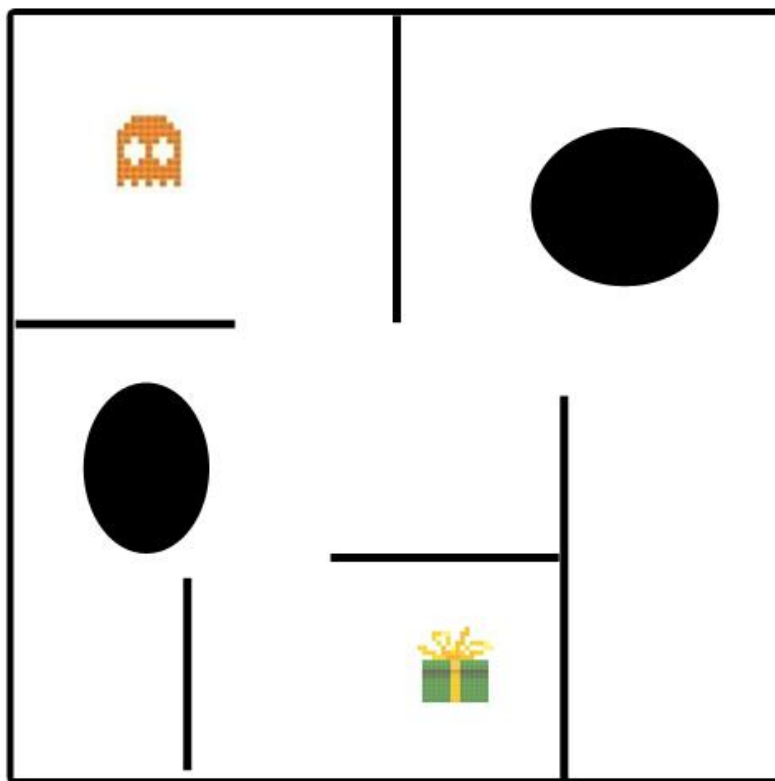


Search Robot



Software Architecture Document

Camille Zanni (zannc2)
Simon Gfeller (gfels4)

Inhalt

Contents

Inhalt	2
Abbildungsverzeichnis	2
Änderungsverzeichnis	4
Einleitung	4
Frontend	4
Use Cases in kürze	4
Hindernisse setzen	4
Roboter setzen	4
Ziel setzten	4
Roboter starten	4
Suche abbrechen	4
Spielfeld löschen	5
Spielfeld importieren	5
Spielfeld exportieren	5
Implementation	5
Field und View	5
Hindernisse	6
Robot	10
Use Cases in kürze	10
Ziel suchen	10
Implementation	10
Strategie	Error! Bookmark not defined.
Strategie	11

Abbildungsverzeichnis

Abbildung 1: UML View und Field	6
Abbildung 2: UML Hindernisse	6
Abbildung 3: UML Tools.....	7
Abbildung 4: UML Handler (Beispiel Circle)	7
Abbildung 5: UML Observer Hindernisse	8
Abbildung 6: UML Observer Field.....	8
Abbildung 7: Selection State Diagramm.....	9
Abbildung 8: UML Selection.....	9
Abbildung 9: UML Strategie	11
Abbildung 10: SD Strategie	10
Abbildung 11: Visualisierung Wegberechnung	11

Änderungsverzeichnis

Version	Datum	Beschreibung	Author
First Draft	7. Januar 2014	First Draft	gfels4/zannc2
Last Draft	9. Januar 2014	Last Draft	gfels4/zannc2

Einleitung

Das Projekt Search Robot wurde in zwei Unterprojekte Frontend und Robot aufgeteilt. So konnte zuerst das Frontend, welches den Editor enthält, implementiert werden. Die Roboterlogik konnten wir dann an das funktionierende Frontend anhängen und implementieren. Diese Aufteilung ist ebenfalls in der Struktur des Codes ersichtlich.

Hinweis: Für die bessere Lesbarkeit stehen alle Diagramme als Bild zur Verfügung.

Frontend

Use Cases in kürze

Für detaillierte Informationen nehmen Sie das Dokument *Use Case Frontend* zur Hand.

Hindernisse setzen

Ein Spieler setzt alle gewünschte Hindernisse (2D Hindernisse) auf dem Spielfeld.

Roboter setzen

Der Spieler setzt den Roboter auf das Spielfeld. Diese Position wird zugleich die Startposition des Roboters sein.

Ziel setzten

Der Spieler setzt das Ziel, das der Roboter suchen muss, auf das Spielfeld.

Roboter starten

Der Spieler startet das Spiel, d.h. der Roboter beginnt das Ziel zu suchen.

