# MechaWars

# -

# RoboRally in Java programmiert.

**Handbuch erstellt von**

**Tim Michael Kretzschmar**

**(mit freundlicher Unterstützung von Marc André Uxa)**

**Software-Projekt des Jahrgangs 19INB der HTWK Leipzig**

# Inhaltsverzeichnis

# Kapitel 1

# Einleitung

# Kapitel 2

# Die Packages

## 2.1 htwk.mechawars

### 2.1.1 ConfigReader.java

In dieser Klasse werden alle Grundeinstellungen aus der configDatei eingelesen und in Variablen gespeichert, welche alle über Getter-Funktionen verfügen und somit für den Programmablauf verfügbar gestellt werden.
Da man die Spieler-Anzahl ändern kann z.B. existiert eine Funktion, um diese Anzahl in der SStartupconfig.txtßu manipulieren.

### 2.1.2 GameOverScreen.java

Dies ist der Bildschirm, welcher erscheint, wenn man verliert. Dabei besteht dort die Möglichkeit, wieder zum *Startbildschirm/Hauptmenü* zurückzukehren oder das Spiel zu beenden.

### 2.1.3 MainMenue.java

Hier wird die GUI (Graphical User Interface $\widehat{=}$ Graphische Benutzeroberfläche) für den Startbildschirm verwaltet. So sind z.B. im Konstruktor die zwei Buttons hier gelistet mit ihren Funktionen - Weiter zum Auswahlbildschirm und Beenden des Spiels.

Ebenfalls wir hier in Zeile 32 der Hintergrund festgelegt, welcher in Zeile 102 dann gerendert wird in der von LibGDX mitgelieferten render(float delta)-Funktion.
Die Klasse implementiert *Screen* von LibGDX, womit als Interface eine ganze Menge Funktionen eingebunden werden, die jedoch zum großen Teil unverändert dastehen. Somit also von LibGDX selbst verwaltet/standartisierd sind.

### 2.1.4 MechaWars.java

Diese Klasse *extends Game*, ist also von der LibGDX Klasse abhängig. Besonderes Augenmerk sollte man der `create()`-Funktion widmen. Dort wird der *ConfigReader* aufgerufen und anschließend überprüft, ob der *Startbildschirm* aufgerufen werden soll, oder sofort das Spiel mit den Grundeinstellungen gestartet wird..

### 2.1.5 OptionScreen.java

Dieser Bildschirm ist eine umfangreiche Einstellmöglichkeit im Spiel. Hier ist auswählbar ob man im Tainingsmodus oder normalen Spielmodus starten möchte. Ebenfalls die Anzahl der rivalisierenden Roboter und die gewünschte Karte ist hier auswählbar. Sollte keine Map eingetragen sein, so spielt man mit der als Standart hinterlegten Karte.

### 2.1.6 VictoryScreen.java

Das Pandon zum *GameOverScreen.java*. Dieser Bildschirm erscheint, wenn man gewinnt. Dabei besteht dor die Möglichkeit, wieder zum *Startbildschirm/Hauptmenü* zurückzukehren oder das Spiel zu beenden.

## 2.2 htwk.mechawars.ai

### 2.2.1 AiCardGeneration.java

Für den KI-Mitspieler werden hier der LinkedList<Card> Karten herausgenommen und zurück-übergegeben.

### 2.2.2 AiInterface.java (Interface)

Dieses Interface organisiert alle KI-Klassen, so dass sie eine Menge an Roboter haben, ein Board und die Funktion generateCards().

### 2.2.3 AiManager.java

Hier werden alle KI (-Objekte) verwaltet. Import und Export (Speichern) erfolgen ebenfalls hier.

## 2.3 htwk.mechawars.board

### 2.3.1 Board.java

In dieser Klasse wird das Spielfeld als board (engl. = (Spiel-)Brett) angelegt. Dazu gibt es dort beim Erstellen die Pflicht, einen String, ursprünglich aus einer Datei, als Einträge zu übergeben. Diese werden dort dann interpretiert und auf Fehler wie zu lang oder zu kurz überprüft.
Über die `toString`-Funktion kann das aktuelle Board als String wieder ausgelesen werden.
Weiterhin wird hier das Bewegen eines Roboters über die `move`-Funktion ausgeführt.
Es finden sich noch weitere Funktionen, deren Funktionalität über den Funktionen als Kommentar nachzulesen sind, wie der Laser-Check z.B.

### 2.3.2 Dir.java (Enumeration)

Eine Enumeration für die Codierung der Richtung eines Roboters. Dabei gibt es eine `North` (Norden), `East` (Osten), `South` (Süden) und `West` (Westen). Dies wird als Integer-Wert abgespeichert und über die dazu erstellte Funktion *intToDirection()* dann als `Dir` zurückgegeben bei Aufruf oder Bedarf.

