**ReaFID**

**Ein RFID-Reaktionsspiel auf Basis eines Arduinos**



**Semesterarbeit Désirée Sacher,   
Informatik Teilzeit Studium - ZHAW**

6. /7. Semester 12. September 2014

**Danksagung**

Ich möchte mich in erster Linie bei meinem Lebenspartner Patrick Diezi bedanken, für seine Geduld und Unterstützung in jeder Lage. Er hat mir nicht nur den Rücken gestärkt und mich im Haushalt entlastet, wir konnten auch Design-Entscheide diskutieren und argumentieren, was ich immer als spannend empfinde.

Auch danken möchte ich Alain Lafon für das entgegen gebrachte Vertrauen speziell in der Anfangsphase bis zur Finalisierung der Aufgabenstellung, in Bezug auf die Machbarkeitsanalyse der ursprünglichen Aufgabe und seine Ratschläge, in Bezug auf Entwicklungsumgebung und Toolkits für RFID, Arduino und Python über das gesammte Projekt hinweg.

Valentin Zahnd, Informatik Student in Winterthur, möchte ich danken für das Teilen seiner Erfahrungen mit Python und Beitragen von Hinweisen, welche die Suche nach Problemlösungen um einiges beschleunigt haben.

Zu guter letzt danke ich allen Entwicklern und Entwicklerinnen, welche vor mir schon auf die gleichen Probleme gestossen sind und sie im Internet entsprechend publiziert haben. Ihr habt mein Vorankommen massgeblich unterstützt.

**Inhaltsverzeichnis**

[Verzeichnisse 5](#_Toc398226707)

[Abbildungsverzeichnis 5](#_Toc398226708)

[Tabellenverzeichnis 6](#_Toc398226709)

[Zusammenfassung 7](#_Toc398226710)

[1. Einleitung 8](#_Toc398226711)

[2. Umsetzung des Projekts 9](#_Toc398226712)

[2.1. Aufgabenstellung 9](#_Toc398226713)

[2.2. Arbeitsplanung 9](#_Toc398226714)

[2.3. Projektumsetzung 11](#_Toc398226715)

[3. Definitionen 14](#_Toc398226716)

[3.1. RFID 14](#_Toc398226717)

[3.1.1. RFID-Technologie und Einsatzgebiete 14](#_Toc398226718)

[3.1.2. Mifare Standard 15](#_Toc398226719)

[3.2. Arduino 16](#_Toc398226720)

[3.3. Python 17](#_Toc398226721)

[4. ReaFID – Spiel, Umsetzung und Analyse 18](#_Toc398226722)

[4.1. Das Spiel 18](#_Toc398226723)

[4.2. Ausgangslage 19](#_Toc398226724)

[4.3. Systemarchitektur 19](#_Toc398226725)

[4.4. GUI-Architektur 20](#_Toc398226726)

[4.5. Software Architektur 21](#_Toc398226727)

[4.5.1. Design Patterns 21](#_Toc398226728)

[4.5.1.1. MVC – Model View Control Pattern 21](#_Toc398226729)

[4.5.1.2. Singleton Pattern 22](#_Toc398226730)

[4.5.1.3. Factory Pattern 23](#_Toc398226731)

[4.5.1.4. Strategy Pattern 24](#_Toc398226732)

[4.5.2. Use Cases 25](#_Toc398226733)

[4.5.2.1. Spieldurchlauf 26](#_Toc398226734)

[4.5.2.2. Konfiguration der seriellen Verbindung 27](#_Toc398226735)

[4.5.2.3. Wahl des Spielmodus 29](#_Toc398226736)

[4.6. Implementierungsdetails 32](#_Toc398226737)

[4.6.1. Zeitmessfunktion 32](#_Toc398226738)

[4.6.2. Serielle Verbindung zum Arduino 32](#_Toc398226739)

[4.6.2.1. Implementation Arduino 32](#_Toc398226740)

[4.6.2.2. Implementation Python 33](#_Toc398226741)

[4.6.3. Serialisierung der Kartenkonfiguration 33](#_Toc398226742)

[4.6.4. Entwickeln des GUI 34](#_Toc398226743)

[4.6.4.1. Analyse 34](#_Toc398226744)

[4.6.4.2. Umsetzung 34](#_Toc398226745)

[4.6.5. Entwickeln der Spiellogik 36](#_Toc398226746)

[4.6.5.1. 37](#_Toc398226747)

[4.6.6. Tests 38](#_Toc398226748)

[4.7. Fazit 40](#_Toc398226749)

[4.8. Ausblick 41](#_Toc398226750)

[5. Diskussion des eignen Beitrags 42](#_Toc398226751)

[5.1. Studieren der RFID Technologie und Bauen eines Reader mittels eines Arduinos und einem RFID Shield 42](#_Toc398226752)

[5.2. Dokumentieren von Herausforderungen beim Bauen eines RFID Readers 42](#_Toc398226753)

[5.3. Machbarkeitsanalyse zum Auslesen unserer neuen Campus Card 43](#_Toc398226754)

[5.4. Spiel in Python und mit einem Arduino 45](#_Toc398226755)

[6. Schlussfolgerungen 46](#_Toc398226756)

[7. Literaturverzeichnis 47](#_Toc398226757)

[8. Anhang 48](#_Toc398226758)

[8.1. Glossar 48](#_Toc398226759)

[8.2. Sprint Reports 49](#_Toc398226760)

[8.2.1. Sprint 1 49](#_Toc398226761)

[8.2.2. Sprint 2 49](#_Toc398226762)

[8.2.3. Sprint 3 50](#_Toc398226763)

[8.2.4. Sprint 4 50](#_Toc398226764)

[8.2.5. Sprint 5 51](#_Toc398226765)

# Verzeichnisse

## Abbildungsverzeichnis

[Abbildung 1: Aufgabenstellung 9](#_Toc398224574)

[Abbildung 2: Übersicht RFID-Typen (aus RFID Handbuch - Finkenzeller, Abbildung 2.18) 14](#_Toc398224575)

[Abbildung 3: RFID-Kommunikationsprotokoll nach ISO 14443 15](#_Toc398224576)

[Abbildung 4: Arduino UNO Version 3 16](#_Toc398224577)

[Abbildung 5: Arduino UNO mit einem XBee RFID Shield 17](#_Toc398224578)

[Abbildung 6: Spielhauptfenster nach Start mit zufällig gewählter Karte 18](#_Toc398224579)

[Abbildung 7: GUI Hauptfenster Beschreibung 21](#_Toc398224580)

[Abbildung 8: GUI-Konfiguration der seriellen Verbindung 21](#_Toc398224581)

[Abbildung 9: MVC-Dateistruktur des Python-Programms 22](#_Toc398224582)

[Abbildung 10: UML-Klassendiagramm Singleton und dessen Anwenderklassen 23](#_Toc398224583)

[Abbildung 11: UML-Klassendiagramm der Card Factory 24](#_Toc398224584)

[Abbildung 12: UML-Klassendiagramm des umgesetzten Strategy Pattern 25](#_Toc398224585)

[Abbildung 13: Flussdiagramm Spieldurchlauf 26](#_Toc398224586)

[Abbildung 14: UML Sequenzdiagramm Spieldurchlauf 27](#_Toc398224587)

[Abbildung 15: Flussdiagrammzur Konfiguration der Seriellen Verbindung 28](#_Toc398224588)

[Abbildung 16: UML Sequenzdiagramm Serielle Interface Aktualisierung 29](#_Toc398224589)

[Abbildung 17: GamePlayMode-Dialog 29](#_Toc398224590)

[Abbildung 18: Flussdiagramm zur Wahl des Spielmodus 30](#_Toc398224591)

[Abbildung 19: Sequenzdiagramm Spielmodus Dialog 31](#_Toc398224592)

[Abbildung 20: UML-Klassendiagramm „TimeMeasure 32](#_Toc398224593)

[Abbildung 21: UML-Klassendiagramm RFIDReader 33](#_Toc398224594)

[Abbildung 22: Source Code Ausschnitt "pickle" 34](#_Toc398224595)

[Abbildung 23: UML-Klassendiagramm MainGUI 34](#_Toc398224596)

[Abbildung 24: UML-Klassendiagramme der Dialog GUIs 35](#_Toc398224597)

[Abbildung 25: Source Code zur Methode play() der Klasse GameStrategyEasy 36](#_Toc398224598)

[Abbildung 26: Source Code des GamePlayManagers 38](#_Toc398224599)

[Abbildung 27: Test Klasse des Card Manager 39](#_Toc398224600)

[Abbildung 28: Ausschnitt aus der Dokumentation zum Seeed Studio Reader 42](#_Toc398224601)

[Abbildung 29: Success Story InterCard - ZHAW 43](#_Toc398224602)

[Abbildung 30: Kartentypen im Angebot von InterCard 44](#_Toc398224603)

[Abbildung 31: Auszug aus dem technischen Prospekt der InterCard 45](#_Toc398224604)

## Tabellenverzeichnis

[Tabelle 1: Arbeitsschritte des Phase 1 des Projekts 9](#_Toc398224705)

[Tabelle 2: Arbeitsschritte der Phase 2 des Projekts 10](#_Toc398224706)

[Tabelle 3: Projekttermine 11](#_Toc398224707)

[Tabelle 4: Tatsächliche Aufwände Phase 1 11](#_Toc398224708)

[Tabelle 5: Tatsächliche Aufwände Phase 2 12](#_Toc398224709)

[Tabelle 6: Während des Sprints erstellte Stories 13](#_Toc398224710)

[Tabelle 7: Summe der Aufwände 13](#_Toc398224711)

[Tabelle 8: Weiterentwicklungsmöglichkeiten 41](#_Toc398224712)

[Tabelle 9: Literaturverzeichnis 47](#_Toc398224713)

[Tabelle 6: Glossar 48](#_Toc398224714)

# Zusammenfassung

In dieser Arbeit sollte ein Projekt mit Hilfe eines Arduinos und einem RFID realisiert und erste Erfahrungen mit Python gesammelt werden. Das primäre Ziel war dabei eine Machbarkeitsanalyse zum Thema Sicherheit der Campus Card, als Alternativziel wurde allerdings gleich zu Beginn ein Spiel auf Basis der Technologien in Betracht gezogen.

Realisiert wurde ein Spiel, bei welchem mit Hilfe des RFID-Sensors die Reaktionsgeschwindigkeit des Spielers getestet wird. Für die Programmierung wurden diverse Design Patterns verwendet und implementiert.

Diese Arbeit enthält Erläuterungen zu den eingesetzten Technologien, welche dem technischen Verständnis dienen. Überlegungen und Design Entscheide zur Architektur der Software sind genauso festgehalten worden, da schon von Anfang an die mögliche Erweiterungsfähigkeit berücksichtigt und diese entsprechend priorisiert wurden. Herausforderungen und Schwiergkeiten, welche meist technischer Natur waren und mit Hilfe von Suchmaschinen und vertiefter Einarbeitung gelöst werden konnten runden die Arbeit ab.

Zum Abschluss wird die konkreten Aufgabenstellungen und die Erreichung der Ziele diskutiert, wobei nach dem DRY[[1]](#footnote-1)-Prinzip zum Teil auf die entsprechenden Kapitel verwiesen wird.

# Einleitung

Das ursprüngliche Ziel dieser Arbeit war es, Einblicke zu erhalten, wie gut unsere Studentenausweise abgesichert sind. Durch administrative Einschränkungen wurde in der Anfangsphase des Projekts das Ziel abgeändert und stattdessen ein Reaktionsspiel mit Hilfe eines RFID Readers, einem Arduino und der Programmiersprache Python implementiert. Alle drei Komponenten waren für mich neue Technologien für welche Interesse bestand, sich einzuarbeiten.

Diese Arbeit diente als Möglichkeit, sich mit der RFID-Technologie auseinanderzusetzen. Da die RFID-Technologie, speziell der in Europa verbreitete Mifare-Standard, in den letzten Jahren häufig an Hacker-Konferenzen thematisiert wurde, sind dessen Schwachstellen in der Impementation ausreichend bekannt.

Ursprünglich war geplant, die von InterCard hergestellten Studentenausweise der ZHAW genauer zu analysieren. Allerdings wurde diesem Vorhaben von den Sicherheitsbeauftragen der ZHAW einige Hürden in Form von NDAs in den Weg gestellt, welche eine sinnvolle Auseinandersetzung inklusive der Dokumentation der Befunde erschwert hätten.

Nach Absprache mit dem Betreuer der Semesterarbeit wurde auf Basis von einem Arduino und einem RFID Shield ein Spiel entwickelt, welches die Technologie kreativ nutzt. Der ursprüngliche Fokus von „Hacken“ und „Reverse Engineering“ der Technologie wurde verlegt auf das Entwerfen und Programmieren eines Spiels in einer bis dato dem Autor unbekannten Sprache. Da die RFID-Technologie in beiden Projektzielen verwendet wird, konnte trotzdem das Wissen über die Technologie angeeignet und entwickelt werden.

Das Spielkonzept wurde selbst erfunden und entwickelt um die RFID Technologie zu anderen Zwecken als zum verfolgen von Gegenständen zu benutzen.