Suche abbrechen

Der Spieler unterbricht die Suche nach dem Ziel und kehrt zum Spielfeldeditor zurück

Spielfeld löschen

Der Spieler hat die Möglichkeit das Spiel jederzeit zurückzusetzen, d.h. Hindernisse, Ziel und Roboter werden vom Spielfeld entfernt.

Spielfeld importieren

Ein gespeichertes Spielfeld kann von einer Datei geladen werden.

Spielfeld exportieren

Das Spielfeld wird in einer Datei gespeichert werden.

Implementation

Field und View

Das Field dient als Container der Hindernisse. Die View verwaltet das Field und zeigt alle entsprechenden Hindernisse an, resp. sie gibt bei Änderungen auf dem Spielfeld den Befehl zum neu Zeichnen.

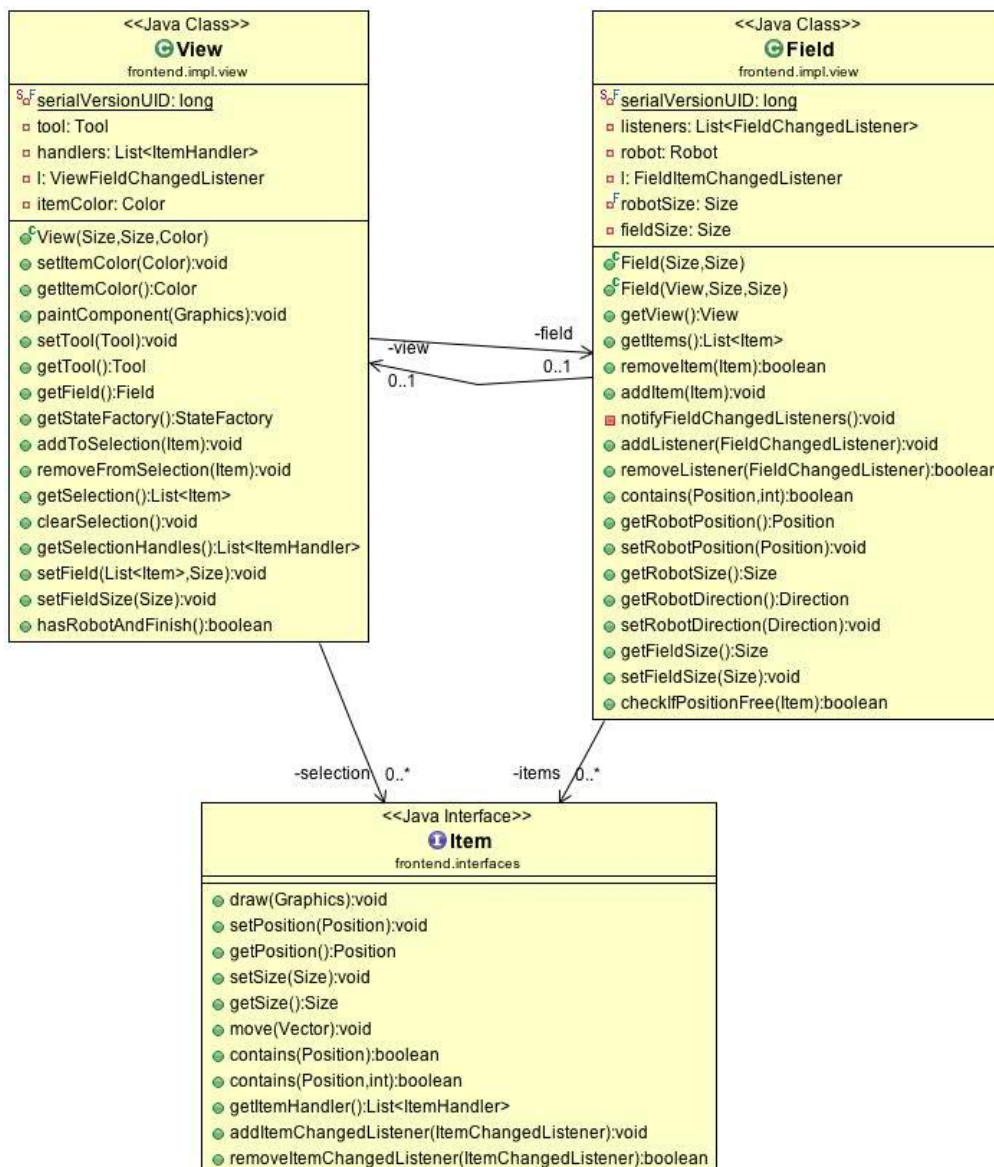


Abbildung 1: UML View und Field

Hindernisse

Zur Implementation der Hindernisse (Items) wurde das Decorator Pattern verwendet. So könnten ohne grosse Modifikationen weitere Hindernisse definiert werden.

