

## 3.2. Technischer Kontext

**System: Go Robo**

| was | Eingabe | Ausgabe |
|---|---|---|
| Pi4j Library (Fremdsystem) | Erfassung der erkannten physischen Elemente durch den RFID Reader mit Hilfe der Pi4J Library | Auslösung von Events im Spiel in Reaktion auf erkannte physische Elemente |

# 4. Lösungsstrategie

In diesem Abschnitt wird die ein Überblick über die Architektur geschaffen. Die Ziele und ihre Lösungsansätze werden zudem angeschaut.

| **Qualitätsziele** | **Massnahmen dafür in der Architektur** |
|---|---|
| hoher Spassfaktor | - Aufbau wie ein Spiel<br>- ansprechendes Design<br>- Fortschritt durch mehrere Levels<br>- Durch Fortschritt herausfordernder |
| zuverlässiges System (Die Software soll möglichst Störungsfrei laufen) | - Robuste Systeme werden verwendet<br>- bekannte Programmiersprache im Einsatz (Java)<br>- Nur Fremdsysteme (Librarys) im Einsatz, welche von Dozenten empfolen wurden |
| Leistungseffizienz (Die Reaktionszeit zwischen Karte platzieren und Spielfigurenbewegung soll minimal sein) | - Processing Library ist performant<br>- RPI hat angemessen viel Rechenleistung<br>- Spiel wird grafisch simpeler gehalten<br>- grafische Elemente mit hoher auflösung nur reduziert im Einsatz |

## 4.1. Aufbau

Das Spiel Go Robo ist als Java Programm mit dem MVC Pattern konzipiert und kann in folgend Ablauf wie folgt grob aufgeteilt werden

- Auf Physisches MVC und auf logischem MVC
- Ansteuern des RFID Readers auf Physischem MVC mit inizialisierung auf dem logischem
- Anbindung an eine grafische Benutzeroberfläche über Fremdsystem Processing
- verschiedene Logische Klassen welche den Spiel Architektur beinhalten
- Hauptklasse, welches den Spielablauf handelt

Weitere Details, siehe 5. Bausteineinsicht

## 4.2. Levels

Die verwendet Levels im Spiel werden von Hand erstellt und sollen so eine für die Zielgruppe angepasste Schwierigkeit haben. siehe 3.Kontextabgrenzung für genaueres

## 4.3. Spielstrategie

Das Spiel besitzt keine eigenen Algorithmen, da die Gedankengänge für die Spieler konzipiert sind. Es bietet sich aber ein z.B. ein Minimax-Algorithmus zu Implementieren um am Schluss den Optimalsten weg anzuzeigen.

## 4.4. Anbindung

Go Robo erzeugt mittels der Fremd Library Processing ein UI und kann mittels Physisches "Karten" gesteuert werden.