# Umsetzung des Projekts

## Aufgabenstellung

Die Aufgabenstellung wurde im EBS wie folgt definiert:

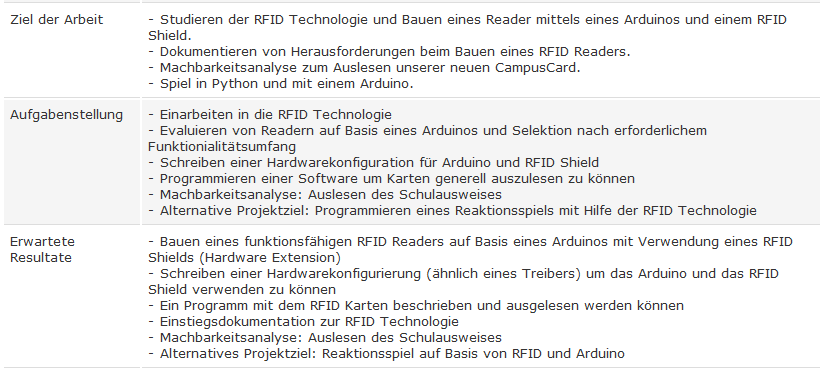


Abbildung 1: Aufgabenstellung

Da schon zu Beginn das Risiko abgewogen werden musste, dass das „Auslesen des Schulausweises“ nicht umgesetzt werden kann, wurde auch von Anfang an ein Alternatives Projektziel definiert. Viele der ursprünglichen Ziele konnten aber während des Projekts trotzdem erfüllt werden.

## Arbeitsplanung

In der ersten Phase, welche in der Tabelle 1 dargestellt ist, wurden die Arbeiten in Scrum Stories aufgeteilt und der Aufwand geschätzt. In dieser Phase konnte auch geklärt werden, dass das ursprüngliche Ziel (Auslesen eines Schulausweises) aus rechtlichen Gründen nicht sinnvoll als Semesterarbeit umsetzbar ist. Am Design Review wurde entsprechend entschieden, dass stattdessen das alternative Projektziel „Programmieren eines Reaktionsspiels mit Hilfe der RFID-Technologie“ verfolgt wird. Aus diesem Grund musste anschliessend die Arbeitsplanung für die verbleibende Zeit angepasst werden. Die Planung für die zweite Phase ist in der Tabelle 2 zu finden.

Bis zum Projektende hin wurden drei Sprints (Nummer 3 – 5) für die Umsetzung vorgesehen und die Arbeitsschritte auf diese verteilt. Stories welche nach Abschätzung des Zeitaufwandes von der Priorität herunter gestuft wurden, wurden im Backlog festgehalten. Die Aufwände wurden dabei jeweils in 2-Stundenblöcke (analog zur Pomodoro-Technik) unterteilt und über die zur Verfügung stehenden Wochen der Semesterarbeit verteilt. Die Sprint-Planung und der Fortschritt wurde mittels JIRA Agile durchgeführt und festgehalten

Tabelle 1: Arbeitsschritte des Phase 1 des Projekts

|  |  |  |  |
| --- | --- | --- | --- |
| **Sprint** | **JIRA ID** | **Arbeitsschritt** | **Aufwand in Stunden** |
| 1 | RFID-1 | Projekt aufsetzen | 3 |
| 1 | RFID-2 | Lernen von Python (Code Academy) | 8 |
| 1 | RFID-3 | Einlesen in RFID-Technologie | 8 |
| 2 | RFID-4 | Unterschiede der Arduino RFID Reader ausfindig machen | 6 |
| 2 | RFID-5 | Bestimmen des genauen Kartentyps | 4 |
| 2 | RFID-6 | Erstellen des Skripts zum Auslesen | 12 |

Tabelle 2: Arbeitsschritte der Phase 2 des Projekts

|  |  |  |  |
| --- | --- | --- | --- |
| **Sprint** | **JIRA ID** | **Arbeitsschritt** | **Aufwand in Stunden** |
| 3 | RFID-7 | ReaFID Funktion – Konfigurieren von Karten | 8 |
| 3 | RFID-8 | ReaFID Funktion – Lesen von Karten | 8 |
| 3 | RFID-9 | ReaFID Spielfunktion – Zeitstoppen | 5 |
| 3 | RFID-12 | ReaFID Dokumentation 1 | 10 |
| 3 | RFID-20 | ReaFID Projektmanagement / Admin 1 | 1 |
| 4 | RFID-10 | ReaFID Spiel-GUI – Spieloberfläche Hauptfenster | 15 |
| 4 | RFID-11 | ReaFID Spiel-GUI – Abbrechen / Stoppen Spiel | 5 |
| 4 | RFID-13 | ReaFID Dokumentation 2 | 10 |
| 4 | RFID-21 | ReaFID Projektmanagement / Admin 2 | 2 |
| 5 | RFID-14 | ReaFID Dokumentation 3 | 20 |
| 5 | RFID-22 | ReaFID Projektmanagement / Admin 3 | 1 |
| Backlog | RFID-15 | ReaFID Spiel-GUI – Top Scorer-Liste | 8 |
| Backlog | RFID-16 | ReaFID Spiel-GUI – Spieloberfläche Kartenkonfiguration | 12 |
| Backlog | RFID-17 | ReaFID Spiel-GUI – Anzeige Auslesen Karte | 6 |
| Backlog | RFID-18 | ReaFID Spiel-GUI – Menüpunkt Select Modus | 4 |
| Backlog | RFID-19 | ReaFID Spiel - Advanced Mode erstellen | 20 |

## Projektumsetzung

Für das Projekt galten folgende Termine, wobei das Kick Off, das Design Review, der Abgabetermin und der Präsentationstermin für Semesterarbeiten vorgegebene Termine sind.

Tabelle 3: Projekttermine

|  |  |  |
| --- | --- | --- |
| **Nr.** | **Termin** | **Beschreibung** |
| **1** | 17.02.14 – 09.03.14 | Sprint 1 |
| **2** | 24.03.14 | Kick Off |
| **3** | 17.02.14 – 09.03.14 | Sprint 1 |
| **3** | 29.04.14 | Präsentationstermin Security BSides London 2014 |
| **4** | 11.07.14 | Design Review |
| **5** | 14.07.14 – 03.08.14 | Sprint 3 |
| **6** | 04.08.14 – 24.08.14 | Sprint 4 |
| **7** | 25.08.14 – 12.09.14 | Sprint 5 |
| **8** | 12.09.14 | Abgabetermin |
| **9** | 05.11.14 | Präsentationstermin |

Zwischen Sprint 2 und 3 wurden nicht nur die rechtlichen Abklärungen getätigt, sondern auch ein 15- minütiger Vortrag für die Security BSides-Konferenz in London als „Rookie Track“[[2]](#footnote-2) vorbereitet. Da die Arbeiten zum Vortrag nur die Erkenntnisse aus den technischen Abklärungen zu RFID und Arduino verwendeten und aufbereiteten und nicht direkt für die Semesterarbeit waren, wurden diese nicht in JIRA Agile festgehalten und sind entsprechend nicht in der Projektplanung berücksichtigt. Die Informationen aus jener Konferenzpräsentation wurden allerdings in dieser Semesterarbeit miteinbezogen und bereicherten entsprechend die Story „Dokumentation“.

Tatsächlich wurden für die Stories die in der Tabelle 4 und 5 folgenden Aufwände verwendet.

Tabelle 4: Tatsächliche Aufwände Phase 1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Sprint** | **JIRA ID** | **Arbeitsschritt** | **Aufwand in Stunden geschätzt** | **Aufwand in Stunden tatsächlich** |
| 1 | RFID-1 | Projekt aufsetzen | 3 | 3 |
| 1 | RFID-2 | Lernen von Python (Code Academy) | 8 | 5 |
| 1 | RFID-3 | Einlesen in RFID-Technologie | 8 | 5 |
| 2 | RFID-4 | Unterschiede der Arduino RFID Reader ausfindig machen | 6 | 6 |
| 2 | RFID-5 | Bestimmen des genauen Kartentyps | 4 | 8.5 |
| 2 | RFID-6 | Erstellen des Skripts zum Auslesen | 12 | 12 |

Total geschätzt: 41; Aufwand: 39.5

Tabelle 5: Tatsächliche Aufwände Phase 2

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Sprint** | **JIRA ID** | **Arbeitsschritt** | **Aufwand in Stunden geschätzt** | **Aufwand in Stunden tatsächlich** |
| 3 | RFID-7 | ReaFID Funktion – Konfigurieren von Karten | 8 | 8 |
| 3 | RFID-8 | ReaFID Funktion – Lesen von Karten | 8 | 8 |
| 3 | RFID-9 | ReaFID Spielfunktion – Zeitstoppen | 5 | 3 |
| 3 | RFID-12 | ReaFID Dokumentation 1 | 10 | 10 |
| 3 | RFID-20 | ReaFID Projektmanagement / Admin 1 | 1 | 1 |
| 4 | RFID-10 | ReaFID Spiel-GUI – Spieloberfläche Hauptfenster | 15 | 12 |
| 4 | RFID-11 | ReaFID Spiel-GUI – Abbrechen / Stoppen Spiel | 5 | 0 |
| 4 | RFID-13 | ReaFID Dokumentation 2 | 10 | 4 |
| 4 | RFID-21 | ReaFID Projektmanagement / Admin 2 | 2 | 2 |
| 5 | RFID-14 | ReaFID Dokumentation 3 | 20 | 33 |
| 5 | RFID-22 | ReaFID Projektmanagement / Admin 3 | 1 | 1 |
| Backlog | RFID-15 | ReaFID Spiel-GUI – Top Scorer-Liste | 8 | 0 |
| Backlog | RFID-16 | ReaFID Spiel-GUI – Spieloberfläche Kartenkonfiguration | 12 | 6 |
| Backlog | RFID-17 | ReaFID Spiel-GUI – Anzeige Auslesen Karte | 6 | 0 |
| Backlog | RFID-18 | ReaFID Spiel-GUI – Menüpunkt Select Modus | 4 | 4 |
| Backlog | RFID-19 | ReaFID Spiel - Advanced Mode erstellen | 20 | 0 |

Folgende User Story wurde zudem erst während dem Sprint definiert, da erkannt wurde dass diese bei der Planung nicht berücksichtigt wurde:

Tabelle 6: Während des Sprints erstellte Stories

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Sprint** | **JIRA ID** | **Arbeitsschritt** | **Aufwand in Stunden geschätzt** | **Aufwand in Stunden tatsächlich** |
| 3 | RFID-23 | Game Play Mode Programming | 8 | 8 |

Total geschätzt 85; Aufwand: 100

Damit wurden folgende Aufwände tatsächlich verwendet:

Tabelle 7: Summe der Aufwände

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Sprint** | **JIRA ID** | **Arbeitsschritt** | **Aufwand in Stunden geschätzt** | **Aufwand in Stunden tatsächlich** |
| Phase 1 des Projekts | | | 41 | 39.5 |
| Phase 2 des Projekts | | | 85 | 100 |

Einsicht in die Burndown Charts des Projekts wird im Anhang gegeben.

# Definitionen

## RFID

### RFID-Technologie und Einsatzgebiete

RFID steht für „radio-frequency identification“, was einer „Identifizierung mit Hilfe elektromagnetischer Wellen“ entspricht. Die Imlpementationen basieren dabei auf „Low Frequency“ (9 kHz bis 135 kHz), „High Frequency“ (6,78 MHz, 13,56 MHz, 27.125 MHz, 40,680 MHz) oder „Ultra High Frequency“ (Europa: 865-869 MHz, USA/Asien: 950 MHz). Während frühere Implementationen (zum Beispiel für Zutrittssysteme von Gebäuden, Hotelzimmern oder zur Verfolgung von Gegenständen) mit Low Frequency durchgeführt wurden, hat in den letzten Jahren die High Frequency einiges an Verbreitung gewonnen. Ultra High Frequency wird unter anderem auch bei Skilift Ausweisen oder für das verfolgen bei Sportveranstaltungen verwendet, ist allerdings nicht global standardisiert.