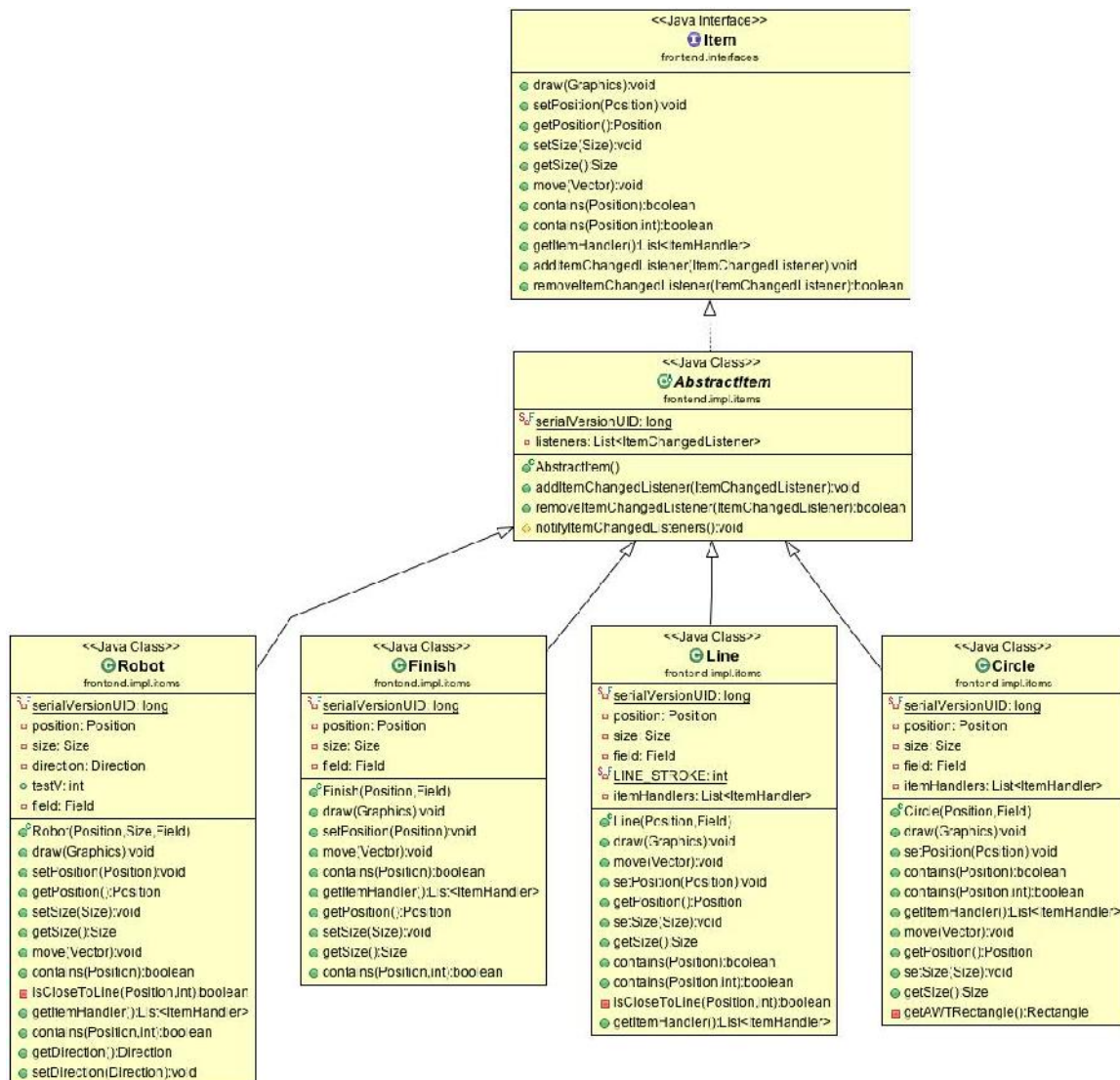


Abbildung 2: UML Hindernisse

Die Hindernisse werden jeweils durch das zugehörige Tool erstellt. Diese wurden anhand des Factory Patterns implementiert.