### 2.3.3 Robot.java

Diese Klasse beschreibt einen Roboter. In ihr werden essentielle Attribute gespeichert, auszugsweise wären das die Richtung des Roboters (*dir*), die aktuelle Position (`xcoor` und `ycoor`), die Lebenspunkte (`damagePoints`) oder der aktuelle Zerstörungsstatus (boolean `destroyed`).
Es gibt noch eine Funktion neben dem Konstruktor, der die übergebene Anzahl an Robotern erstellt, als Robot[]-Array.
Die Bewegung des Roboters wird ebenfalls hier definiert, genauso wie eine Drehung abzulaufen hat.
Neben den ganzen Getter- und Setter-Funktionen gibt es auch noch weitere Funktionen, wie die *addCard()*-Funktion um eine weitere *Karte* hinzuzufügen.

## 2.4 htwk.mechawars.cards

### 2.4.1 Card.java

Diese Klasse enthält das Gerüst für Karten. Deshalb sind dort die drei Attribute `cardAttributeType`, `cardAttributeMovCount` und `cardAttributePriority` angelegt.
Das erste beschreibt als Enumeration die Art der Karte (siehe *Type.java*)
Das zweite Attribut dagegen ist ein ”byteünd sagt aus, wie viele Bewegungen der Art ausgeführt werden sollen. Dies ist durch folgende Tabelle codiert:

| *byte* | *mov (Bewegung)* | *turn (Drehung)* |
|---|---|---|
| -1 | bewege dich ein Feld zurück | tue nichts (nicht definiert) |
| 1 | bewege dich ein Feld nach vorne | Führe eine Rechtsdrehungen aus |
| 2 | bewege dich zwei Feld nach vorne | Führe zwei Rechtsdrehungen aus, also drehe dich um |
| 3 | bewege dich drei Feld nach vorne | Führe drei Rechtsdrehungen aus, also eine Linksdrehung |

Das dritte und letzte Attribut definiert die Priorität, da jede Karte in RoboRally/MechaWars eine Priorität in seiner Abarbeitung unter den Robotern besitzt.
Jedes Attribut hat ebenfalls einen Getter in der Card.java. Einen Setter benötigt man nicht, da im Konstruktor einer Karte
**Card(*Type* cardAttributeType, *byte* cardAttributeMovCount, *int* cardAttributePriority)**
also beim Erstellen einer Karte die Attribute angelegt werden und anschließend nie verändert werden müssen. Ist im Programmablauf die Erstellung einer oder einiger Karten nötig, so kann dies ebenfalls so geschehen und in einer ArrayList<Card> z.B. zwischengespeichert werden.
Als letzte Funktion ist noch eine `toString`-function definiert, damit die Abfrage der

Enumerations einen String zurückliefert und die Auswertung sowohl bei Bewegung als auch Drehung effektiv abläuft.
Hierzu kann folgende Tabelle zum weiteren Verständnis helfen.

| *byte* | *mov (Bewegung)* | *turn (Drehung)* |
|---|---|---|
| -1 | Rueckwaerts” | siehe default |
| 1 | ”1 Vor” | Rechtsdrehung” |
| 2 | ”2 Vor” | ”Kehrtwendung” |
| 3 | ”3 Vor” | Linksdrehung” |
| default | Fehler | Fehler |

### 2.4.2 Deck.java

Die Deck-Klasse beinhaltet zunächst einen Konstruktor, welcher bei Aufruf die Funktion `initDeck()` wiederum aufruft. Dort wird, wie der Name schon vermuten lässt, ein Deck initialisert, welches als ArrayList<Card> in der Klasse existiert. Das Schema, nach welchem die Karten erstellt werden, lässt sich unter folgendem Link ausführlich aufgeführt nachlesen:
https://boardgamegeek.com/thread/645061/need-specific-list-cards
Weiterhin findet man in der Klasse die Funktion `shuffel()`. Beim Aufruf dieser wird die ArrayList<Card> `deck` gemischelt (engl. shuffle = mischeln).
Und schließlich ist noch die Getter-Funktion vorhanden. Diese ermöglicht den Zugriff auf die ArrayList<Card> `deck`. Ein Setter ist nicht nötig, da das Deck nicht verändert werden muss.
Die ArrayList<Card> `deck` beinhaltet Einträge vom Typ *Card*.

### 2.4.3 Type.java (Enumeration)

In dieser Enumeration sind die Typen einer Karte definiert, welche in dem Spiel auftreten. Dies wären einerseits die `mov` - abgeleitet von *movement* (engl. = Bewegung) - und andererseits der `turn` (engl. *turn* = Drehung).

## 2.5 htwk.mechawars.field

[Hinweis] Die wichtigste Klasse hier ist die *Field.java*. Alle anderen Felder erben von dieser.
Eine Ausnahme stellen da die BarrierCorner und BarrierSide, da diese über Variablen als Felder mit Schranken anzusehen sind.