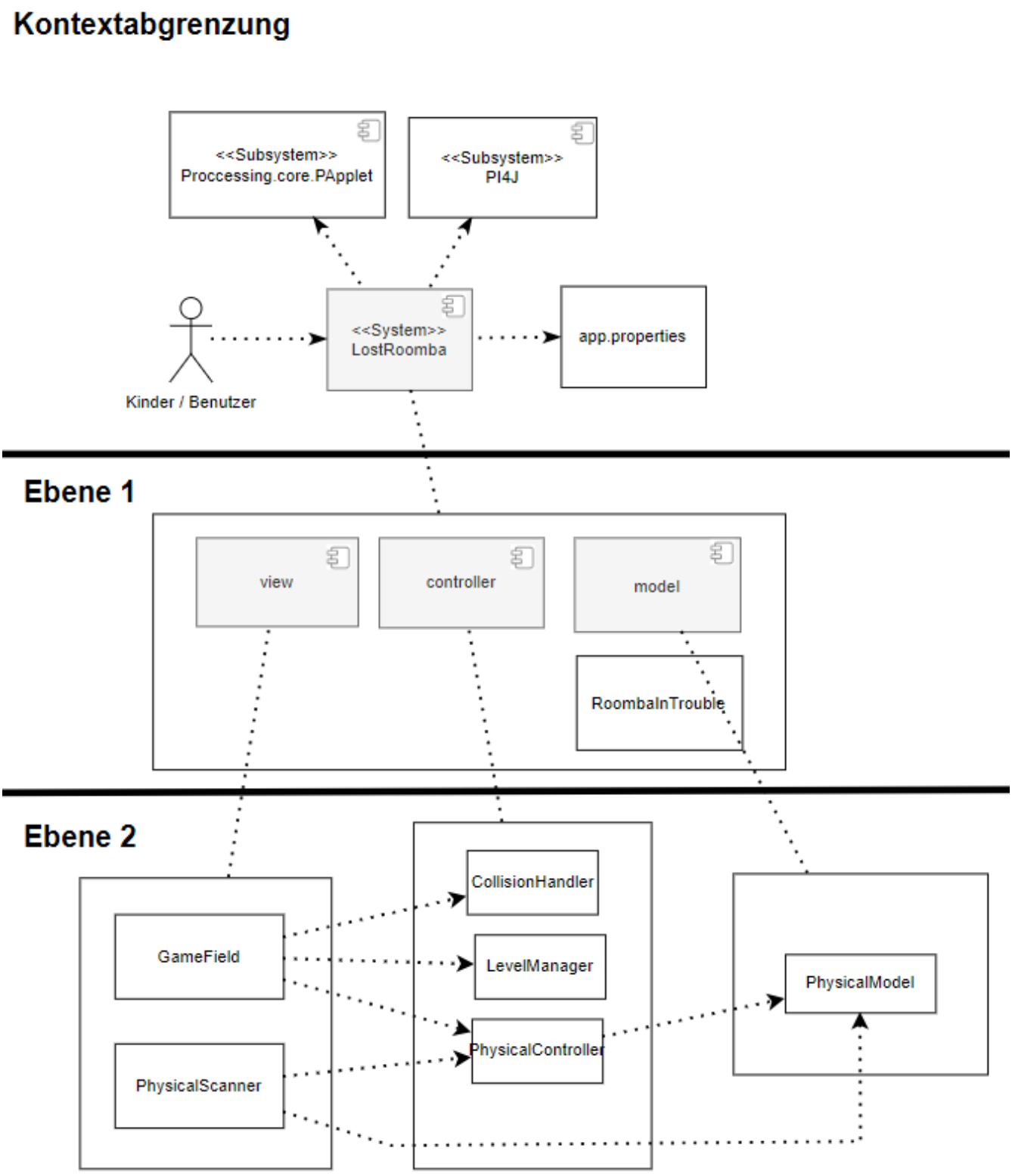
## 4.5. Angewante Lösungsansätze

- Um die Physischen Karten im Spielkontext zuverlässig zu registrieren wird jeder Processing Gameloop geschaut ob sich in der Queue der Karten etwas drinn ist, da die Implementierung des Observables nicht so verhalten hat wie erwartet.
- Um einen Collision bug zubeheben musste in AnimatedSprite von jeden Frame zu jeden 5ten Frame die Richtung überprüft werden, ansonsten kam es vor das der Spieler durch wände bugte.
- Die Karten sind nicht am Produkt befästigt, da dies sonst zu stark im Spielverläuf störend würde, das würde auch mit der Kundin abgesprochen.
- Um den Raspberry auch unter Hallen bedingungen Kühl zuhalten wird am Raspberry ein Lüfter angebaut, sowie am Case wurden Lüftungsrillen berücksichtigt.
- Da es mit Processing und dem MVC Pattern schwer und zeitaufwendig ist, gute Unittest zu schreiben, wird als Ersatz ein Testprotokoll erstellt, welches aufzeigt, wie von Hand, mit welchen Inputs und mit welchen Ergebnissen getestet werden kann.
- Damit der Raspberry bei verschiedensten Bildschrimen immer die gleiche Auflösung nimmt wurde im PI die Auflösung Fix FHD in settings geschalten.
- Damit der RFID input effizient ist wird nun ein Event geschossen um die Jeweiligen Componenten die ID mitzuteilen.

```
Geplant:
* Damit das Spiel automatisch startet nach dem der PI eingeschalten wude, wird noch
am PI der Kiosk mode eingeschalten.
```

# 5. Bausteinsicht

In diesem Abschnitt wird die Architektur vom Spiel dargestellt.