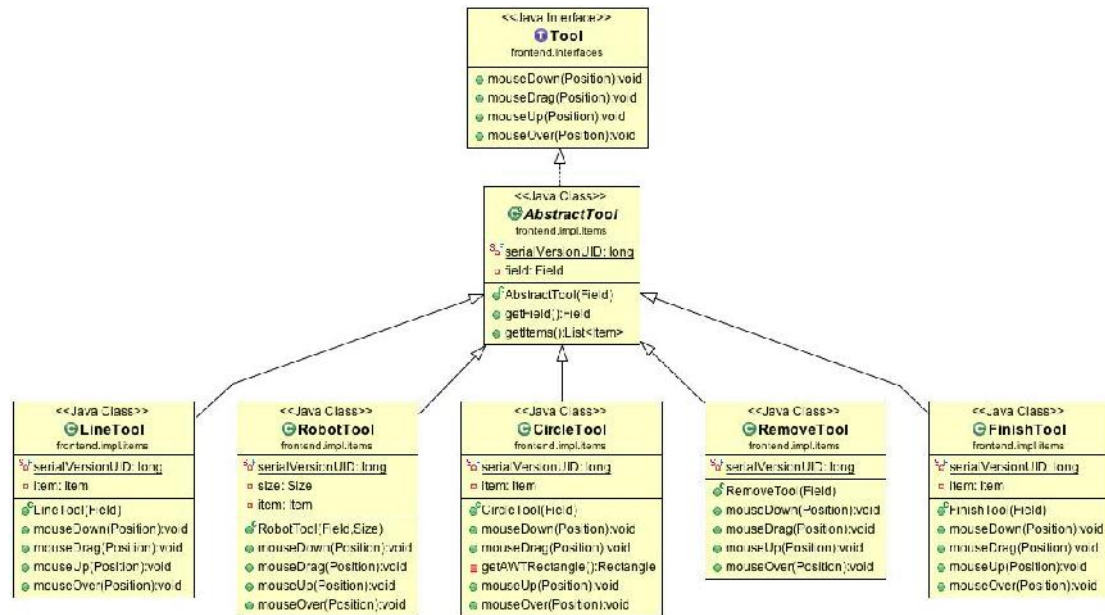


Abbildung 3: UML Tools

Zum vergrößern resp. verkleinern der Hindernisse wurden für die Hindernisse entsprechende Handler implementiert. Die Implementation wurde ebenfalls anhand des Decorator Patterns durchgeführt.