### 2.5.1 BlackHole.java

Dieses Feld hat eine `texture` (engl. für ”Textur”).
Sollte dieses Betreten werden, so ist der Roboter verloren. Es symbolisiert ein Schwarzes Loch (engl. black hole).

### 2.5.2 Checkpoint.java

Dieses Feld hat eine `Nummer` und eine `texture` (engl. für ”Textur”).
Diese Art von Feldern dient für die Möglichkeit das Spiel zu gewinnen. nur wenn diese in der Reihenfolge abgelaufen werden, kann das dem Spieler gelingen.

### 2.5.3 ConveyorBelt.java

Dieses Feld hat einen Start- und End-Wert als Number (int) und eine `texture` (engl. für ”Textur”).
Sobald ein Roboter in den Genuss dieses Feldes kommt, wird er weiter transportiert in die Richtung, in die es zeigt. Sollte er von einem anderen Fließband daraufgelangen, so wird er lediglich bewegt und nicht gedreht.

### 2.5.4 ExpressConveryorBelt.java

Ein *ConveyorBelt*/Fließband mit erhöhter Priorität, wird also eher ausgeführt als ein normales *ConveyorBelt*/Fließband beim Zugende/anfang.

### 2.5.5 Field.java (Vererbungsklasse)

### 2.5.6 Gear.java

### 2.5.7 Laser.java

### 2.5.8 RepairSite.java

### 2.5.9 StandardField.java

### 2.5.10 StartField.java

## 2.6 htwk.mechawars.game

### 2.6.1 Buttons.java

Buttons, zu deutsch ”Knöpfe”, sind im Spiel das wichtigste Element für die Rundenvorbereitung. Aber auch zum Starten und Beenden des Spiels existieren welche und eben jene sind alle in dieser Klasse organisiert. Somit ist dank dieser Auslagerung eine bessere Übersichtlichkeit der Bildschirm-Klassen gewährleistet.
Näheres zu den einzelnen Buttons findet man im Kommentar über der Funktion.

### 2.6.2 GameScreen.java

### 2.6.3 Info.java

Diese Klasse dient hauptsächlich zum Auslagern von einem großem Textanteil. Sollte der/die Spieler(in) während des Spiels Fragen zu den Karten haben, oder noch etwas nachlesen wollen, so kann dies über den INFO-Button geschehen. Der dort aufgehende *Dialog* wird inhaltlich in dieser Info.java Klasse verwaltet.

### 2.6.4 ScrollPanel.java

Hier wird ein ScrollPanel verwaltet, welches für die Auswahl der Karten in einer Runde zuständig ist. Dabei finden sich Funktionen hier wie `cardOrderClear()` zum Beispiel, womit bei Betätigen des zuständigen Buttons die interne Kartenreihenfolge gelöscht wird. Oder eine optische Funktion wie die `buttonsClean()`, wo jede Karte (ebenfalls durch einen Button dargestellt) wieder ihre Ursprungsfarbe zugewiesen bekommt. Der Zugriff auf das gesamte Kartendeck wird ebenso in dieser Klasse realisiert, siehe Funktionen `getDeck()` und `setDeck()`.

## 2.7 htwk.mechawars.desktop

Hier ist das Herzstück des Programms.

Die main-Funktion des gesamten Programms befindes sich hier.

# Kapitel 3

# Die Tests

## 3.1 (default package)

In diesem Package findet man verschiedene Klassen, die jeweils Kernfunktionalitäten überprüfen, wie z.B. der Board-Test. Dort wird das einlesen eines Boards simuliert und überprüft.

## 3.2 cardtests

Um die Funtkionalität der Karten und somit dem Deck und darausfolgend wiederrum dem ganzen Spiel. Getestet werden z.B. ob die einzelnen Karten im Deck nur die 4 möglichen Bewegungswerte haben oder ob die Priorität ausgewählter Karten korrekt ist.
Es befinden sich auch eine Misch-Test-Funktion (engl. shuffle). Diese ist zwar mit Vorsicht zu genießen, aber die Wahrscheinlichkeit, dass das Mischen genau die selbe Kombination erzeugt liegt bei 84! (Fakultät). Das wäre – 1:3’314’240’134’565’353’266’999’387’579’130’131’288’000’666’286’242’049’487’118’846’032’383’059’131’291’716’864’129’885’722’968’716’753’156’177’920’000’000’000’000’000’000 – und somit sehr unwahrscheinlich.

# Kapitel 4

# assets

Hier befindet sich eine Auflistung aller graphischen Elemente.