## 5.1. Whitebox Gesamtsystem

**Ebene 1** ist die Whitebox-Beschreibung des Gesamtsystems, zusammen mit Blackbox-Beschreibungen der darin enthaltenen Bausteine.

**Ebene 2** zoomt in einige Bausteine der Ebene 1 hinein. Sie enthält somit die Whitebox-Beschreibungen ausgewählter Bausteine der Ebene 1, jeweils zusammen mit Blackbox-Beschreibungen darin enthaltener Bausteine.

**Begründung**

Das Gesamtesystem basiert auf einem Raspberry Pi, welcher als Hardware-Plattform dient. Die Benutzerinteraktion wird duch PI4J verarbeitet und an das System RoombaInTrouble weitergeleitet. Proccesing wird für die grafische Darstellung und Spiellogik verwendet.

**Enthaltene Bausteine**

*<Beschreibung der enthaltenen Bausteine (Blackboxen)>*

| Subsystem | Kurzbeschreibung |
|---|---|
| PI4J | Verarbeitet Benutzereingaben und leitet sie an das System weiter. |
| Proccesing.core.PApplet | Bietet eine integrierte Umgebung für Grafik, Animation und Spielinteraktionen. Dient als Basis für die gesamte Spielvisualisierug und - Steuerung |

**Wichtige Schnittstellen**

- **Benutzereingaben:** Schnittstellefür die Interaktion mit dem Benutzer über Richtungskarten.

# 5.2. Kontextabgrenzung

### 5.2.1. 0. Benutzer

**Zweck/Verantwortung** Der Benutzer bedient das Spiel.

### 5.2.2. 0. Whitebox PI4J

**Zweck/Verantwortung** PI4J fungiert als Brücke zwischen den Benutzereingaben und dem Spiel, indem es EIngaben von externen Geräten (Richtungskarten) empfängt aufbereitet. Es gewährleistet die reibungslose interaktion des Spielers mit dem Spiel

**Schnittstelle(n)** Nimmt Eingaben von Richtungskarten auf und konvertiert sie in spielinterne Befehle.

### 5.2.3. 0. Whitebox Proccesing.core.PApplet

**Zweck/Verantwortung:** Proccesing.core.PApplet stellt die Grundlage für die grafische Darstellung und Interaktion des Spiels dar. Es ermöglicht das Zeichnen von Objecten, die Animation von Charakteren und die Handhabung von Benutzerinteraktion innerhalb des Spiels.

**Schnittstellen:** Bereistellung von Funktionen zum Zeichnen von Spielobjekten und szenen.

### 5.2.4. 0. app.properties

**Zweck/Verantwortung:** Eine Datei die zur veränderung von Spielwerten genutz wird.

**Schnittstelle(n)** Übergibt die Werte dem Spiel.

### 5.2.5. 0. Blackbox LostRoomba

**Zweck/Verantwortung:** System, Ordner in dem alle Dateien für das Spiel beherbergt.

**Schnittstelle(n)** Arbeitet mit der PI4J und PApplet Bibliothek.

## 5.3. Ebene 1

### 5.3.1. Blackbox view (Ordner)

**Zweck/Verantwortung** Der Ordner beinhaltet nach der MVC Struktur alle Dateien welche für die visuelle game Darstellung genutzt werden.

### 5.3.2. Blackbox controller (Ordner)

**Zweck/Verantwortung** Der Ordner beinhaltet nach der MVC Struktur alle Dateien welche für die Gamelogik zuständig sind.