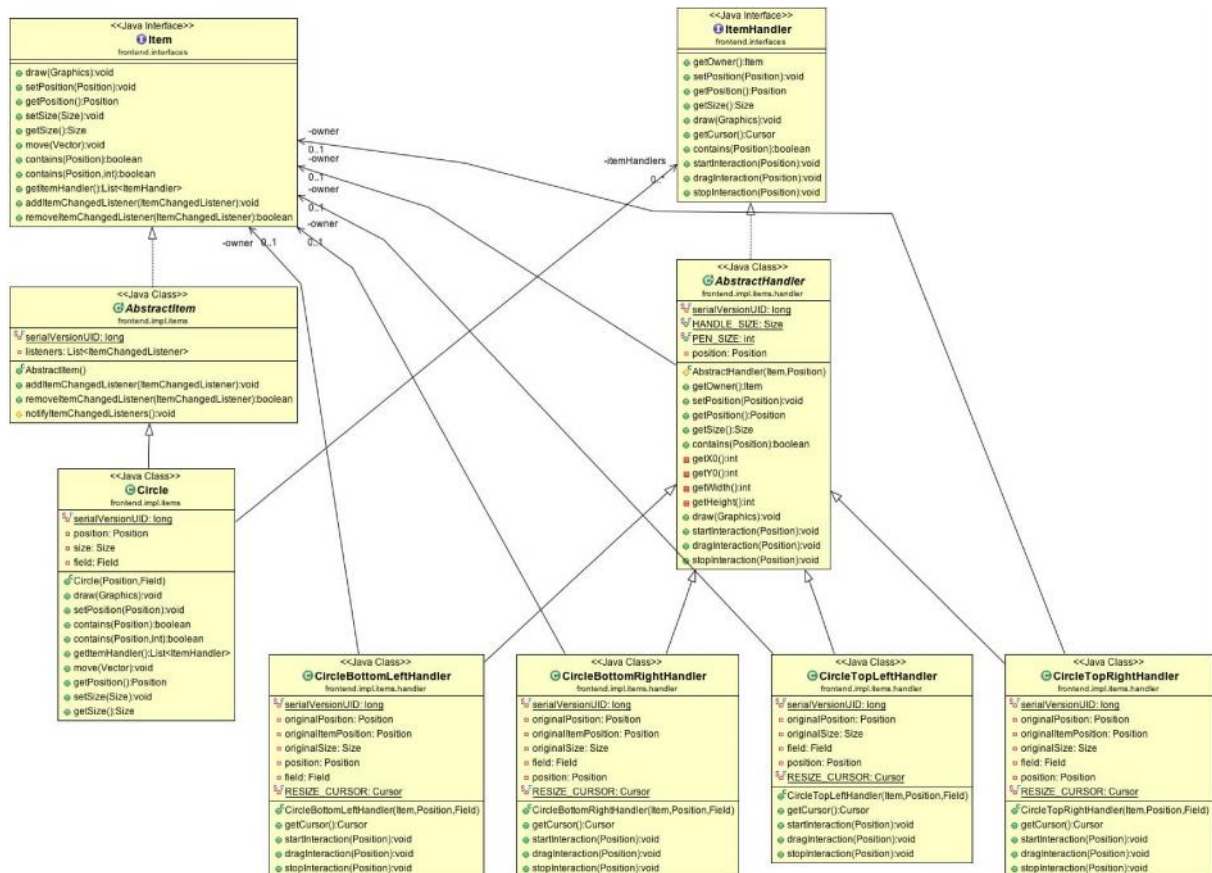


Abbildung 4: UML Handler (Beispiel Circle)

Mit der Vorlage des Observer Patterns wurde für jedes Hindernis ein Listener implementiert. Dieser informiert das Field (in welchem das Hindernis enthalten ist) über diese Änderungen. Das Field besitzt ebenfalls solche Observer, welche die View über allfällige Änderungen informiert. Die View führt schlussendlich die entsprechende Aktion durch.