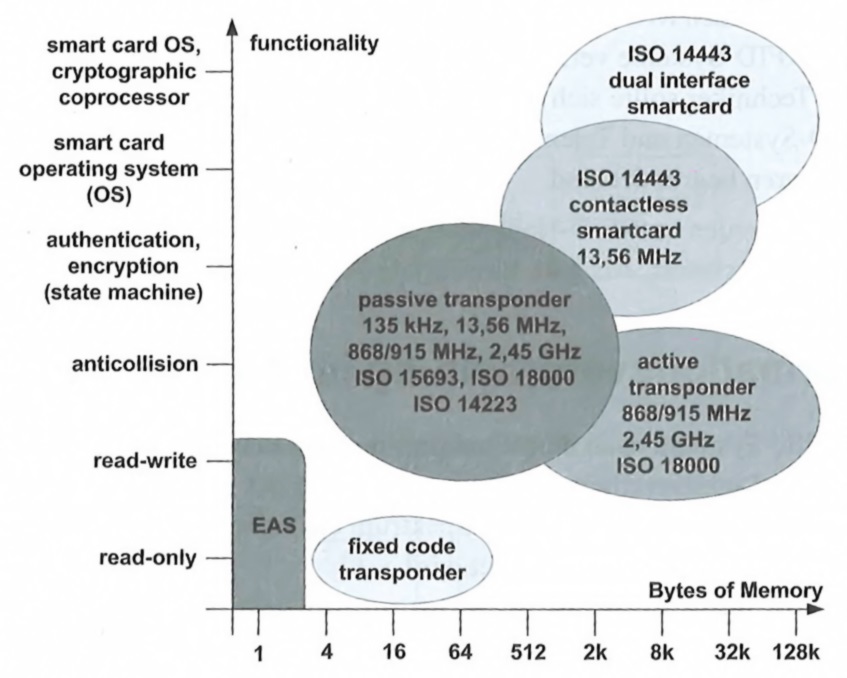


Abbildung 2: Übersicht RFID-Typen (aus RFID Handbuch - Finkenzeller, Abbildung 2.18)

In den meisten Systemen wird ein aktives Lesegerät und ein passiver Tag verwendet. Sobald der Tag in die Nähe des Lesegeräts gelangt, werden elektromagnetische Wellen erzeugt und vom Lesegerät verarbeitet.. Das Protokoll der Übertragung zwischen Lesegerät und Tag wird in diversen Standards festgelegt, welche inzwischen zum Teil ISO genormt sind. Hersteller verwenden allerdings aus „Sicherheitsgründen“, wobei es sich dabei mehr um „Security by Obscurity“ handelt, oft eigene Implementationsvarianten.

Durch eine zielgerichtete Recherche betreffend des Studentenausweises konnte einfach festgestellt werden, dass es sich bei dem in der ZHAW verwendeten System um ein nach ISO 14443 auf der Frequenz 13,56 MHz festgelegten Standard handelt, wobei sich die ZHAW für die Implementation ohne Chip, also dem „Mifare Classik 1k“ Standard, entschieden hatte.

Implementationen mit Chip sind einiges Leistungsfähiger als die ohne. So können zum Beispiel RFID-Kreditkarten mit Chip für jede Transaktion einen neuen CVV Code generieren. Dies ist einer der Hauptgründe, warum das RFID-Kreditkartensystem mit „Chip und Pin“, so wie es in Europa verbreitet ist, einiges sicherer ist als das in den USA verbreitete ohne Chip.

### Mifare Standard

Die Tags welche im Mifare Standard zum Einsatz kommen, sind in mehreren Formen erhältlich.. Für die Studentenausweise wird zum Beispiel die Form einer Karte verwendet.

Die Norm für 1k Tags besagt, dass ein Tag 16 Sektoren hat, wobei jeder Sektor vier Blöcke enthält. Jeweils der letzte Block jedes Sektors enthält den „Access Key“, welcher für verschlüsselte Sektoren nötig ist. Pro Sektor kann es maximal zwei verschiedene Access Keys geben, wobei diese unterschiedliche Zugriffsberechtigungen besitzen können. Der erste Block (Block 0) enthält die UID der Karte und ist normalerweise schreibgeschützt. Karten mit veränderbaren UIDs sind relativ einfach über das Internet erhältlich. Falls ein Sektor verschlüsselt ist, wurde dafür die Crypto1 Verschlüsselung verwendet. Informationen zu dieser Verschlüsselung wurden schon vor Jahren am CCC präsentiert, inklusive der Information, wie die verschlüsselten Inhalte trotzdem ausgelesen werden können.

Die Authentisierung, welche per Sektor möglich ist, findet nach folgendem Schema statt:

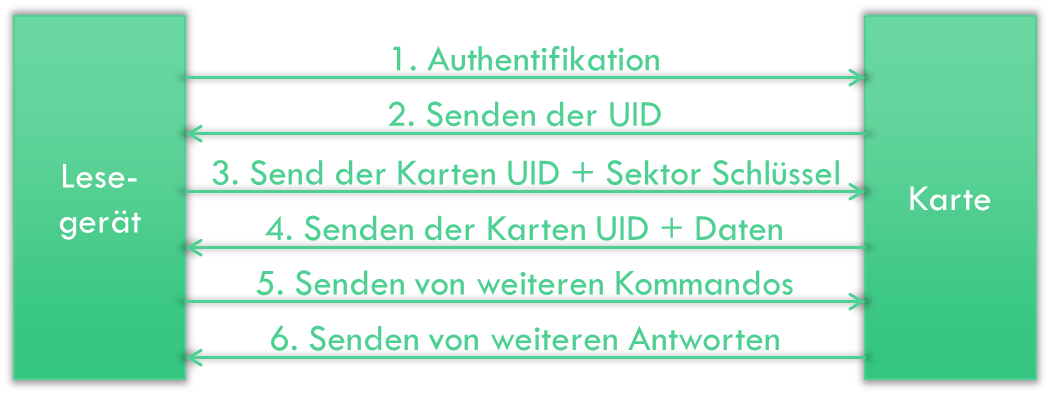


Abbildung 3: RFID-Kommunikationsprotokoll nach ISO 14443

Das Lesegerät sendet zur Protokolleröffnung den Befehl zum Lesen eines Sektors und gibt den Sektor an. Die Karte antwortet mit der UID, welche danach jeweils zur Identifikation weiter verwendet wird. Da diese Implementation auf dem Reader selber statt findet, wurden diese Protokoll vom Arduino und nicht vom Spielcode ausgeführt.

Da es sich um eine drahtlose Kommunikation handelt, ist ein Angriff über Mithören der Übertragung und danach durch separate Abfrage möglich. Zudem können Karten geklont werden. Da die Verschlüsselung in Sicherheitskreisen als unsicher gilt, sind die darauf abgelegten Informationen schlecht geschützt. Häufig wird aus diesem Grund bei Implementationen auf das Abspeichern auf der Karte verzichtet und stattdessen jegliche Informationen auf dem „Backend System“ gespeichert. Bei der Prüfung der Karte wird lediglich die UID abgerufen und geprüft, ob diese im System konfiguriert ist. Dementsprechend müssen jegliche Kontrollmechanismen auf dem Backend System implementiert sein.

## Arduino

Ein Arduino ist eine Open Source Computerplattform, welche auf einem simplen Mikrokontroller (Atmel ATMEGA8) basiert und eine dazugehörige Entwicklungsumgebung zur Programmierung mitbringt. Der Mikrokontroller kann Programme in Form einer daueraktiven State Machine verarbeiten und wird deshalb oft für die Zusammenarbeit mit Sensoren jeglicher Art verwendet.

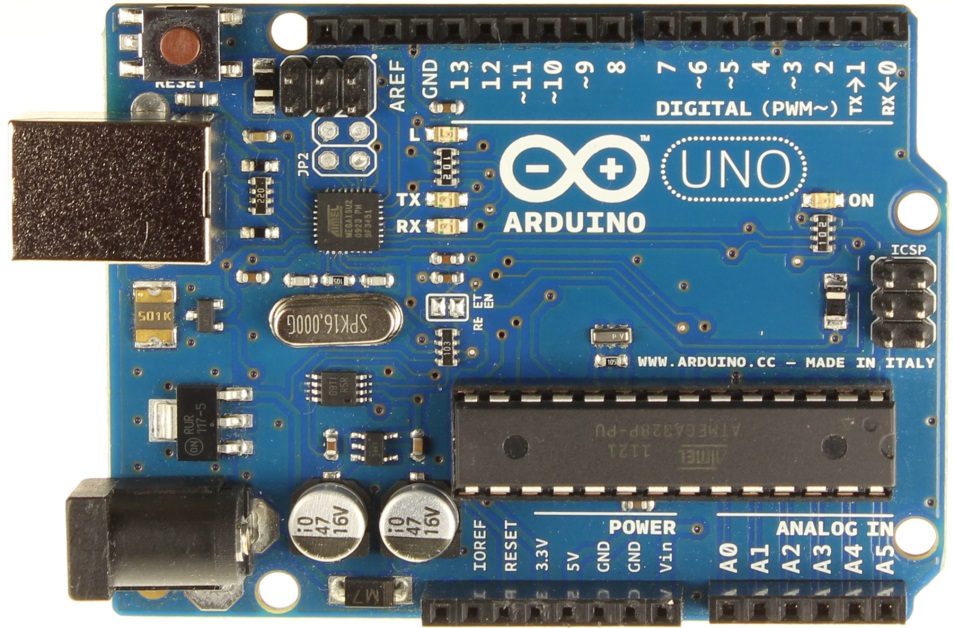


Abbildung 4: Arduino UNO Version 3

Um die Verknüpfung mit diesen Sensoren (welche generell als Input Systeme verstanden werden können) zu optimieren und Installationen weniger anfällig zu machen, wurden einige „Boards“ oder „Shields“ entwickelt, welche direkt auf das Arduino aufgesetzt werden können. Sie beanspruchen gewöhnlicherweise mehr Schnittstellen, machen die Installationen aber auch ein wenig robuster.

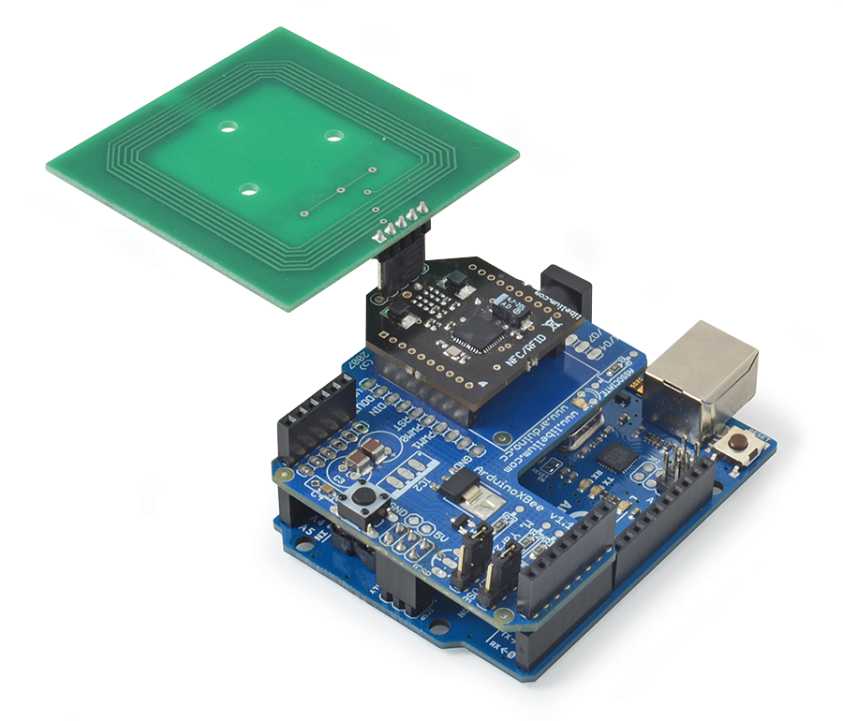


Abbildung 5: Arduino UNO mit einem XBee RFID Shield

Es gibt heute eine Vielzahl an Shields, welche von WLAN Interfaces, über RFID/NFC Adaptern bis zu Feuchtigkeitsmesser reichen. Falls kein Shield vorhanden ist, können die Sensoren und Aktoren gewöhnlicher Weise direkt oder mit Hilfe eines Breadboards angeschlossen und angesprochen werden. Starter Kits für Arduinos liefern nicht nur eine Vielzahl solcher Sensoren und Aktoren mit, sondern beinhalten auch detaillierte Anleitungen. Durch eine inzwischen grossen Community werden nicht nur ständig neue Sensoren entwickelt, es finden auch verbreitet Workshops für Anfänger statt und Anleitungen sind für viele Projektideen im Internet zu finden.,

Der Output findet normalerweise über Dioden, den Computerbildschirm (über das serielle Interface) oder einem direkt angehängten Bildschirm statt.

## Python

Python ist eine relativ moderne Programmiersprache (Erscheinungsjahr 1991) welche entsprechend der Entwurfsphilosophie stark auf die Programmlesbarkeit und Einfachheit der Syntax setzt. In den letzten Jahren fand sie starke Verbreitung durch Online-Lernprogramme wie von Codeacademy und durch den omnipräsenten Einsatz bei Google sowie bei Open Source-Projekten.

Die Sprache unterstützt mehrere Programmierparadigmen, unter anderem auch den Objekt-orientierten und den strukturierten Ansatz. Durch diese Vielfältigkeit, kann ein zur Aufgabe passendes Paradigma ausgewählt werden, ohne dass neue Syntax erlernt werden müssen.

Zusätzlich ist es hilfreich, dass für Python eine grosse Auswahl an Bibliotheken und Entwicklungsumgebungen existieren, dies kann Entwicklungen erheblich beschleunigen da auf bestehende Projekte aufgebaut werden kann.