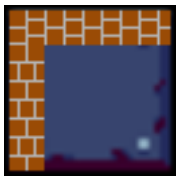
- Roboter

- Standard-Feld

- Start-Felder

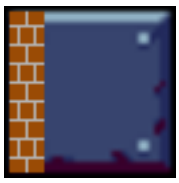

Beispielbild. Es gibt weiter Arten

- Eck-Blockaden (BarrierCorner)



Beispielbild. Es gibt weiter Arten

- Seiten-Blockaden (BarrierSide)



Beispielbild. Es gibt weiter Arten

- Schwarzes-Loch (BlackHole)

- Ziel-Punkte (Checkpoints)

Beispielbild. Es gibt weiter Arten

- Fließband (ConveyorBelt)

Beispielbild. Es gibt weiter Arten

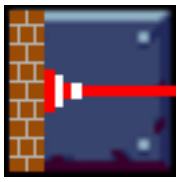
- Express-Fließband (Express-ConveyorBelt)

Beispielbild. Es gibt weiter Arten

- Zahnrad (Gear)

Beispielbild. Es gibt weiter Arten

- Laser



Beispielbild. Es gibt weiter Arten

- Drücker (Pusher)


Beispielbild. Es gibt weiter Arten

- Reperaturfeld (RepairSite)

- Backup-Feld (Checkpoint (Backup))

# Kapitel 5

# Bedienung - User/Anwender

Startet man das Spiel, so kann man im Startbildschirm auf Spiel starten klicken und somit die Mapauswahl, die Anzahl der Spieler (Jeder über 1 entspricht einem KI-Spieler) und der Map (Karte). Sollte keine expliziete Karte eingegeben werden, so wird automatisch die Standardkarte ausgewählt.
Anschließend kann man wiederrum das Spiel beginnen.
Im Spielbildschirm angekommen, sieht man links das Spielfeld (=board) und rechts eine Anzahl an buttons. Diese dienen zur Interaktion. So findet man im Schwarz unterlegten Bereich die Spielkarten des Spielers z.B.
Für eine Spielanleitung bitte folgende Webseite konsulieren:
https://www.yumpu.com/de/document/read/27630938/roborallytm-regelbuch

## Hinzufügen von neuen KIs per Schnittstelle

Das manuelle einfügen von KIs erfolgt über das erstellen und importieren von selbst geschriebenen Klassen.
Dabei müssen zwei Anforderungen erfüllt sein:

- Die KI-Klasse muss das AiInterface implementieren.
- Die KI-Klasse muss sich im htwk.mechawars.ai package befinden.

Das Hinzufügen erfolgt folgendermaßen:

1.: Sicherstellen dass die oben genannten Vorraussetzungen erfüllt sind.

2.: Importieren der KI in der Startupconfig.txt-Datei:

´´Importing AI-Class-files syntax: ’AiImport: Class-file-Name’´´ (Auszug aus configreadme.txt). Man muss die Import-Zeile in der angegebenen Syntax in die Startupconfig.txt schreiben.

3.: Festlegen der KI, die die Spieler nutzen werden:

In der Zeile zum Festlegen der Starpositionen der Spieler, steht die letzte Zahl für den KI-modus des Spielers.

Beispiel: playerposition: 1-3-3-2

Bedeutung: Spieler 1 startet an Position 3,3 und nutzt die zweite KI.

Die Möglichkeiten für den KI-Modus sind wie folgt:

0 - Keine KI: der Spieler bewegt sich nicht.

1 - AiCardGeneration-KI: Standard-KI, die zufällig Karten aus einer Liste wählt.

2-n - die Importierten KIs, in der Reihenfolge in der sie in Schritt 2 importiert wurden. Wenn keine Angabe zum Spieler erfolgt, wird der Spieler keine KI besitzen. Wenn eine zu hohe Zahl angegeben wird, nutzt der Spieler die Standard-KI.

# Kapitel 6

# Konsolen-Modus

### 6.0.1 Einrichtung

Zunächst muss im Ordner `code` die gradlew.bat einmal ausgeführt werden. Anschließend kann in die Kommandozeile folgendes eingegeben werden:

- gradlew.bat desktop:dist

Dadurch wird im (neuerstellten) Unterordner `code/desktop/builds/libs` eine .jar Datei erstellt.

### 6.0.2 Starten

Diese muss nun ausgeführt werden. Dazu ist der Befehl:

- java -jar desktop-1.0.jar

im entsprechenden Verzeichnis einzugeben.

### 6.0.3 Konfiguration(en)

Für die Darstellung aller möglichen Konfigurationen muss in die Kommandozeile wiederrum die Eingabe erfolgen:

- java -jar desktop-1.0.jar -h

Das *h* steht dabei für help (dt. = Hilfe).

Sollte das Spiel im Konsolenmodus gestartet werden und der Befehl -*w* eingetragen werden, so ist das spiel im Fullscreen-Modus (dt. = Vollbild).