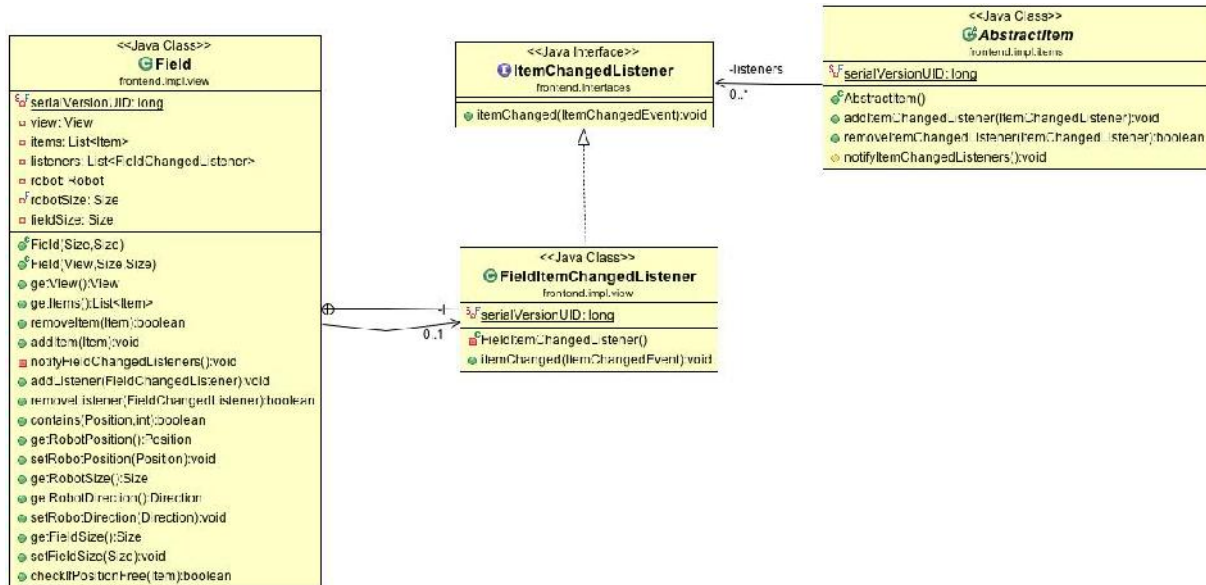


Abbildung 5: UML Observer Hindernisse

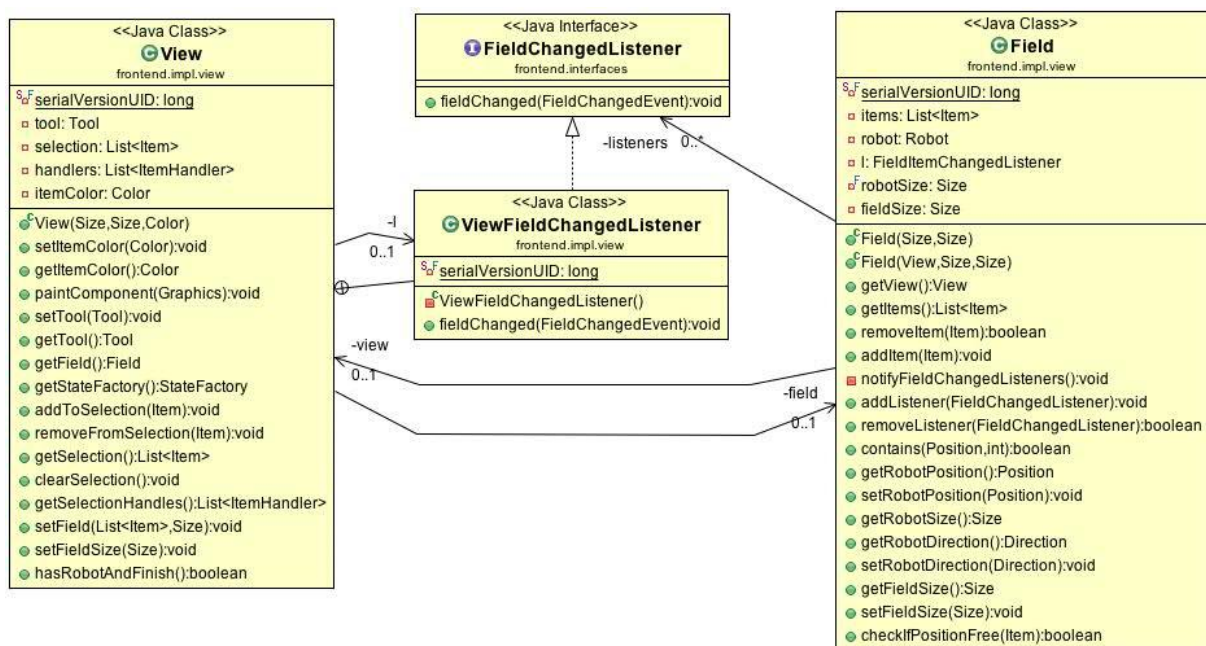


Abbildung 6: UML Observer Field

Das Verschieben und Bearbeiten der Hindernisse verwaltet das Selection Tool. Dieses verarbeitet anhand diverser States (gemäss dem State Pattern) die Mausaktivitäten.