# ReaFID – Spiel, Umsetzung und Analyse

## Das Spiel

Die Idee des Spieles ist es, auf dem Computer Informationen zu einer zufällig gewählten Karte einzublenden, wobei daraufhin vom Spieler so schnell wie möglich die korrekte Karte aus den zur Verfügung stehenden farbigen Karten herausgesucht werden und an den RFID Sensor hingehalten werden muss. Das Spiel prüft die hingehaltene Karte und zählt die benötigte Zeit, wobei falsche Karten Strafzeiten ergeben. Nach 10 Durchläufen wird die benötigte Zeit addiert und als Endezeit ausgegeben. Ziel ist es, so wenig Zeit wie möglich zu verwenden um die jeweils aufgerufene Karte zu identifizieren. Anschliessend kann ein neuer Durchgang gestartet werden.

Durch die Erweiterung um eine Top Scorer-Liste kann zusätzlich die Motivation erhöht werden, in dem frühere Zeiten oder Zeiten von Kollegen unterboten werden sollen.

Für das Spiel müssen die zu verwendenden Karten zuerst konfiguriert und einer Farbe zugeordnet werden. Jegliche im Spiel zu verwendende Karte muss vorab definiert im Spiel angelegt werden. RFID-Karten werden dazu nicht verändert, lediglich die UID wird ausgelesen.

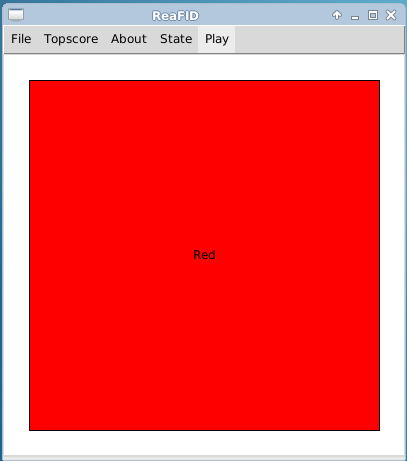


Abbildung 6: Spielhauptfenster nach Start mit zufällig gewählter Karte

Der Name des Spiels ist ein Kunstwort aus der Kombination Reaction und RFID.

## Ausgangslage

Zu Beginn der Semesterarbeit hatte bereits ein initiales Kennenlernen betreffend Arduino und RFID stattgefunden. Um allerdings die Technologien weiter kennen zu lernen, war aus mangelnder Freizeit und existierendem Interesse ein Projekt nötig. Die Sprache Python wurde ausgewählt, da sie vielseitig einsetzbar ist und in Zukunft damit schnell weitere Projektideen umgesetzt werden sollen.

Die Entwicklung wurde auf einer Arch Linux Installation durchgeführt, wobei als IDE Eclipse eingesetzt wurde. Zur Interpretation von Python 3.4 wurde das Plugin PyDev verwendet. Das Arduino wurde über die dazu gelieferte IDE programmiert.

Wenn das Arduino gestartet wird, aktiviert sich der darauf abgespeicherte Code automatisch. Zuerst wird die setup() Methode durchgeführt, danach wird der loop() ausgeführt, wobei sich dessen Verhalten ab diesem Zeitpunkt nur noch über Sensoren oder die Serielle Verbindung beeinflussen lässt.. Die Kommunikation zwischen RFID Reader und Python findet entsprechend auch über den seriellen Anschluss statt.

Zum Thema RFID und Arduino sind im Internet bereits einige Anleitungen und Code-Beispiele zu finden. Auf der Basis der Anleitung von Cooking Hacks[[3]](#footnote-3) wurden die Hardwarekomponenten bestellt. Die Beispiel-Skripte konnten für das Auslesen von leeren RFID-Karten verwendet werden. Um den Zugriff für das Spiel allerdings zu optimieren, wurden die Skripte angepasst.

Da es sich bei dem Projekt um ein Reaktionsspiel handelt, ist der Faktor Zeit eine wichtige Komponente. Für das Spiel wurde entsprechend das Auslesen optimiert und der Handshake wird nach Auslesen der Karten-UID beendet. Dementsprechend wird zu keinem Zeitpunkt versucht auf verschlüsselten Sektoren zuzugreifen.

Ein Verändern der RFID-Karten oder der darauf geschriebenen Daten findet für das Spiel nicht statt. Dies ermöglicht es, die Auswahl von Spielkarten beliebig zu erweitern und zum Beispiel mit verschiedenen Studentenausweisen das Spiel durchzuführen. Zur Zeit sind lediglich die Spielvarianten mit Farbkarten ausprogrammiert, der Kartenmanager ist für Studentenausweise mit ausgelegt, allerdings ist dazu noch keine Spielvariante vorhanden. Weitere Kategorien und Spielvarianten sind beliebig erweiterbar.

Für dieses Projekt wurde mit der neuester Version von Python (Version 3.4) entwickelt. Durch den grossen Versionssprung zwischen Version 2 und 3 und den sich damit ergebenden Änderungen sind nicht mehr alle GUI Toolkits mit der aktuellen Version kompatibel. Dies musste entsprechend bei der weiteren Auswahl von Bibliotheken und Anleitungen berücksichtigt werden. Für die GUI Umgebung wurde aus diesem Grund auf die Bibliothek TkInter gesetzt.

## Systemarchitektur

Um das Spiel zu Spielen sind mehrere Systemkomponenten nötig:



Um RFID-Karten zu lesen wurde für die Projektimplementation ein Arduino mit einem entsprechenden Modul verwendet. Auf der linken Seite sind die technischen Komponenten gezeigt, welche nötig sind um RFID mit RFID Karten zu kommunizieren. Auf der rechten Seite ist der Computer, welcher dem Spieler auch als Anzeige dient, sowie das Python Spiel, welches die Anzeige steuert.

Das Arduino übernimmt die Steuerung des RFID-Sensors, welcher mit Hilfe eines Communication Shields (welches in unserem Fall Produktabhängig ist und nicht für jeden RFID Sensor benötigt wird) angeschlossen ist. Zudem kommuniziert das Arduino die gelesenen UIDs über die serielle Verbindung an den Computer.

Auf dem Computer wird bei Start des Spiels der serielle Port geöffnet und während des Spiels auf vom Arduino gelesene und über die serielle Verbindung übertragene Informationen gehorcht. Bei Erhalt eines Signals reagiert das Python-Programm auf vordefinierte Weise und lässt das Spiel eine entsprechende Aktion ausführen.

## GUI-Architektur

Das GUI besteht aus insgesamt zurzeit aus vier implementierten Fenstern. Zum Spielstart wird das Hauptfenster „ReaFID“(implementiert im „MainGUI“) geöffnet. Aus diesem Fenster können mittels „File“-Menü drei Dialogfenster geöffnet werden, in welchen Konfigurationseinstellungen zum Spiel vorgenommen werden können. Das Topscore Menü ist für die zukünftige Implementation einer Topscore Liste geplant, in welcher die Bestzeiten enthalten sind. Das „About“ Menü soll in naher Zukunft ein kleines ReadMe enthalten, welches Auskunft über dieses Spiel gibt. Das State Label soll während des Spiels anzeigen ob die serielle Verbindung geglückt ist. Über „Play“ kann das Spiel gestartet werden.

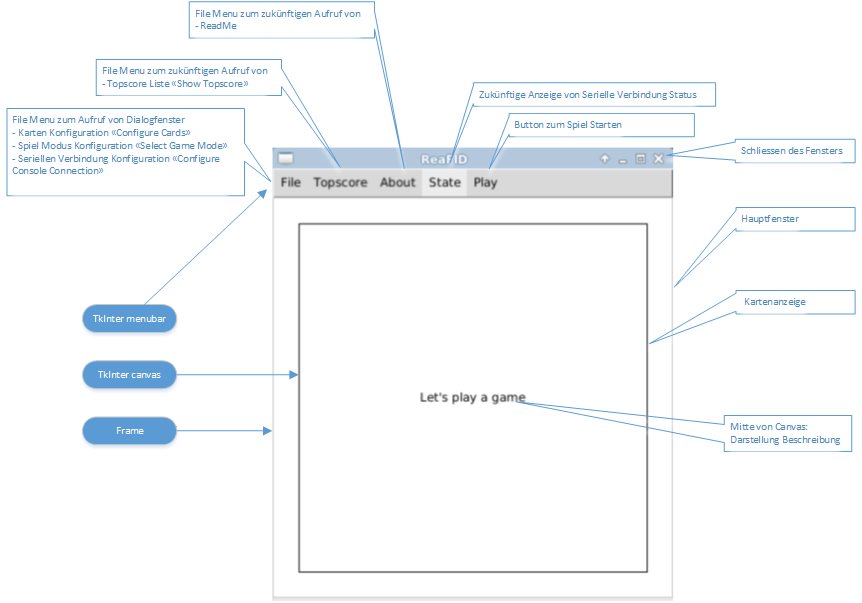


Abbildung 7: GUI Hauptfenster Beschreibung

Die Dialogfenster sind jeweils simpel gestaltet und zeigen nur die nötigen Felder für ihre Anpassungen. Abbildung 8 ist beispielweise das Fenster zur Konfiguration der seriellen Verbindung.

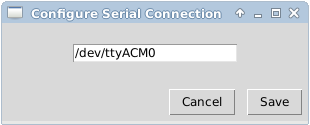


Abbildung 8: GUI-Konfiguration der seriellen Verbindung

## Software Architektur

### Design Patterns

In diesem Projekt kamen einige Designe Patterns zum Zuge, welche im folgenden erläutert werden.

#### MVC – Model View Control Pattern

Als Grundstruktur wurde das MVC Pattern verwendet. MVC hilft durch Kapselung eine saubere Trennung von Anzeige, Modell- und Kontrolllogik umzusetzen. Heute wird es allerdings oft nicht mehr in seiner Originalform verwendet, da die Control-Komponente zu künstlichen Kapselungen führen kann, welche keine Effizienzsteigerungen bringen[[4]](#footnote-4). Für die Implementation dieses Spiels war ein Controller aber durchaus sinnvoll genug, weshalb die Kapselung konsequent umgesetzt wurde.

Model, View und Controller wurden je als Paket realisiert, was sich auch in der Filestruktur darlegte und diese Klasse entsprechend zusammenfasste.

|  |
| --- |
| src/  | --- model/  | --- --- \_\_init\_\_.py  | --- --- cardFactory.py  | --- --- cardManager.py  | --- --- gameLogics.py  | --- --- gamePlayModes.py  | --- --- rFIDReader.py  | --- --- singleton.py  | --- --- timeMeasure.py  | --- view/  | --- --- \_\_init\_\_.py  | --- --- mainGUI.py  | --- --- subGUIs.py  | --- controller/  | --- --- \_\_init\_\_.py  | --- --- gameController.py  | --- --- starter.py |

Abbildung 9: MVC-Dateistruktur des Python-Programms

Durch das \_\_init\_\_.py File in jedem Ordner wird das Unterverzeichnis als Paket erkannt, über welches die Dateien danach kontrolliert importiert werden können, ohne dass der Ausführungspfad von Python verändert werden muss.

#### Singleton Pattern

Das Singleton Pattern erzwingt, dass von einem Klasse nur eine (globale) Instanz existiert. Zur Kontrolle der Zugriffe wurde es für das Auslesen der seriellen Schnittstelle (RFIDReader), dem Karten Manager (CardManager), dem Spiel Prüfer (GameController) und dem Spiel Modus Manager (GamePlayManager) verwendet.

Um die bestmöglichste Umsetzung des Singleton unter Python zu finden, wurde via Google recherchiert und schlussendlich auf Stack Overflow eine entsprechende Lösung gefunden. Diese Lösung sieht vor, dass im Projekt eine Klasse „Singleton“ erstellt wird, welche danach mit Hilfe der „metaclass“ Funktion[[5]](#footnote-5) implementiert wird. In dieser Klasse „Singleton“ wird das Verhalten der implementierenden Klassen definiert, was in diesem Fall ein ständiger Verweis auf die dasselbe Objekt ist. Da es sich um die einzige Metaklasse des Projekts handelte, wurde sie ebenfalls im Paket „model“ implementiert.



Abbildung 10: UML-Klassendiagramm Singleton und dessen Anwenderklassen

#### Factory Pattern

Das Factory Pattern wird zum Erstellen von Objekten über Methoden, anstatt wie sonst üblich über Konstrukturen, verwendet. Typisch in der Anwendung ist, dass Unterklassen bestimmen welche Objekte erzeugt werden sollen, wobei diese Unterklassen zum Beispiel unterschiedliche Eigenschaften darstellen können.

Für dieses Reaktionsspiel wurde das Pattern verwendet, um verschiedene Typen des Kartenobjekts mit Hilfe einer zentralen Instanz (der „CardFactory“) zu implementieren. Diese unterschiedlichen Kartentypen können unterschiedliche typspezifische Methoden enthalten.

In dieser Implementation hat der Typ „Farbkarte“ (ColorCard) eine Variable „color“ und entsprechende Get- und Set-Methoden. Für den Typ „Studentenkarte“ (StudentCard) wiederum kann der Name erfasst und verwaltet werden.



Abbildung 11: UML-Klassendiagramm der Card Factory

Weitere Karten-Typen können mit Hinzufügen weiterer Klassen realisiert werden. In einer Factory wird jeweils ein Objekt eines spezifischen Typ erstellt und die geteilten Fähigkeiten der vererbten Klasse übernommen. Die CardFactory-Klasse überprüft den Typ, welcher beim erstellen mitgegeben wird und löst ein entsprechendes erstellen des Kartenobjekts aus. Neue Typen von Karten können neue Merkmale haben, nach welchen geprüft wird.