### 5.3.3. Blackbox model (Ordner)

**Zweck/Verantwortung** Der Ordner beinhaltet nach der MVC Struktur alle Dateien welche Gamedaten "Baupläne" beinhalten.

### 5.3.4. Whitebox RoombaInTrouble

**Zweck/Verantwortung** Die Starterklasse, welche das Spielstartet und die Physischen Komponente einsatz bereit macht. **Schnittstelle(n)** Kommuniziert mit dem PhysicalController und holt sich die neusten Eingaben.

## 5.4. Ebene 2

### 5.4.1. Whitebox GameField (view)

**Zweck/Verantwortung** Eine view welche zuständig ist alle controller zu bündigen und diese Informationen Sinnvoll darzustellen.

### 5.4.2. Whitebox PhysicalScanner (view)

**Zweck/Verantwortung** Die view handelt die Bindung zwischen Komponente (physisch) und Code.

**Schnittstelle(n)** Die view handelt die Kommunikation mit dem physischen Model und dem controller.

### 5.4.3. Whitebox CollisionHandler (controller)

**Zweck/Verantwortung** Gibt an als was eine Collision zählt.

**Schnittstelle(n)** Wird von Player und GameField verwendet.

### 5.4.4. Whitebox LevelManager (controller)

**Zweck/Verantwortung** Generiert aus den csv Dateien GameObjecte welche als Level fungieren.

**Schnittstelle(n)** GameField nutzt dies.

### 5.4.5. Whitebox PhysicalController (controller)

**Zweck/Verantwortung** Spricht mit dem Model um auf den Componenten Input zuzugreifen.

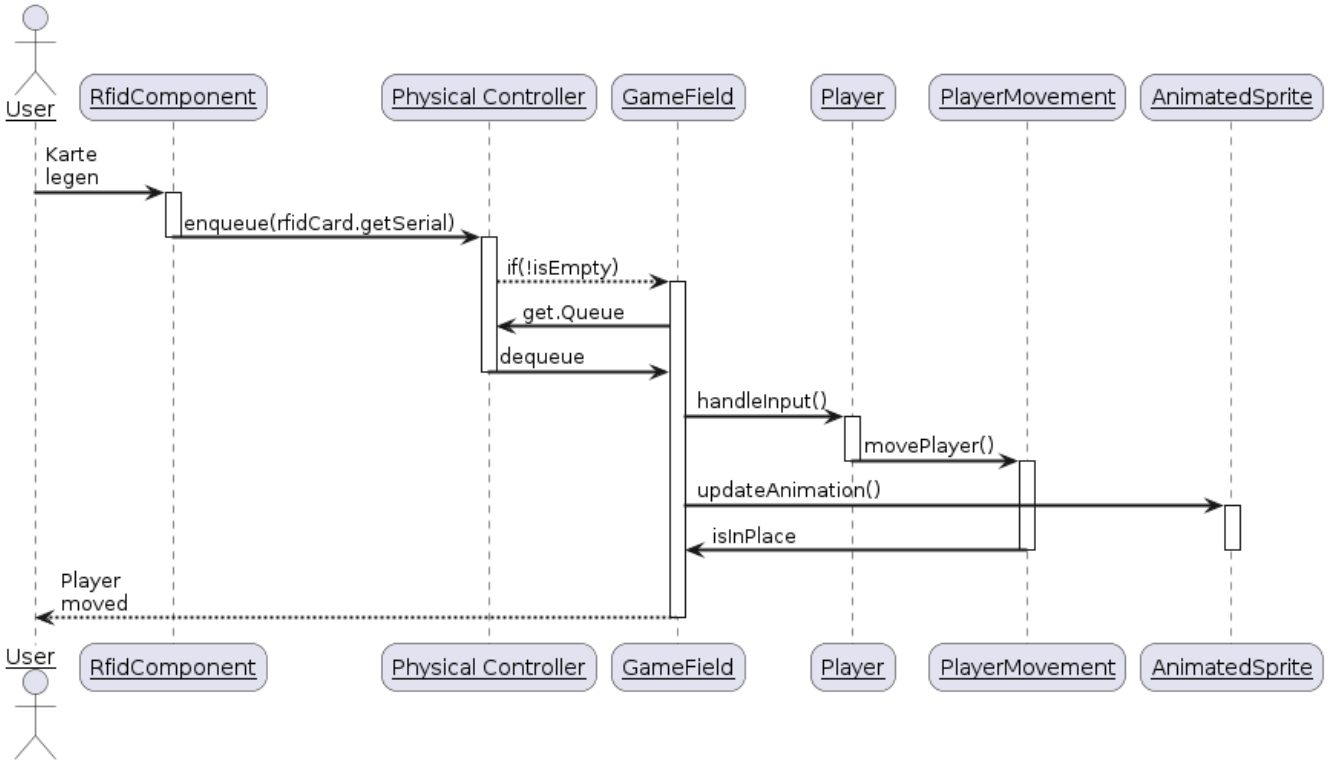
**Schnittstelle(n)** physischeModel

### 5.4.6. Whitebox PhysicalModel (model)

**Zweck/Verantwortung** Speichert die neuen Componenten eingaben.

# 6. Laufzeitsicht

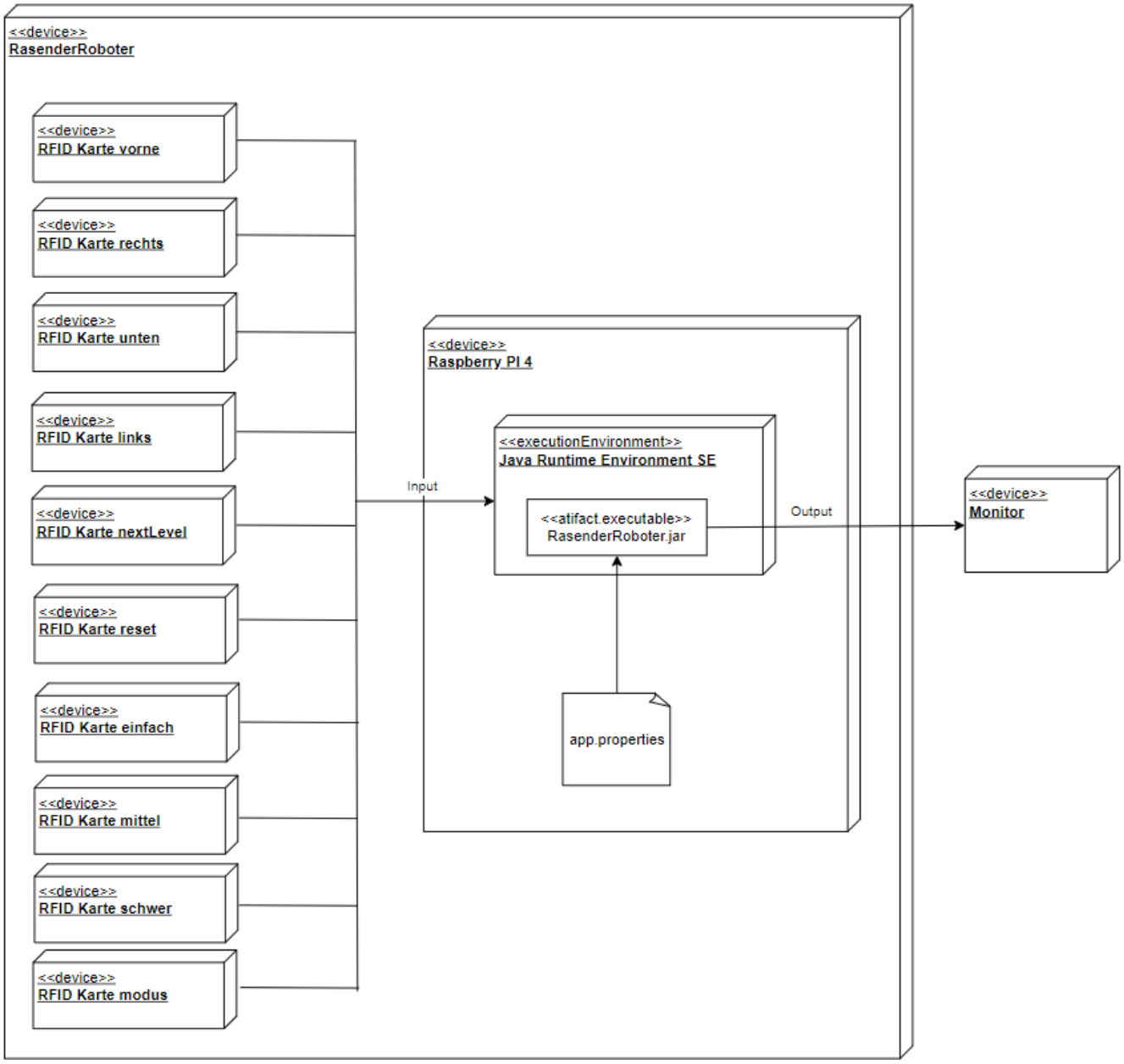
## 6.1. Laufzeitszenario 1 "Karte legen"