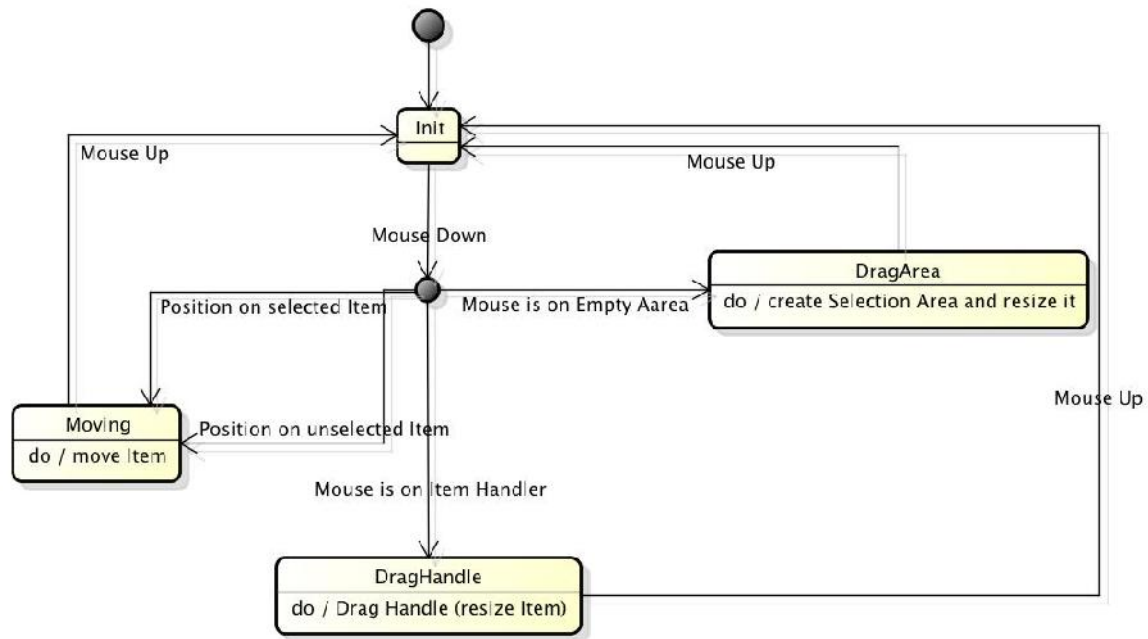


Abbildung 7: Selection State Diagram

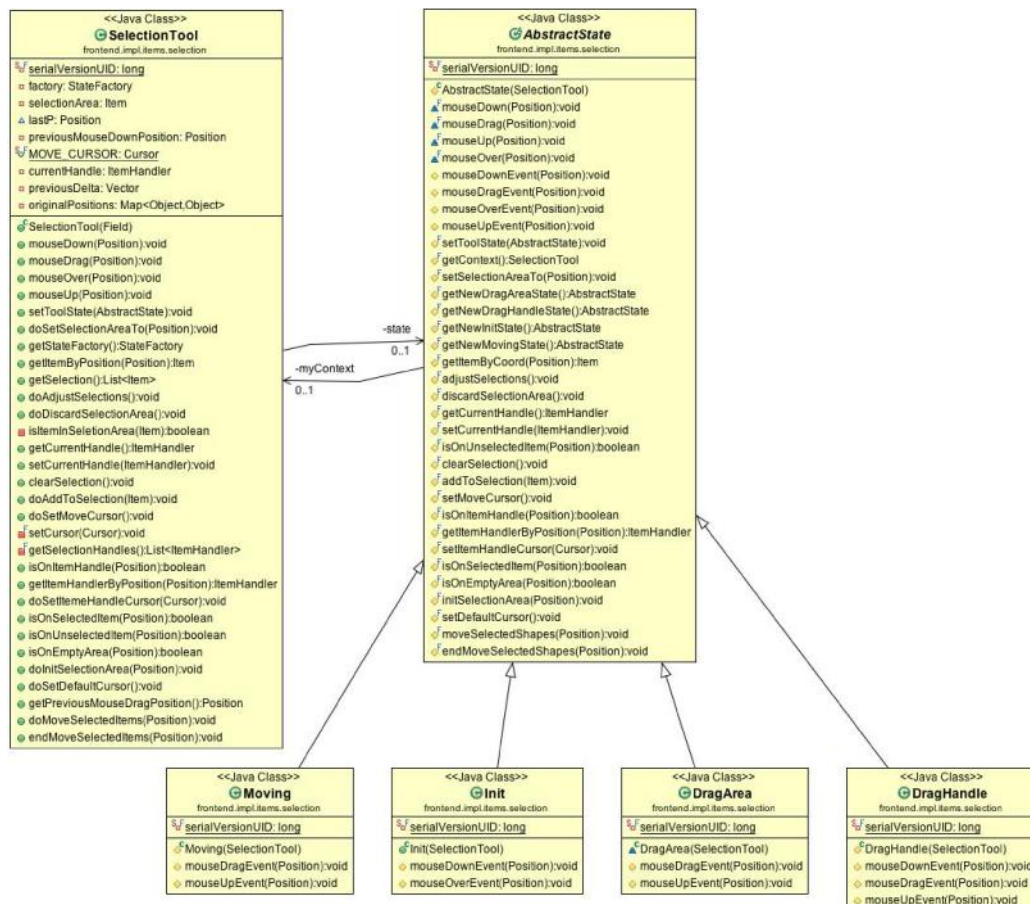


Abbildung 8: UML Selection

Robot

Use Cases in kürze

Für detaillierte Informationen nehmen Sie das Dokument *Use Case Robot* zur Hand.

Ziel suchen

Der Roboter sucht nach einem bestimmten Algorithmus das Spielfeld ab, d.h. er kann sich fortbewegen, jeweils -90° und $+90^\circ$ in Blickrichtung scannen und so das Spielfeld mit den Hindernissen erforschen.

Implementation

Die Robotersuche wird mit Hilfe eines Threads gestartet und falls nötig abgebrochen.

Der Algorithmus unserer Robotersuche funktioniert nach folgendem Sequenz-Diagramm. Erst wird die Umgebung abgescannt (*scanField*), dann mithilfe einer implementierten Strategie (*computePath*) der Pfad zur nächsten Position berechnet und schlussendlich fährt (*move*) der Roboter zu dieser Position. Das wiederholt sich, bis der Roboter das Ziel erreicht hat oder das ganze Spielfeld abgescannt hat und das Ziel nicht finden konnte (Falls es von Hindernissen umgeben ist).