Hinweise zur Implementation eines Factory Pattern unter Python wurden auch über Google recherchiert[[6]](#footnote-6) und entsprechend umgesetzt.

#### Strategy Pattern

Das Strategy Pattern wird verwendet um parallel mehrere Strategien zu implementieren, von welcher zur Laufzeit eine zur aktiven Ausführung festgelegt wird, während des Betriebs aber auch ausgetasucht werden kann. Es eignete sich optimal um die verschiedenen Spielvarianten zu implementieren. Es werden mehrere Klassen als separate Strategien implementiert, welche alle gemeinsame Interface-Methoden bieten aber eine unterschiedliche Logik repräsentieren.

Zur besseren Illustration ist das Pattern in der Abbildung 12 dargestellt. Im GamePlayManager wird die aktive Strategie festgelegt anhand dessen Konfiguration die GamePlayStrategy (GameStrategyEasy oder GameStrategyAdvanced) ausgewählt wird.



Abbildung 12: UML-Klassendiagramm des umgesetzten Strategy Pattern

Im Interface GamePlayStrategy ist die Methode play() definiert, über welche das Spiel gestartet werden kann. Die weiteren Methoden welche in der GameStrategyEasy implementiert sind wurden zur einfacheren Gliederung der Klasse erstellt, können aber beliebig gewählt werden. Die GameStrategyAdvanced wurde noch nicht ausprogrammiert, was der Grund für ihr leeres Aussehen ist.

Implementationsbeispiele zur Umsetzung des Patterns in Python wurden über Google recherchiert[[7]](#footnote-7) und analog dieser realisiert.

### Use Cases

Um die Komplexität des Spiels zu erfassen, wurden einige Fähigkeiten und Anwendungsfälle erdacht. Eine Auswahl ist in den folgenden Unterkapiteln behandelt.

#### Spieldurchlauf

Um ein Spiel mit den Standardeinstellungen zu spielen, kann im Hauptfenster der Startknopf in der Menübar gedrückt werden. Weitere Dialogfenster sind dabei, wie in der folgenden Abbildung 13 dargestellt, nicht nötig.



Abbildung 13: Flussdiagramm Spieldurchlauf

Um die zufällige vom Computer gewählte Karte während dem Spiel zu aktualisieren, muss die Anzeige im GUI angepasst werden damit sie als Aufruf aus dem Model direkt ausgeführt wird.

Der genaue Ablauf der Interaktion wird in folgendem Sequenzdiagramm dargestellt:



Abbildung 14: UML Sequenzdiagramm Spieldurchlauf

Bei jeder Aktualisierung der Darstellung der zufällig gewählten Karte wird über die „random“ Funktion eine Nummer generiert, welche im Kartenmanager einer Karte zugeordnet ist. Aufgrund dieser wird die Anzeige im Hauptfenster angepasst. Die Spiellogik wird zu diesem Zeitpunkt im Model ausgeführt. Sobald die Anzeige aktualisiert wird, wird auch die Zeitmessfunktion gestartet. Danach ist der Spieler aufgefordert schnellstmöglich die korrekte Karte am RFID Reader anzulegen. Sobald der RFID-Sensor eine Karte erkennt wird die UID ausgelesen und im Kartenmanager anhand der UID geprüft, ob es sich um eine valide Eingabe handelt. Etwa zeitgleich wird die Zeit ausgelesen und die verwendete Dauer kalkuliert.

Der Spielmodus ist zum Zeitpunkt der Projektabgabe noch nicht komplett fertig programmiert. Das vergleichen der Zeit und der Karte findet erst über die Konsolenausgabe und noch nicht über einen separaten Dialg statt, weshalb dies im Sequenzdiagramm nicht entsprechend abgebildet ist.

#### Konfiguration der seriellen Verbindung

Um die serielle Verbindung anzupassen, was in seltenen Fällen möglich sein muss, damit die Verbindung zum RFID Reader funktioniert, kann über das „File“ Menü der entsprechende Dialog aufgerufen werden. Der Ablauf wird in Abbildung 16 verdeutlicht.



Abbildung 15: Flussdiagrammzur Konfiguration der Seriellen Verbindung

Durch die zentrale Referenzierung von Einstellungen im Controller muss nach Aktualisierung im Model diese Information mittels des GameController auch vom RFIDReader aus im GameController hinterlegt werden. Bei der Initialisierung der seriellen Verbindung wird deswegen auch das aktive Interface im GameController ausgelesen.



Abbildung 16: UML Sequenzdiagramm Serielle Interface Aktualisierung

Das Dialogfenster kann dabei über die Optionen „Save“ und „Cancel“ verlassen werden. Bei „Save“ wird die aktuelle Anzeige übernommen und hinterlegt, bei „Cancel“ werden allfällige Änderungen verworfen.

#### Wahl des Spielmodus

Der Spielmodus kann über einen separaten Dialog aufgerufen werden. Die folgende Abbildung 18 zeigt das Dialogfenster, in der nachfolgenden Abbildung 19 wird der Verlauf in einem Flussdiagramm dargestellt.

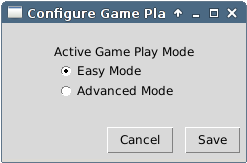


Abbildung 17: GamePlayMode-Dialog



Abbildung 18: Flussdiagramm zur Wahl des Spielmodus

Auch für diese Funktion wurde das MVC Pattern eingehalten. So ruft der Dialog aus der View den Controller auf, wobei dieser die den Aufruf direkt weiterleitet. Das Model wiederum registriert Änderungen wiederum beim Controller. Dies ist noch besser in der folgenden Abbildung 26 ersichtlich:



Abbildung 19: Sequenzdiagramm Spielmodus Dialog

Auch dieses Dialog Fenster kann über die Optionen „Save“ oder „Cancel“ wieder verlassen werden.

## Implementierungsdetails

Im folgenden Abschnitt sind einige interessante Hin- und Verweise auf Implementierungen zu finden.

### Zeitmessfunktion

Das Messen der Zeit wurde in der Klasse „TimeMeasure“ realisiert. Die separate Klasse ermöglicht eine einfache Kapselung der Funktion und wird nur bei Bedarf vom aktiven Spiel Modus aufgerufen.

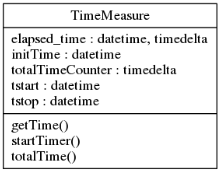


Abbildung 20: UML-Klassendiagramm „TimeMeasure

Zum Messen der Zeit wurde dabei auf die Bibliothek datetime gesetzt. Ein interessantes Detail dazu ist, dass datetime Additionen von Zeiten über die Funktion timedelta zulässt.

### Serielle Verbindung zum Arduino

#### Implementation Arduino

Das Arduino erhält den Code über die produkteigene Entwicklungsumgebung „Arduino IDE“. Über dessen-interne „Hochladungsfunktion“ kann der erstellte Programcode auf das Arduino gespielt, sowie eine serielle Verbindung zum Arduino geöffnet werden. Danach wird der programmierte Ablauf in einer ständigen Schlaufe ausgeführt.

Das im Spiel verwendete Arduino Skript (welches bei Arduino „Sketch“ genant wird)führt in der dauernd laufenden Schlaufe das permanente Auslesen von UIDs der angelegten Karten durch. Als Karten erkannte Eingaben werden vom Skript eindeutig markiert und über die serielle Verbindung an Python übertragen, wo sie von Python so formatiert werden, dass sie im Spiel weiterverwendbar sind.

Eine Herausforderung der Implementation war die Behandlung der ständig auch vorhandenen „Noise“, welches auch nach Optimieren des Skripts nicht vollständig ausgeschlossen werden konnte. Die Noise wird vom Xbee Reader leider mit verursacht und ist bereits auf dem Cooking Hacks Blogeintrag in den Bildern erkennbar. Die genaue Ursache konnte während des Projekts nicht eruiert werden.

#### Implementation Python

Auf Seite von Python wurde das Auslesen in der Python-Klasse „RFIDReader“ realisiert. Dabei wird im Konstruktor der Zugriff auf das Serial Interface geöffnet und erst beim Beenden des Spiels wieder entfernt. Damit technisch keine parallelen Zugriffe möglich sind, wurde diese Klasse als Singleton implementiert, was ein paralleles existieren von mehreren Instanzen der selben Klasse verhindert. Die Abbildung 21 zeigt die UML Darstellung der dafür erstellten Klasse.

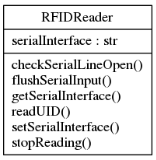


Abbildung 21: UML-Klassendiagramm RFIDReader

Für den Zugriff auf das Serielle Interface wurde die Bibliothek „serial“ verwendet. Standardmässig ist es dabei möglich, dass beim Hinhalten einer Karte die Karte mehrfach ausgelesen wird. Um den Buffer nicht zu schnell vollaufen zu lassen und so das Spiel zu verderben, wird die erste erkannte UID gelesen und weitergeleitet. Zum Start der nächsten Runde wird zuerst der Buffer geleert, so dass in jedem Durchgang eine Karte erneut erkannt werden muss.

Auch zu Berücksichten war die Tatsache, dass das öffnen der Verbindung einen kurzen Moment braucht und so Karten nicht sofort erkannt werden können. Dies war der Hauptgrund für das öffnen der Verbindung zu Beginn und beenden nach Ende des Durchgangs, und nur einem leeren des Cashes zwischen den Runden.

### Serialisierung der Kartenkonfiguration

Um die Kartenkonfiguration abzuspeichern wurden die zu Testzwecken erfassten Karten im CardManager serialisiert. Python bietet dazu die Bibliothek „pickle“ an. In der aktuellen Implementation wird dafür ein vordefinierter Name verwendet.

Wie in der Abbildung 21 sichtbar, findet das „pickeln“ nach einer einfachen Syntax statt. Der Einfachheit halber werden dabei nicht die als String aufbereiteten Informationen im Kartenmanager abgespeichert, sondern direkt das Array „activeCards“, welches die aktuell konfigurierten Karten enthält.

|  |
| --- |
| def saveConfiguration(self):  with open('cardPickle.pickle', 'wb') as f:  pickle.dump(self.activeCards, f, pickle.HIGHEST\_PROTOCOL)  def loadConfiguration(self):  with open('cardPickle.pickle', 'rb') as f:  self.activeCards = pickle.load(f) |

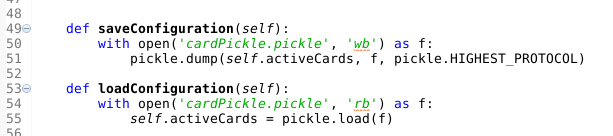


Abbildung 22: Source Code Ausschnitt "pickle"

### Entwickeln des GUI

#### Analyse

Aus Kompatiblitätsgründen wurde das GUI mit TkInter entwickelt. Eine zusätzliche Herausforderung war der gewählte objektorientierte Ansatz, welcher nicht in allen Hilfestellungen und Anleitungen abgebildet war. Umfassende Anleitungen dazu waren relativ schwierig zu finden, da sie meistens nur sehr spezifisch für ein Problemfalll waren und es keine Anleitung gab, welche die generelle Verwendung von Tkinter in der gewählten Python-Version mittels objektorientierter Programmierung beschrieben.

#### **Umsetzung**

Jedes Fenster wird über seine eigene Klasse angesprochen. Zugriffe auf die Modellklassen wurden über einen Controller realisiert. Insgesammt gibt es zurzeit ein Hauptfenster und drei Dialogfenster, über welche Spielkonfigurationen vorgenommen werden können.

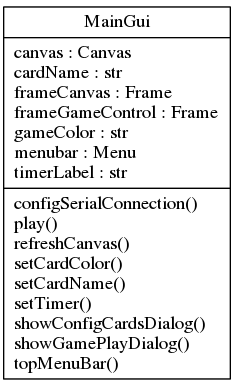


Abbildung 23: UML-Klassendiagramm MainGUI

Die grösste Herausforderung bei der Implementierung des Hauptfensters war die automatische Aktualisierung sobald eine neue Karte geladen werden sollte.Während der Aktualisierung der Karte im Hauptfenster befindet sich die Applikation immer noch in der Spieldurchführung, wodurch das GUI nicht automatisch aktualisisert wird, obwohl die entsprechenden GUI Elementarameter aktualisiert wurden. Standardmässig würde dieses Problem mit Hilfe von zwei Threads gelöst, welche die GUI Updates parallel im zweiten Thread ausführen würden, während im ersten Thread die play() Methode der aktiven GameStrategy weiter ausgeführt würde. Dies wurde allerdings vermieden, um die Komplexität nicht noch zusätzlich zu erhöhen. Die Problematik von Aktualisierungen von Canvas war bereits bekannt und konnte über ein hinzufügen von „self.canvas.update\_idletasks()“ gelöst werden.

Das folgende Bild 24 zeigt nun die vorhandenen Dialogfenster.