## 6.2. Ablauf

Die RFID Karte wird mittels der PI4J libary erkannt. Die Karten Informationen werden dann dem Model übergeben. Aktuell wird in jedem Gameloop geschaut ob es eine neue Karte im Model hat, falls ja wird diese dequed und mittels der Proccessing libary wird das movement geupdated.

# 7. Verteilungssicht




Das Endprodukt ist ein Rätselspiel das an verschiedenen Messen ausgestellt wird, damit Kinder eine Spielerische sicht bekommen wieso man an die FNHW kommen soll bzw. wieso diese Berufe interessant sind.

## 7.1. .1 Software voraussetzungen:

- Java Runtime Environment SE 17
- Betriebsystem: Pi4J-Basic-OS (pi4j-download.com/latest.php?flavor=basic)
- Processiong.core.PApplet Bibliothek
- pi4j Bibliothek

## 7.2. .2 Konfiguration und Einrichtung:

- Die Javaanwenudung (.jar) enthält den Kompilierten Sourcecode mit sämtlichen Abhängigkeiten, welche beim Remote-Deploy übergeben werden
- Das Spiel wird direkt auf dem Raspberry PI ausgeführt
- Durch das verändern der Datei "app.properties" können verschiedene Spiel veränderungen gemacht werden
- Durch das verändern der CSV dateien können die level angepasst werden und durch neue Dateien neue hinzugefügt werden

## 7.3. .3 Benutzerinterface

- Die Ausgabe der Benutzeroberfläche wird über den angeschlossenen Monitor dargestellt
  - Port: HDMI (vom Monitor und Micro HDMI für den PI)
- Die Eingabe erfolgt über RFID Karten, welche mittels eines Scanners erkannt werde.
  - Karte wird auf Scanner gehalten
- Die Administrativen Spiel Konfigurationen geschehen über Maus und Tastatur.
  - Port: USB 3.0

Zusästzliche Informationwn für die Pi Software pi4j.com/pi4j-os/

# 8. Querschnittliche Konzepte

## 8.1. 1. Fehlerbehandlung und Logging

Erklärung:

Die Fehlerbehandlung und das Logging sind wesentliche Bestandteile eines robusten Systems. Für "Go Robo" sind spezifische Mechanismen und Strategien zur Fehlererkennung, -behandlung und -protokollierung implementiert.

Details:

- Fehlererkennung: Das System erkennt Fehler durch Validierungseingaben und durch Überwachung des Systemzustands in Echtzeit.
- Fehlerbehandlung: Bei Auftreten eines Fehlers versucht das System, den Fehler zu beheben oder zu umgehen, um den Spielablauf nicht zu unterbrechen. Dies beinhaltet das Zurücksetzen bestimmter Komponenten oder das Neustarten von Services.
- Logging: Alle Fehlerereignisse und bedeutende Systemaktivitäten werden in Logdateien protokolliert. Diese Protokolle werden regelmäßig überprüft, um potenzielle Probleme frühzeitig zu erkennen und zu beheben.