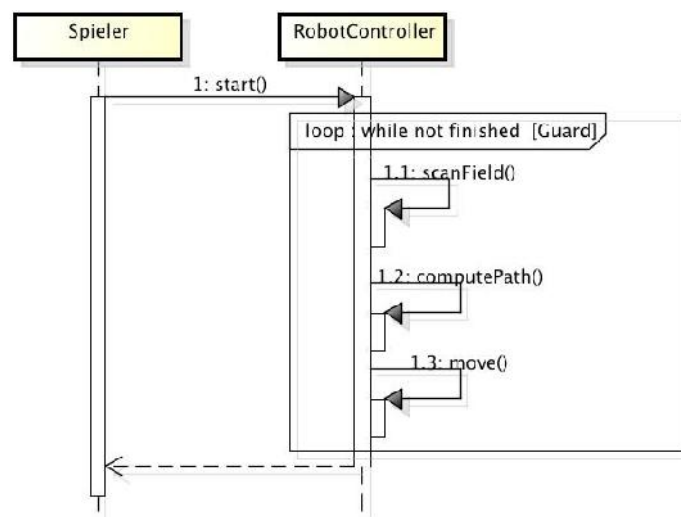


Abbildung 9: SD Algorithmus Robotersuche

Strategie

Um später ev. noch andere Strategien zum Berechnen der Zielsuche zu implementieren, haben wir das Strategy Pattern angewendet.

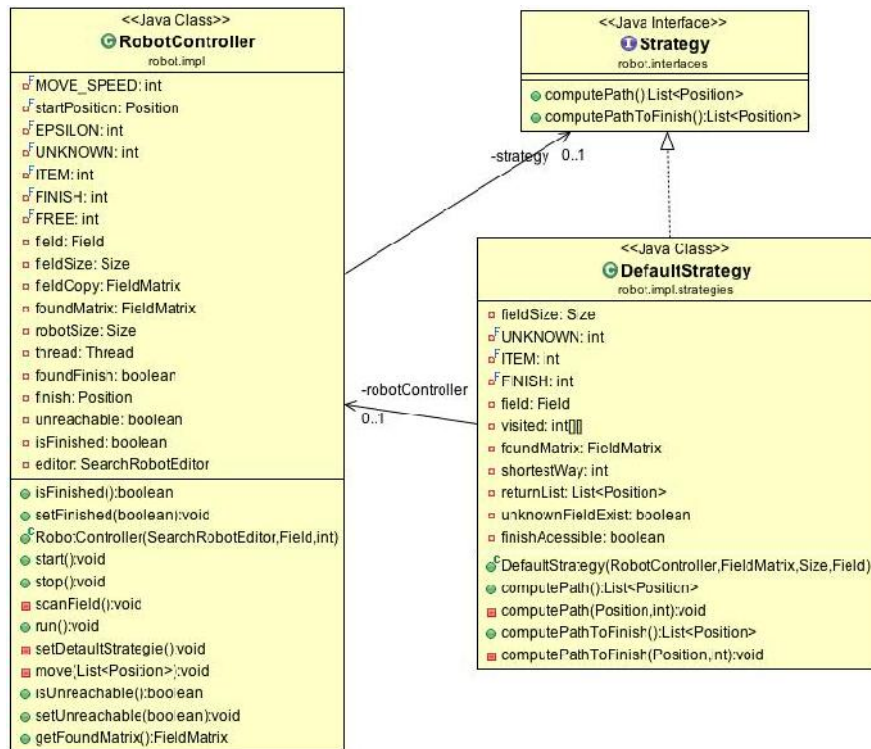


Abbildung 10: UML Strategie

Die Default Strategie, welche wir als effektivste eruiert und dann implementiert haben, sucht jeweils das nahste unbekannte Feld und berechnet den kürzesten Pfad dorthin.

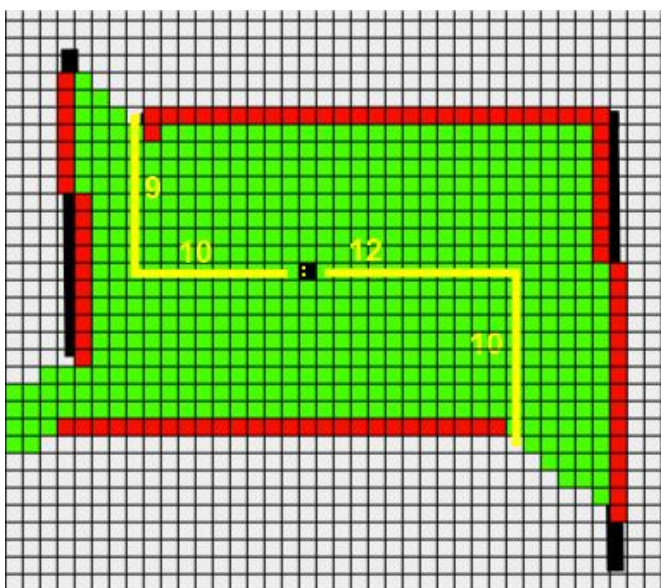


Abbildung 11: Visualisierung Wegberechnung