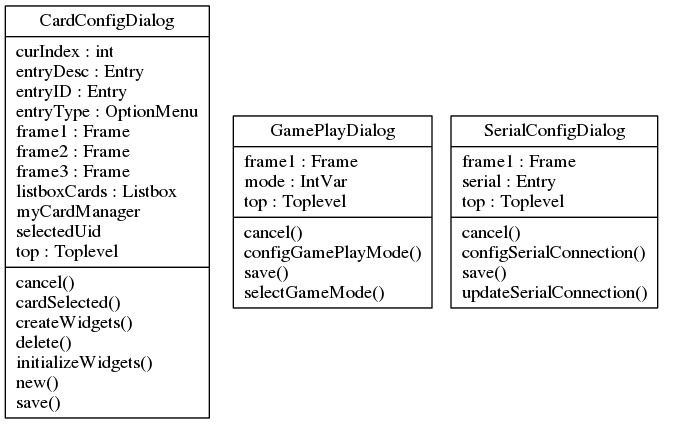


Abbildung 24: UML-Klassendiagramme der Dialog GUIs

Beim Dialog für die Kartenkonfiguration war die grösste Herausforderung ein Konzeptzu entwickeln, wie Karten editiert werden können. Angedacht ist nun, dass bei einem Hinzufügen über den „New“ Button ein neuer Eintrag in der Liste erstellt wird, in welchem danach die Daten eingetragen werden können. Entfernt werden kann eine Karte mittels Selektion in der Liste und dem anschliessenden drücken des „Delete“ Button. Ein direktes editieren ist zur Zeit nicht angedacht, Da dieses Fenster zudem nicht fertig ausprogrammeirt wurde, wird es nicht detailiert beschrieben.

Die beiden anderen Dialoge zur Konfiguration des seriellen Interfaces und zur Wahl des Spielmodus sind bereits ausführlich im Abschnitt 4.5.2.2 und 4.5.2.3 beschrieben.

### Entwickeln der Spiellogik

Während des ganzen zweiten Teils des Projektverlaufs zeigte sich, dass die anfängliche Wahl des Factory Pattern zur Erzeugung der Spielkarten nun die Programmierung verkomplizierte. Zu jedem Zeitpunkt zu welchem auf eine Karte zugegriffen oder Informationen dieser dargestellt werden sollte, musste eine Prüfung des Typs durchgeführt werden damit die korrekte Methode auf dem Objekt ausgeführt werden konnte. Der einfachheit halber wurde deshalb bei der Implementierung der GameStrategyEasy auf ein solches prüfen verzichtet und nur Karten des Typs „Color“ implementiert. Wie bereits im Kapitel 4.5.2.1 dargestellt musste die implementierte Spiellogik viele Funktionen verwenden. Da diese Funktionen aber bereits zu Beginn ordentlich gekapselt wurden, waren Zugriffe über dafür entworfene Methoden einfach möglich.

|  |
| --- |
| def play(self):  print ("Started Easy Mode")  activeRound = 0  while (activeRound < self.numberOfRounds):  print("-------------next round")  self.showNeededCard()  self.readSignal()  timeNeeded = self.myTimeMeasure.getTime()  print("Time needed was: ", timeNeeded)  print("total time in gamePlayModes: ", self.myTimeMeasure.totalTime())  ###  time.sleep(2)  activeRound +=1  self.myRFIDReaderConnection.stopReading()  print("end time in gamePlayModes: ", self.myTimeMeasure.totalTime())  print("stopped reading rfid signal")    def showNeededCard(self):  self.activeDescription = gameLogics.GameLogic.getRandomDescription(self)  self.activeColorParsing = self.activeDescription.islower()  print("Current Card: ", self.activeDescription)    rFIDReader.RFIDReader().flushSerialInput()  controller.gameController.GameController().updateCurrentCardbyColor(self.activeDescription)    def readSignal(self):  startTime = self.myTimeMeasure.startTimer()  print("Start Counter at ", startTime)  readUid = self.myRFIDReaderConnection.readUID()    stringUid = readUid.decode(encoding='UTF-8')  print("read UID: ", readUid)  print("string UID: ", stringUid)  readCard = self.myCardManager.getCardByID(stringUid)  readDescription = readCard.getColor()  print("read Description: ", readDescription ) |

Abbildung 25: Source Code zur Methode play() der Klasse GameStrategyEasy

Der Spielmodus ist dabei noch sehr rudimentär. Ebenfalls sind im Code die Ausgaben über die Konsolen ersichtlich, welche die Prüfergebnisse und Zeiten bereits einblenden. Eine entsprechende Implementation im GUI ist noch ausstehend.

Die Wahl der Spielstrategie wird über den dafür angefertigten Dialog festgelegt. Zur Zeit lässt sich die Strategie wie folgt beschreiben, wobei sie noch nicht komplett so umgesetzt ist:

#### 

Die Spiellogik der „GameStrategyEasy“ ist es, nach Zufall eine Karte aus dem Kartenmanager zu laden und deren Farbe darzustellen. Zeitgleich wird die Zeitmessung gestartet. Sobald die korrekte Karte an das Lesegerät gehalten wird, wird die Zeitmessung gestoppt und die nächste Karte geladen. Die Reaktionszeiten aller 10 Durchgänge (10 ist in einer separaten Variablen konfiguriert, ein Ausbau um die Anzahl Runden zu verändern wäre möglich) werden zusammengezählt. Pro falsch hingehaltene Karte werden 5 Strafsekunden hinzugefügt.

Weitere Strategien sind über neue Klassen hinzufügbar. Ideen dafür sind zum Beispiel für weitere Farbkarten, die Anzeige in Schrift und Farbe unterschiedlich zu randomisieren, wobei dann nur eine Farbanzeige wirklich für die korrekte Wertung relevant ist und die andere verwirren soll. Eine weitere Möglichkeit für den Advanced-Modus wäre, die Farbanzeige automatisch nach wenigen Sekunden (oder Hunderstelsekunden) zu ändern, so dass dies quasi einer „verpassten Gelegenheit“ gleich käme. Dabei könnten die korrekt hingehaltenen Karten gezählt werden.

Der GamePlayManager setzt über die im Array zugeordnete Nummer (welche der Reihenfolge des einlesens von Spiel Strategien entsprich) die Strategie aktiv. Im Code wurde dies wie in Abbildung 26 gezeigt gelöst.

|  |
| --- |
| class GamePlayManager(metaclass=singleton.Singleton):    def \_\_init\_\_(self):  '''  Constructor  '''  self.activeGamePlayStrategy = 0  print("gamePlayStrategy init")  self.gameStrategies = []  self.gameStrategies.append(GameStrategyEasy())  self.gameStrategies.append(GameStrategyAdvanced())    def getGamePlayMode(self):  print("Mode is currently: ", self.activeGamePlayStrategy)  return self.activeGamePlayStrategy    def setGamePlayMode(self, newMode):  self.activeGamePlayStrategy = newMode  controller.gameController.GameController().registerGameStrategy(self.gameStrategies[self.activeGamePlayStrategy])    print("Mode has been set to: ", self.activeGamePlayStrategy) |

Abbildung 26: Source Code des GamePlayManagers

Zuerst werden die vorhandenen Strategien in einem Array hinzugefügt. Die dem Arrayeintrag zugeordnete Nummer wird danach über die registerGameStrategy() Methode gesetzt und dieser Spielmodus danach ausgeführt.

### Tests

Um die Entwicklung effizient zu gestalten zeigte sich schon bald, dass einige Tests mit Hilfe der unittest Bibliothek das Programmieren stark vereinfachten. Wurden zuerst noch Klassen direkt über Aufrufe im File getestet, wurde dies zum Zeitpunkt von mehreren zusammenhängenden und aufeinander aufbauenden Klassen unrealistisch.

Unittests wurden für die technischen Basis Funktionen wie für die Card Factory, dem Card Manager, dem Testen des Seriellen Zugriffs und dem Testen von Random Werten und der Auswahl der entsprechenden Karten verwendet. Da für das korrekte Starten der GUIs einige Objekte zuerst angelegt und definiert werden müssen, wurde dies auch über eine Testklasse ausgelöst. Unittests wurden dabei den aus Java bekannten Prinzipien ähnlich implementiert.

Als Beispiel ist im folgenden die Test Klasse zum Card Manager aufgeführt.

|  |
| --- |
| #! /usr/bin/python  # Test code  import unittest  import model.cardManager as cardManager  import model.cardFactory as cardFactory  import model.rFIDReader as rFIDReader  class TestManagerCreation(unittest.TestCase):    #Create Test Object  def setUp(self):  self.myCardFactory = cardFactory.CardFactory()  self.card1 = self.myCardFactory.createCard("Color")  self.card2 = self.myCardFactory.createCard("Color")  self.card3 = self.myCardFactory.createCard("Color")  self.card1.setColor("Red")  self.card2.setColor("Blue")  self.card3.setColor("Green")  self.card1.setID("AD B8 57 94")  self.card2.setID("B2 FB 30 E9")  self.card3.setID("8D F4 61 94")  self.myCardManager = cardManager.CardManager()  self.myCardManager.addCard(self.card1)  self.myCardManager.addCard(self.card2)  self.myCardManager.addCard(self.card3)    def tearDown(self):  cardManager.CardManager().getEmptyManager(self.myCardManager)      # Run Tests  def test\_getArraySize(self):  self.assertEqual(self.myCardManager.getSizeCardArray(), 3)      def test\_singletonImplementation(self):  self.myCardManager.addCard(self.card1)  self.myCardManager.addCard(self.card2)  self.myCardManager.addCard(self.card3)  self.assertEqual(self.myCardManager.getSizeCardArray(), 6)      def test\_storeConfiguration(self):  self.size = self.myCardManager.getSizeCardArray()  self.assertEqual(self.myCardManager.saveConfiguration(), None)    def test\_loadStoredConfiguration(self):  self.myCardManager = cardManager.CardManager()  self.myCardManager.loadConfiguration()  self.myCardManager.outputCards()    self.assertEqual(self.myCardManager.getSizeCardArray(), 3)  def test\_getCard(self):  self.myCardManager = cardManager.CardManager()  self.foundCard = self.myCardManager.getCardByID("AD B8 57 94")  self.assertTrue(type(self.foundCard))  # print("testCard: ", type(self.foundCard))  if \_\_name\_\_ == '\_\_main\_\_':  unittest.main() |

Abbildung 27: Test Klasse des Card Manager

Da Tests nicht immer in der selben Reihenfolge ausgeführt werden ist es wichtig, Tests nicht aufeinander aufbauend zu entwerfen oder wenn doch, dies entsprechend so fest mit Methodenaufrufe zu verankern. Dies war ein Problem als die Implementation des Singleton getestet wurde, konnte danach aber mit dem sauberen Aufbauen einer setUp() und tearDown() Methode gelöst werden. SetUp() und TearDown() bereiten, ihren englischen Namen entsprechend, die Testumgebung vor und räumen sie anschliessend wieder auf. Dies wird vor und nach jeder Test Methode durchgeführt.

Die Tests hätten beliebig weiter geführt werden können, was die Qualität von Code stark erhöhen kann, darauf wurde allerdings aus zeitlichen Gründen verzichtet. Stattdessen wurden funktionale Tests favorisiert.

## Fazit

Die zum Zeitpunkt des Design Review festgelegten Stories konnten zu grossen Teilen umgesetzt werden. Während des Projektverlaufs wurden die Prioritäten zum Teil geändert, da Implementationen während der Umsetzungsphase sich als einfacher herausstellten als ursprünglich gedacht war. Andere Stories wurden in den Backlog verschoben da dazu noch weitere Implementationen und Features nötig gewesen wären, welche zum Zeitpunkt der Planung noch nicht Abschätzbar waren.

Für die Umsetzung wurde viel Wert auf die einfache technische Erweiterbarkeit gelegt um auch nach offiziellem Projektende eine bekannte Übungsumgebung zu haben. Die Planung der Stories hätte noch detaillierter erfolgen können, wobei ich mir erhoffe zukünftige Stories mit steigender Programmiererfahrung besser abschätzen zu können.

Die Testumgebung zeigte sich als äusserst hilfreich und wird in zukünftigen Projekten definitiv gleich zu Beginn auch in der Abschätzung der Aufwände für Stories besser berücksichtigt.

## Ausblick

Die während dem Projekt programmierte Basis ist ein guter Ausgangspunkt für weitere Implementationen. Es können nicht nur weitere Kartentypen hinzugefügt werden, weitere Spielmodi können genauso implementiert werden.

Eine Liste von weiteren Optimierungsbeispielen, Komplettierungen und Erweiterungen ist hier zu finden, sie ist aber natürlich nicht abschliessend.