Implementierung:

- Verwendung der java.util.logging-Bibliothek für das Loggen. Fehler und Ereignisse werden in einer zentralen Logdatei gespeichert, die auf dem Raspberry Pi abgelegt wird.

## 8.2. 2. Sicherheitskonzept

Erklärung:

Da "Go Robo" auf Messen und in öffentlich zugänglichen Bereichen eingesetzt wird, sind bestimmte Sicherheitsmaßnahmen erforderlich, um die Integrität des Systems und der Daten zu gewährleisten.

Details:

- Physische Sicherheit: Das Gehäuse des Raspberry Pi ist stabil und verschlossen, um den Zugang zu internen Komponenten zu verhindern.
- Zugriffssteuerung: Nur autorisierte Personen haben Zugang zur Konfigurationsdatei und können Änderungen an den Systemeinstellungen vornehmen.
- Datenintegrität: Alle Daten, einschließlich der Spielkonfiguration und Leveldaten, werden regelmäßig gesichert, um Datenverlust zu vermeiden.

Implementierung:

- Konfigurationsdateien sind auf dem PI und nur durch Berechtigungen geschützt sofern es remote verändert wird.

- Regelmäßige Backups der Konfigurations- und Leveldateien werden auf einem Repo gespeichert.

## 8.3. 3. Benutzerschnittstellen-Design

Erklärung:

Das Benutzerschnittstellen-Design ist darauf ausgerichtet, eine intuitive und ansprechende Benutzererfahrung für die Zielgruppe zu gewährleisten. Das Design muss sowohl funktional als auch visuell ansprechend sein, um die Aufmerksamkeit und das Interesse der Kinder zu gewinnen.

Details:

- Einfache Navigation: Die Benutzeroberfläche ist so gestaltet, dass Kinder leicht navigieren und die Spielsteuerungen intuitiv verstehen können.
- Visuelle Anreize: Helle Farben, ansprechende Grafiken und Animationen sind integriert, um das Interesse der Kinder zu wecken und zu halten.
- Feedback: Das System gibt visuelles und akustisches Feedback, um den Benutzer über erfolgreiche Eingaben oder Fehler zu informieren.

Implementierung:

- Die Benutzeroberfläche wird mit der Processing-Bibliothek erstellt.
- Farbschema und Layout wurden unter Berücksichtigung der Zielgruppe entwickelt.

## 8.4. 4. Performanzoptimierung

Erklärung:

Die Performanzoptimierung stellt sicher, dass das Spiel reibungslos und ohne Verzögerungen läuft, insbesondere auf der Hardware des Raspberry Pi, die begrenzte Ressourcen hat.

Details:

- Effiziente Code-Struktur: Der Code ist so optimiert, dass er ressourcenschonender arbeitet.
- Minimierung von Latenzen: Die Reaktionszeit zwischen der Eingabe einer Karte und der Ausführung der entsprechenden Aktion im Spiel wird minimiert.

Implementierung:

- Verwendung von Profiling-Tools, um Leistungsengpässe zu identifizieren und zu beheben.
- Optimierung der Algorithmen und Datenstrukturen, um die Verarbeitungsgeschwindigkeit zu erhöhen.