Tabelle 8: Weiterentwicklungsmöglichkeiten

|  |  |  |
| --- | --- | --- |
| 1 | Advanced Mode | Für Farbkarten: „Verwirrungstaktik“ mit Farbeinfärbung anders als dargestelltem Text. |
| 2 | Speed Modus | Für Farbkarten: Beschleunigte Anzeige, begrenzt auf Zeitwert wobei der Punktemodus verändert werden müsste. |
| 3 | Top Score Liste | Implementierung einer Top Scorer Liste, welche die besten Resultate festhält und über eine Serialisierung speichert. |
| 4 | Anzeige ob serielle Verbindung besteht | Eine Anzeige im Hauptfenster zeigt den Status der seriellen Verbindung an. Falls die Verbindung verloren gehen würde könnte somit das Spiel unterbrochen werden. |
| 5 | Anzeige Zeit im Hauptfenster | Eine Anzeige im Hauptfenster der Zeit kann den Spieldruck erhöhen und das Spiel entsprechend spannender machen. |
| 6 | Countdown vor dem Spiel | Ein kurzer Countdown vor dem Spielstart kann die Vorbereitungsphase verbessern und würde den Spieleffekt erhöhen. |

# Diskussion des eignen Beitrags

## Studieren der RFID Technologie und Bauen eines Reader mittels eines Arduinos und einem RFID Shield

Das Protokoll welches für RFID-Transaktionen der Studentenkarte verwendet wird ist im ISO Standard 14443 definiert. Die Erkenntnisse zum Protokoll sind im Kapitel 3.1 festgehalten. Um einen RFID Reader selber zu bauen galt es zuerst herauszufinden, welche der erhältlichen Komponenten für den Einsatz geeignet wären. Ein wichtiger Faktor beim eigenen Zusammenbau war die gute Dokumentation, welche für den Xbee Reader vorhanden war. Zudem sollte ein Chipsatz auf dem Reader verbaut sein, welcher möglichst nicht nur die öffentlich bekannten Funktionalitäten unterstützt, sondern auch Fähigkeiten wie das passive Mitlesen von Transaktionen ermöglicht. Nach einiger Recherche konnte festgestellt werden, dass der Chipsatz PN532 dies technisch unterstützt, entsprechende Arduino Code-Implementationen aber nicht aufgeführt sind.

Durch die gelegte Basis kann dieses Projekt nach Vollendung der Semesterarbeit beliebig fortgesetzt werden. Die Grenzen der Fähigkeiten des Sensors sind noch nicht erreicht.

## Dokumentieren von Herausforderungen beim Bauen eines RFID Readers

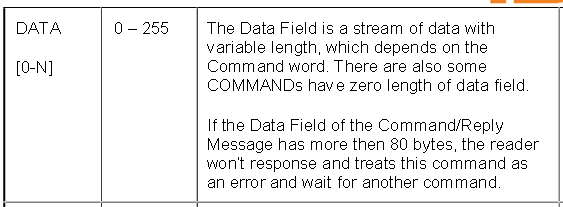
Die grössten Herausforderungen waren die versteckten Limitationen, welche gewisse Reader enthielten, zu finden. So hatte der ursprünglich ausgewählte Reader von Seeed Studios eine Datenleselimite von 80 bytes, weshalb er aus Zuverlässigkeitsgründen nicht weiter eingesetzt wurde 

Abbildung 28: Ausschnitt aus der Dokumentation zum Seeed Studio Reader[[8]](#footnote-8)

Der jetzt verwendete Reader von Xbee wiederum hat eine Limitation, welche ein gleichzeitiges Auslesen und Schreiben über das Serial Interface am Arduino unterbindet. Als Konsequenz kann der ausgeführte Code auf dem Arduino nicht von Eingaben über das Serial Interface beeinflusst werden. Reaktionen können einzig über das RFID-Modul provoziert werden. Dies hat die Entscheidung massgeblich beeinflusst, auf dem Arduino lediglich ein Skript zu aktivieren, welches die Karten erkennt, die UIDs über den seriellen Bus an das Python Programm überträgt, aber die Karten selber nicht weiter verändert.

Alternativ hätte die komplette RFID-Protokolllogik ebenfalls über das serielle Interface übertragen werden können, wobei das Arduino dann nur noch als reine Steckplatine gedient hätte. Dieser Modus wird „Transparent Operation“ genannt. Um dies zu erreichen hätte allerdings auf dem Arduino der ARM-Chip entfernt werden müssen, was bei der für das Projekt vorgelegene Version nicht möglich war. Zudem wäre der Arduino Teil des Projekts daraufhin zu gering ausgefallen, was dem ursprünglichen Ziel, ein Projekt mit Hilfe eines Arduinos zu realisieren, nicht diente.

## Machbarkeitsanalyse zum Auslesen unserer neuen Campus Card

Die Online Recherchen ergaben, dass die Studentenausweise von InterCard erzeugt wurden. Ein Online-Kurzportrait und eine Erfolgsstory zur Implementation konnten dies dokumentieren, sie ist in der Abbildung.29 eingeblendet.



Abbildung 29: Success Story InterCard - ZHAW

Auf der Webseite von InterCard wird präsentiert, welche Kartenmodi zur Verfügung stehen. Ein visueller Abgleich zeigte, dass für unseren Studentenausweis keine Karte mit einem Chip verwendet wurde und entsprechend auf den Typ Mifare Classic 1 Kbyte gesetzt wurde (siehe auch Abbildung 29).

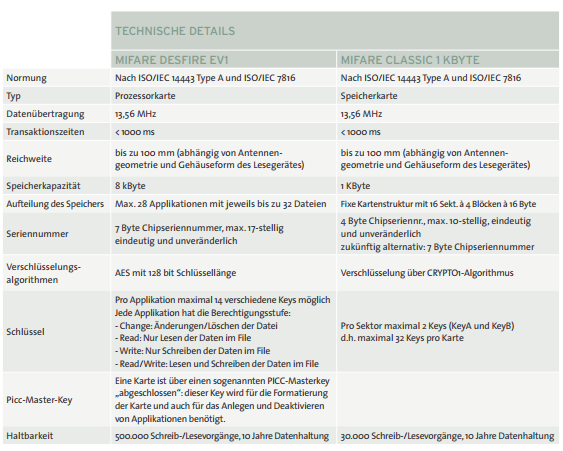


Abbildung 30: Kartentypen im Angebot von InterCard[[9]](#footnote-9)

Für den verwendeten CRYPTO1 Algorithmus der Studentenkarte existiert laut Wikipedia-Eintrag[[10]](#footnote-10) seit 2008 genug Verständnis, um die „Verschlüsselung“ zu brechen.

Nach Abklärungen mit den Sicherheitsbeauftragten der ZHAW wäre es mir nur nach Unterzeichnen eines NDA erlaubt gewesen, die Sicherheit der auf den Karten abgelegten Daten zu testen. Um für diese Semesterarbeit keine zusätzliche administrative und formelle Hürde zu schaffen wurde entsprechend auf diese Herausforderung verzichtet.

Ein Auslesen der CampusCard wäre allerdings mit den heute technisch verfügbaren Mitteln machbar gewesen. Aus diesem Grund weisst auch die InterCard in ihrem technischen Prospekt, welche in Abbildung 31 dargestellt ist, darauf hin, dass stattdessen die Verwaltungssoftware im Hintergrund Validitätsprüfungen durchführen soll. Der Hinweis auf die „vom Chiphersteller vergebene Seriennummer“ ist trügerisch, da im Internet ohne Probleme Karten erhältlich sind, auf welchen der Seriennummereintrag verändert werden kann.

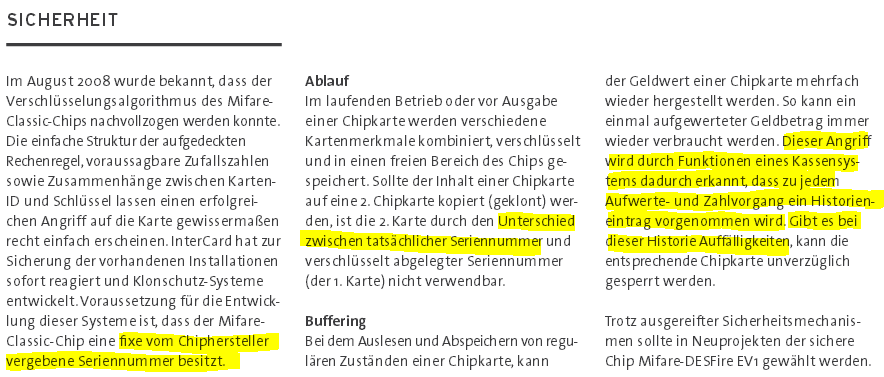


Abbildung 31: Auszug aus dem technischen Prospekt der InterCard

## Spiel in Python und mit einem Arduino

Auf Grund der bereits dokumentierten Schwierigkeiten wurde während der Semesterarbeit das alternative Projektziel eines Spiels realisiert. Die entsprechenden Aufwände und Entscheidungen sind im Kapitel 4 dokumentiert.

Von den ursprünglich erdachten Features wurden zum Projektende einige, allerdings nicht alle umgesetzt. Da die Entwicklung in einer komplett neuen Umgebung statt fand und das letzte Programmierprojekt bereits wieder einige Monate zurück lag, bin ich mit der Umsetzung doch im Grossen und Ganzen zufrieden. Ich hatte gehofft zumindest das GUI komplett fertig ausprogrammieren zu können, was mir beim Kartenkonfigurationsfenster und für die Spiel Feedback Optionen, wie das abgleichen der hingehaltenen Karten und einblenden der verwendeten Zeit, zum offiziellen Endtermin hin nicht gelungen ist. Allerdings waren dafür keine sonderlich technischen Hürden im Weg, sondern einfach die Zeit welche am Ende zu knapp wurde.

Stolz bin ich auf die Architektur der Software und ihre Vorbereitung zur Weiterentwicklung. Die Entwicklung empfand ich dadurch öfters als das Bauen eines Produktes nach einem Plan als dass es bei früheren funktionalen Programmierprojekten der Fall war.

Für ein zukünftiges Projekt würde ich mir mehr Velocity einrechnen, da ich das Suchen und Auswählen von Bibliotheken und das identifizieren von Pattern-Implementationsmöglichkeiten in einer neuen Sprache unterschätzt habe.

# Schlussfolgerungen

Nach Abschluss der Arbeit kann ich sagen, dass das Projekt in weiten Teilen Spass gemacht hat und lehrreich war. Mein Hauptziel des Erlernens der neuen Technologien konnte auf einem Basisniveau erreicht werden. Mit meinen Designentscheiden bin ich sehr zufrieden und denke, diese bei einer Wiederholung wieder so zu lösen. Einzig beim GUI des Spiels hätte ich bei einem nächsten Versuch mit grosser Wahrscheinlichkeit die Aufgabe leicht vereinfacht und stattdessen ein HTML/CSS GUI verwendet, welches über Python angezeigt worden wäre. Da HTML/CSS bereits im Unterricht behandelt wurde hätte dies weniger Einarbeitung benötigt und die GUIs hätten einfacher nach den vorhanden Vorstellungen erstellt werden können. Unsicher bin ich bei der Wahl der Python-Version. Normalerweise ist es immer die beste Idee auf der neuesten, verfügbaren Version zu entwickeln. Da bei den Python-Bibliotheken aber eine so grosse Vielfalt besteht und nicht alle Bibliotheken mit allen Python-Versionen kompatibel sind, hätte ein genaues Abklären von Anforderungen allenfalls eine andere Versionswahl ergeben.

Gerne hätte ich noch mehr am Spielumfang weiter entwickelt. Für das wäre der Zeitaufwand aber durchaus noch 40 Stunden höher ausgefallen, was in Anbetracht des vordefinierten Zeitbudget unrealistisch war.

Schade finde ich, dass ich meine ursprünglichen Versuche zur Sicherheit von RFID-Karten im Speziellen in Bezug auf die CampusCard der ZHAW nicht im Rahmen dieser Arbeit umsetzen konnte. Ein Interesse dafür wäre durchaus immer noch vorhanden, kann aber nicht legal verfolgt werden, da eine Veröffentlichung von Ergebnissen nicht erlaubt ist.

Technisch wäre das Projektprimärziel umsetzbar gewesen, in dem eine direkte Transaktion, zum Beispiel in der Kantine, „mitgehört“ und aufgezeichnet worden wäre. Ein reines passives finden des Schlüssels ohne dazu bekannte Informationen über Brute Force wäre nicht umsetzbar gewesen.

Betreffend dem innovativer Charakter der Arbeit ist festzuhalten, dass meine Recherche keine anderen ähnlichen Spiel zu Tage gebracht hat, welche die RFID-Technologie verwenden und mit Hilfe des Computers deshalb alleine gespielt werden können. Alle Spielmodi (die implementierten und nicht implementierten) waren eigene Ideen analog zu einfachen und bekannten offline Spielvarianten, welche heute zum Beispiel in ähnlicher Form im Spiel Halli Galli bekannt ist.