# 9. Architekturentscheidungen

In diesem Abschnitt werden die wichtigsten Architekturentscheidungen beschrieben.

| **Entscheidung** | **Erläuterung** |
|---|---|
| Grundlegene Architektur | Für die Grundarchitektur verwenden wir das MVC-Muster, wie im vierten Kapitel vierten Kapitel erläutert. Wir haben uns für dieses Muster entschieden, da unser Team damit mehr Erfahrungen hat. |
| Grundbibliothek | Die Grundbibliothek des Projekts ist Processing. Diese Entscheidung wurde getroffen, weil es für den Prototyp empfohlen wurde und weil es sich für die Teammitglieder verständlich erwies im Vergleich zu anderen Optionen. |

# 10. Qualitätsanforderungen


## 10.1. Qualitätsszenarien

| ID | Szenario |
|---|---|
| Z01 | Das System muss auch nach 20 gespielten Runden genauso schnell und fehlerfrei funktionieren wie in der ersten Runde. |
| Z02 | Das System soll nur gültige Spieleingaben verarbeiten. |
| E01 | Das System soll innerhalb von maximal zwei Sekunden auf das Legen einer RFID-Karte auf den Scanner reagieren. |
| B01 | Die Spieler sollen während des Spielens den Spielmodus und den Schwierigkeitsgrad wählen können. |
| B02 | Die Bedienung und Benutzeroberfläche ermöglichen neuen Spielern nach einer Minute problemlos ein Spiel zu beginnen. |

| ID | Szenario |
| --- | --- |
| W01 | Ein erfahrener Java-Entwickler benötigt maximal zwei Tage, um eine Änderung am Spiel vorzunehmen, einschließlich des Einlesens. |
| W02 | Ein Administrator benötigt maximal zehn Minuten, um die Spielkonfiguration anzupassen. |
| W03 | Ein erfahrener Java-Entwickler benötigt maximal eine Woche, um einen neuen Spielmodus hinzuzufügen. |
| W04 | Die Code-Qualität wird durch Checkstyle sichergestellt. |

# 11. Risiken und technische Schulden

| Was | Warum |
|---|---|
| Hardware-Abhängigkeit | Da das Projekt Pi4J verwendet, besteht das Risiko, dass es von der Hardware, insbesondere dem Raspberry Pi, abhängig ist. Probleme wie Hardwareausfälle oder Inkompatibilitäten könnten die Entwicklung und Bereitstellung beeinträchtigen. |
| Komplexität der Integration | Die Integration von Processing für die Benutzeroberfläche und Pi4J für die Hardwaresteuerung kann komplex sein. Das Risiko von Integrationsfehlern und Inkompatibilitäten zwischen den beiden Technologien ist daher vorhanden. |
| Performance-Probleme | Da Processing und Pi4J intensive Berechnungen und Echtzeitsteuerung erfordern können, besteht das Risiko von Leistungsengpässen oder Verzögerungen, insbesondere wenn die Anwendung auf einem Raspberry Pi mit begrenzten Ressourcen läuft. |
| Abhängigkeit von Drittanbieter-Bibliotheken | Das Projekt könnte auf Drittanbieter-Bibliotheken für bestimmte Funktionen oder Erweiterungen von Processing oder Pi4J angewiesen sein. Das Risiko von Kompatibilitätsproblemen, nicht unterstützten Aktualisierungen oder sogar Ausfällen dieser Bibliotheken ist daher vorhanden. |

# 12. Glossar

In dieser Tabelle werden technische oder fachliche Begriffe aufgeführt, die im Zusammenhang mit dem System verwendet werden, sowie ihre Erklärungen.

| **Begriff** | **Definition** |
|---|---|
| *Java Doc* | *Ein Tool zur Dokumentation von Code, das spezielle Kommentare im Code verwendet, um Glossare mit Begriffen oder Funktionen zu erstellen.* |
| *SAD* | *Abkürzung für Software-Architecture-Documentation. Dieses Dokument beschreibt die Architektur des Systems.* |
| *Processing* | *Processing ist eine Java-Bibliothek, die die Entwicklung visueller Kunst und Kreativprojekte erleichtert.* |
| *Pi4j* | *Pi4J ist eine Java-Bibliothek, die es ermöglicht, Java-Anwendungen auf einem Raspberry Pi zu entwickeln und zu steuern, insbesondere für die Interaktion mit den GPIO (General Purpose Input/Output)-Pins des Raspberry Pi.* |