# Literaturverzeichnis

Tabelle 9: Literaturverzeichnis

|  |  |
| --- | --- |
| Nr. | Verweis |
| L1 | Klaus Finkenzeller: *RFID Handbuch*. Hanser, 6. Auflage, 2012 |
| L2 | Link zum Vortrag an der BSides London 2014  <http://www.youtube.com/watch?v=MGPGv2HnGtU> |
| L3 | Link zu den Slides des Vortrags an der BSides London 2014: <http://www.slideshare.net/d3sre/bsideslondon-rookie-talk-rfid-hacking-an-introduction> |
| L4 | Link zu Vorschlägen zur Singleton Implementierung: <http://stackoverflow.com/questions/6760685/creating-a-singleton-in-python> |
| L5 | Link zu Vorschlägen zur Factory Pattern Implementierung:  <http://stackoverflow.com/questions/21025959/factory-design-pattern> |
| L6 | Link zu Vorschlägen zur Strategy Pattern Implementierung:  <http://codereview.stackexchange.com/questions/20718/the-strategy-design-pattern-for-python-in-a-more-pythonic-way> |
| L7 | Cooking Hacks Anleitung : <http://www.cooking-hacks.com/documentation/tutorials/rfid-13-56-mhz-nfc-module-for-arduino> |
| L8 | Bild Arduino R3 Front: <http://arduino.cc/en/uploads/Main/ArduinoUno_R3_Front.jpg> |
| L9 | Bild Arduino mit Xbee RFID Shield: <http://www.cooking-hacks.com/documentation/tutorials/rfid-13-56-mhz-nfc-module-for-arduino> |
| L10 | InterCard Karten Technologien: <http://www.intercard.org/de/produkte/IC_Karten_Technologien.pdf> |
| L11 | Crypto-1 Verschlüsselungs Algorithmus und Sicherheit: <http://en.wikipedia.org/wiki/Crypto-1> |
| L12 | Handbuch zum Protokoll des Seeed Studio RFID Readers: http://neophob.com/files/rfid/PROTOCOL-821-880%20\_2\_.pdf |

# Anhang

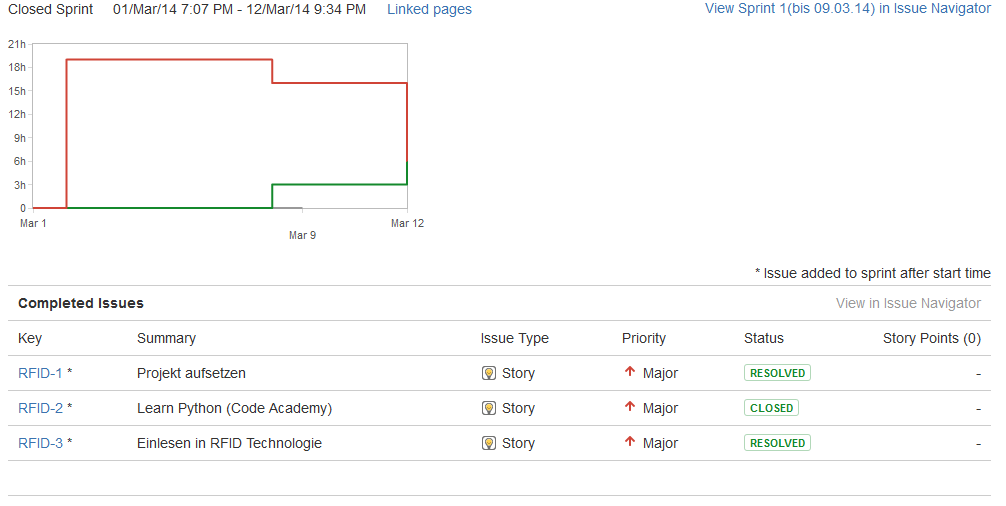
## Glossar

Tabelle 10: Glossar

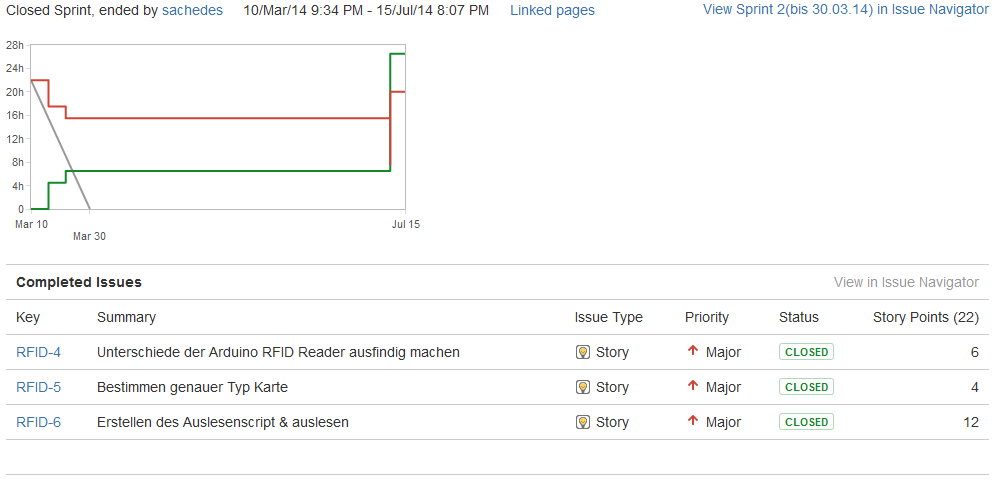
| ID | Begriff | Beschreibung |
| --- | --- | --- |
| G1 | DRY | Don‘t Repeat Yourself, zu Deutsch: Wiederhole dich nicht |
| G2 |  |  |
| G3 | Pomodoro Technik | Methode zum Zeitmanagement, welche Arbeiten in Blöcke unterteilt und Pausen miteinrechnet. Das System wurde für die vorliegende Arbeit angepasst und stattdessen mit zweimal 60-Minutenblöcken gearbeitet.  Siehe: <http://de.wikipedia.org/wiki/Pomodoro-Technik> |
| G4 | Scrum | Agiler Projektmanagementprozess basierend auf Iterationen mit festen Iterationsfenstern.  Siehe: <http://de.wikipedia.org/wiki/Scrum> |
| G5 | Kanban | Agiler Projektmanagementprozess, der ebenfalls wie Scrum mit Iterationen arbeitet, allerdings nicht feste Iterationsfenster nutzt, aber die Anzahl in Bearbeitung befindlicher Tasks limitiert.  Siehe: <http://de.wikipedia.org/wiki/Kanban> |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

## Sprint Reports

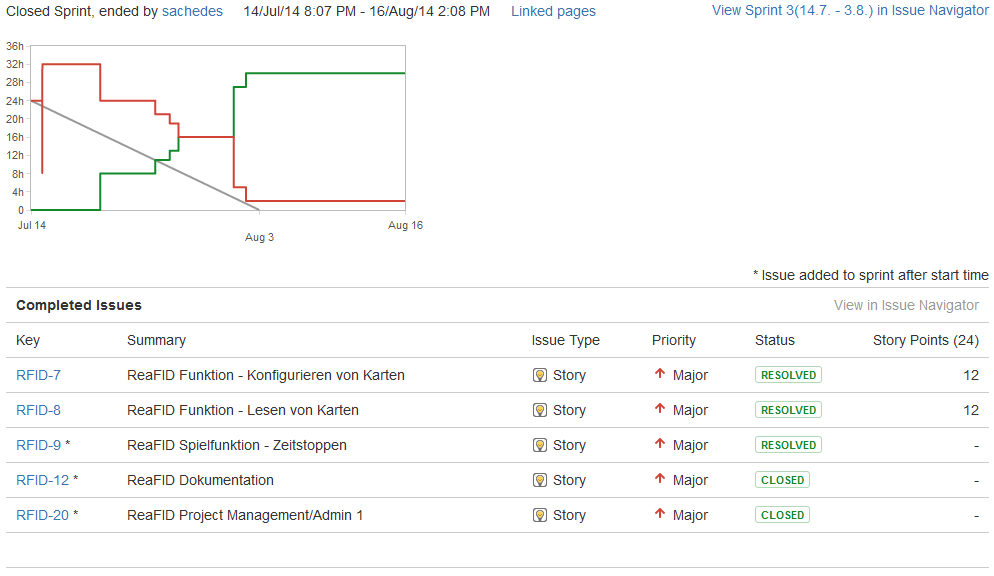
### Sprint 1



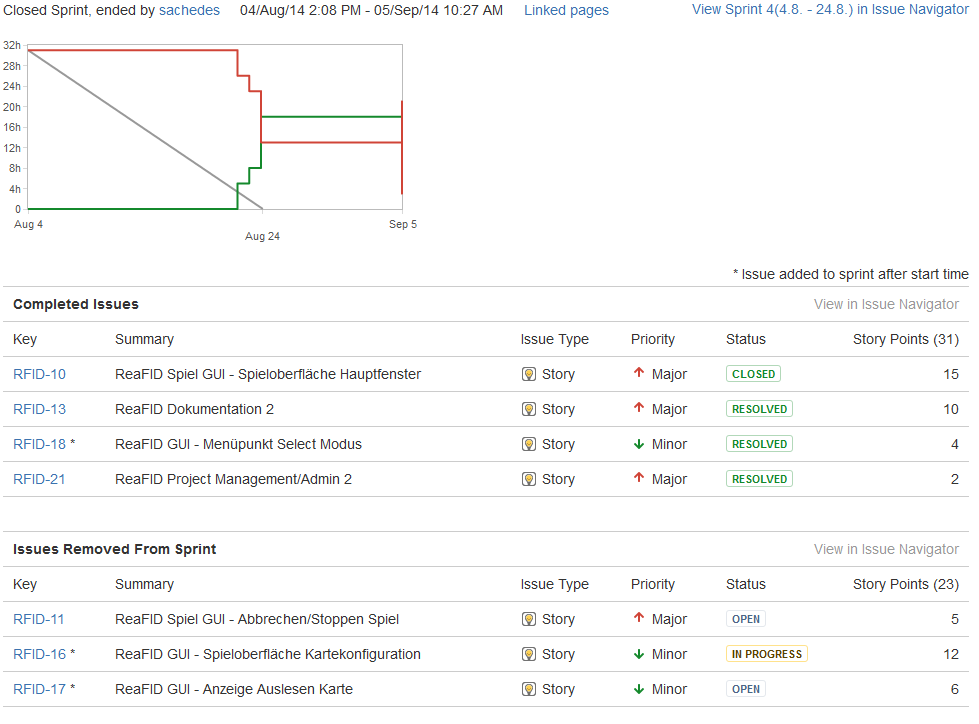
### Sprint 2



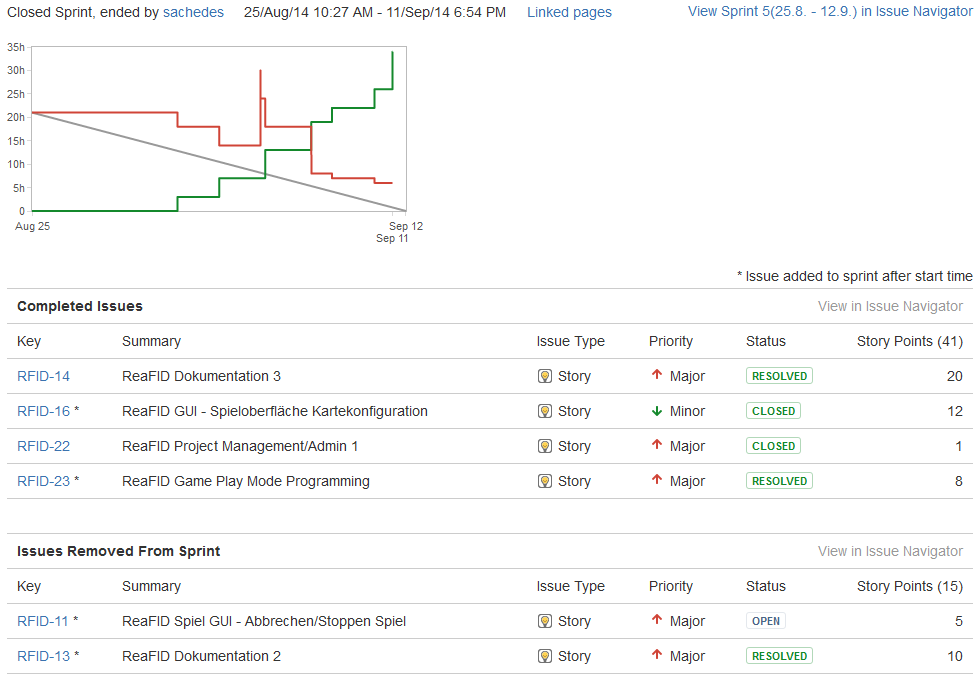
### Sprint 3



### Sprint 4



### Sprint 5



1. Siehe Glossareintrag G1 [↑](#footnote-ref-1)
2. Siehe Literaturlisteneintrag L2 und L3 [↑](#footnote-ref-2)
3. Siehe Literaturverzeichniseintrag: L7 [↑](#footnote-ref-3)
4. Entsprechende Hinweise sind unter anderem auf Wikipedia zu finden. [↑](#footnote-ref-4)
5. Siehe Literaturverzeichniseintrag: L4 [↑](#footnote-ref-5)
6. Siehe Literaturlistenverzeichniseintrag: L5 [↑](#footnote-ref-6)
7. Siehe Literaturverzeichniseintrag: L6 [↑](#footnote-ref-7)
8. Abbildung entnommen von Seite 7 , Literaturlistenverzeichniseintrag L12 [↑](#footnote-ref-8)
9. Siehe Literaturverzeichniseintrag [↑](#footnote-ref-9)
10. Siehe Literaturverzeichniseintrag [↑](#footnote-ref-10)